Forest-based Algorithms in Natural Language Processing

Liang Huang

overview of Ph.D. work done at Penn (and ISI, ICT)

includes joint work with David Chiang, Kevin Knight, Aravind Joshi, Haitao Mi and Qun Liu

CMU LTI Seminar, Pittsburgh, PA, May 14, 2009
NLP is all about ambiguities

- to middle school kids: what does this sentence mean?

I saw her duck.

Aravind Joshi
NLP is all about ambiguities

- to middle school kids: what does this sentence mean?

I saw her duck.

Aravind Joshi
NLP is all about ambiguities

- to middle school kids: what does this sentence mean?

I eat sushi with tuna.

Aravind Joshi
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.
  • I saw her duck with a telescope in the garden...
NLP is HARD!

- exponential explosion of the search space
- non-local dependencies (context)
Ambiguities in Translation

self help terminal device

needs context to disambiguate!
Evil Rubbish; Safety Export

needs context for fluency!
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 $=>$ 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
• Solution 3: approximate joint search (focus of this talk)
  • (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
  - Forest Rescoring
- Forest-based Translation (Solutions 2+3+1)
  - Tree-based Translation
  - Forest-based Decoding
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
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
  - Forest Rescoring
- Forest-based Translation (Solutions 2+3)
  - Tree-based Translation
  - Forest-based Decoding
k-best Viterbi Algorithm 0

- 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! $\Rightarrow$ 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 **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 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 of hyperedges and vertices](image)
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
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>$O(k \cdot V)$</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>node</td>
<td>$O(E + V \cdot k \log k)$</td>
<td>$O(k \cdot V)$</td>
</tr>
<tr>
<td>Algorithm 3</td>
<td>global</td>
<td>$O(E + D \cdot k \log k)$</td>
<td>$O(E + k \cdot D)$</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

\[ \mathcal{O}(E + D \cdot k \log k) \]
Reranking and Oracles

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

- measures the **potential** of real reranking

![Graph showing Oracle Parseval score for different $k$ values.](image)
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
  - Forest Rescoring
- Application: Forest-based Translation
  - Tree-based Translation
  - Forest-based Decoding
Why not $k$-best reranking?

- 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 ($2^5 < 50 < 2^6$)
Redundancies in n-best lists

Not all those who wrote oppose the changes.
Not all those who wrote oppose the changes.
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 \( y \).

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.

```
TOP
   \[ S \]
   \[ NP \]
       \[ PRP \]
       \[ VBD \]
       \[ I \]
       \[ saw \]
   \[ NP \]
       \[ DT \]
       \[ NN \]
       \[ the \]
       \[ boy \]
   \[ PP \]
       \[ IN \]
       \[ with \]
   \[ NP \]
       \[ DT \]
       \[ NN \]
       \[ a \]
       \[ telescope \]

ParentRule is non-local

Rule is local
```
Local vs. Non-Local: Examples

- **CoLenPar** feature captures the difference in lengths of adjacent conjuncts (Charniak and Johnson, 2005)

```
NP
PRP
They VBD VP and VDB VP were VBN PP were VBN PP consulted IN NP surprised IN NP in NN advance at NP VP

4 words

6 words

CoLenPar: 2
```
Local vs. Non-Local: Examples

- CoPar feature captures the depth to which adjacent conjuncts are isomorphic (Charniak and Johnson, 2005)

CoPar: 4

non-local!

(violates DP principle)
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 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

unit instance of ParentRule feature at VP node

local features factor across hyperedges statically

class 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 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

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 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 statically

non-local features factor across nodes dynamically

Forest Algorithms
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 \textit{dynamically}

local features factor across hyperedges \textit{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
**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.

```
[26x12]Forest  Algorithms
[227x304]TOP
[243x261]S
[74x217]NP
[67x174]PRP
[84x131]I
[247x217]VP
[133x174]VBD
[139x131]saw
[222x174]NP
[195x131]DT
[196x88]the
[248x88]NN
[247x88]boy
[355x174]PP
[311x131]IN
[303x88]with
[400x88]NP
[366x88]DT
[375x45]a
[433x88]NN
[409x45]telescope
[406x217].
[406x174].
```
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
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
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

$$\mathbf{w} \cdot f_N(\ldots) = 0.5$$
Approximate Decoding

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

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

<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

- 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>$w_i \ldots w_{j-1}$</th>
<th>$w_j \ldots w_{k-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{i,k}$</td>
<td></td>
</tr>
<tr>
<td>$B_{i,j}$</td>
<td>$C_{j,k}$</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>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

\[ \mathbf{w} \cdot \mathbf{f}_N(\quad) = 0.5 \]
Algorithm 2 => Cube Pruning

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

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

<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

![Graph showing Parseval F-score vs. average # of hyperedges or brackets per sentence.](chart)

- $p=10$ with $n=10$: 97.8
- $p=20$, $n=10$: 98.6
- $n=50$, $n=100$: 96.7, 97.2
- 1-best
- $n$-best oracle
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 beats 50-best & 100-best reranking
- can be trained on the whole treebank in ~1 day even with a pure Python implementation!
- most previous work only scaled to short sentences (<=15 words) and local features

<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>
## Main Results

- forest reranking beats 50-best & 100-best reranking
- can be trained on the whole treebank in ~1 day even with a pure Python implementation!
- most previous work only scaled to short sentences (<=15 words) and local features

<table>
<thead>
<tr>
<th>approach</th>
<th>training time</th>
<th>F1%</th>
<th>space</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline: 1-best Charniak parser</td>
<td></td>
<td>89.72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50-best reranking</td>
<td>4 x 0.3h</td>
<td>91.43</td>
<td>2.4G</td>
<td>19h</td>
</tr>
<tr>
<td>100-best reranking</td>
<td>4 x 0.7h</td>
<td>91.49</td>
<td>5.3G</td>
<td>44h</td>
</tr>
<tr>
<td>forest reranking</td>
<td>4 x 6.1h</td>
<td>91.69</td>
<td>1.2G</td>
<td>2.9h</td>
</tr>
</tbody>
</table>
## 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>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></td>
<td>Carreras et al. (2008)</td>
<td>91.1</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, and fast training*
on to Machine Translation...

applying the same ideas of non-locality...
clear evidence that MT is used in real life.
Context in Translation

![Signage with Chinese characters and English translations](image1)

- "有毒有害垃圾" (Poisonous & Evil Rubbish)
- "小心滑落" (Slip carefully)

---

![Signage with additional text](image2)
Algorithm 2 => cube pruning
fluency problem (n-gram)
Algorithm 2 => cube pruning
fluency problem (n-gram)

xiaoxin  小心  X  <=>  be careful not to X
syntax problem (SCFG)
Context in Translation

Algorithm 2 => cube pruning

fluency problem (n-gram)

syntax problem (SCFG)
Context in Translation

Algorithm 2 => cube pruning
fluency problem (n-gram)

syntax problem (SCFG)
How do people translate?

1. understand the source language sentence

2. generate the target language translation

布什 与 沙龙 举行 了 会谈

Bush and/with Sharon hold [past.] meeting

Bush 沙龙 举行 了 会面

Bùshí yu Shalóng juxíng le huìtán
How do people translate?

1. understand the source language sentence
2. generate the target language translation

布什 与 沙龙 举行 了 会谈
Bùshí yu Shálóng júxíng le huìtán
Bush and/with Sharon hold [past.] meeting
How do people translate?

1. understand the source language sentence

2. generate the target language translation

布什 与 沙龙 举行 了 会谈

Bùshí  yu  Shálóng  juxíng  le  huìtán

Bush  and/ with  Sharon  hold  [past.]  meeting

“Bush held a meeting with Sharon”
How do compilers translate?

1. parse high-level language program into a syntax tree
2. generate intermediate or machine code accordingly

\[ x_3 = y + 3; \]
How do compilers translate?

1. parse high-level language program into a syntax tree
2. generate intermediate or machine code accordingly

```plaintext
x3 = y + 3;
```
How do compilers translate?

1. parse high-level language program into a syntax tree
2. generate intermediate or machine code accordingly

```plaintext
x3 = y + 3;
```

LD     R1,  id2
ADDF   R1,  R1, #3.0  // add float
RTOI   R2,  R1        // real to int
ST     id1, R2
How do compilers translate?

1. parse high-level language program into a syntax tree
2. generate intermediate or machine code accordingly

```plaintext
x3 = y + 3;
```

LD     R1,  id2
ADDF   R1,  R1, #3.0  // add float
RTOI   R2,  R1        // real to int
ST     id1, R2

```

syntax-directed translation (~1960)

Compilers
Principles, Techniques, and Tools
Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman

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

“Bush held a meeting with Sharon”
Syntax-Directed Machine Translation

- recursive rewrite by pattern-matching

Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006

(Bùshí yǔ Shālóng jǔxíng le huìtán)
Syntax-Directed Machine Translation

- recursive rewrite by pattern-matching

(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)
Syntax-Directed Machine Translation

- recursive rewrite by pattern-matching

(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)
Recursive rewrite by pattern-matching (Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)
Syntax-Directed Machine Translation

- recursively solve unfinished subproblems

Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006

NPB

| Bùshí
VV

| jǔxíng
AS

| le

| huìtáń

with

NPB

| Shālóng
Syntax-Directed Machine Translation

- recursively solve unfinished subproblems

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

with

NPB  |  Shālóng
```
Syntax-Directed Machine Translation

- recursively solve unfinished subproblems

Bush

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

with

```
NPB

Shālóng
```

```latex
\begin{align*}
&\text{VPB} \\
&\text{VV \ AS \ NPB} \\
&jǔxíng \ le \ huìtán
\end{align*}
```
Syntax-Directed Machine Translation

- recursively solve unfinished subproblems

Bush

\[
\begin{align*}
\text{VPB} & \quad \text{with} \\
\text{VV} \quad \mid jǔxíng & \quad \text{NPB} \\
\text{AS} \quad \mid le & \quad \text{huìtán} \\
\text{held} & \\
\end{align*}
\]

(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)
Syntax-Directed Machine Translation

- continue pattern-matching

Bush held NPB with NPB
  | huìtán
  | Shālóng
continue pattern-matching

Bush held a meeting with Sharon

NPB | huìtán

NPB | Shālóng

(Galley et al. 2004; Liu et al., 2006; Huang, Knight, Joshi 2006)
Syntax-Directed Machine Translation

- continue pattern-matching

Bush held a meeting with Sharon
Syntax-Directed Machine Translation

- continue pattern-matching

Bush held a meeting with Sharon.

This method is simple, fast, and expressive. But... crucial difference between PL and NL: ambiguity!

Using 1-best parse causes error propagation!

Idea: use k-best parses?

Use a parse forest!
Forest-based Translation

“and” / “with”
Forest-based Translation

pattern-matching on forest

“and” / “with”
Forest-based Translation

pattern-matching on forest

“and” / “with”
Forest-based Translation

pattern-matching on forest

“and” / “with”
Forest-based Translation

pattern-matching on forest

“and” / “with”
Forest-based Translation

pattern-matching on forest

“and” / “with”
Forest-based Translation

pattern-matching on forest

“and” / “with”
Forest-based Translation

pattern-matching on forest
directed by underspecified syntax

“and” / “with”
Translation Forest

The diagram illustrates a hierarchical structure with nodes labeled as IP, NPB, CC, VP, and P, connected by edges labeled with numbers and symbols. The diagram shows relationships between these nodes, possibly indicating a flow or transformation process. The specific labels and symbols suggest a complex algorithmic or computational process, likely related to forest algorithms as indicated by the title.
Sharon held a meeting with Bush.
“Bush held a meeting with Sharon”
The Whole Pipeline

input sentence → parser → parse forest → translation forest → translation+LM forest

- pattern-matching w/ translation rules (exact)
- Algorithm 2 => cube pruning (approx.)
- Algorithm 3 (exact)

- best derivation
- 1-best translation

1-best translation

k-best translations

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

input sentence

parser

parse forest

pattern-matching w/ translation rules (exact)

Algorithm 2 => cube pruning (approx.)

Algorithm 3 (exact)

best derivation

translation forest

packed forests

translation+LM forest

l-best translation

k-best translations

(Huang and Chiang, 2005; 2007; Chiang, 2007)
$k$-best trees vs. forest-based

1.7 Bleu improvement over 1-best, 0.8 over 30-best, and even faster!
forest as virtual $\infty$-best list

- how often is the $i^{th}$-best tree picked by the decoder?

![Graph showing the percentage of sentences (in %) against the rank of the tree picked in the n-best list. The graph compares 30-best trees and forest decoding. There are annotations suggesting that 32% go beyond the 100-best, and 20% beyond the 1000-best. The graph is suggested by Mark Johnson.](image-url)
Larger Decoding Experiments

- 2.2M sentence pairs (57M Chinese and 62M English words)
- larger trigram models (1/3 of Xinhua Gigaword)
- also use **bilingual phrases** (BP) as flat translation rules
- phrases that are consistent with syntactic constituents
- forest enables larger improvement with BP

<table>
<thead>
<tr>
<th></th>
<th>T2S</th>
<th>T2S+BP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-best tree</td>
<td>0.2666</td>
<td>0.2939</td>
</tr>
<tr>
<td>30-best trees</td>
<td>0.2755</td>
<td>0.3084</td>
</tr>
<tr>
<td>forest</td>
<td>0.2839</td>
<td>0.3149</td>
</tr>
<tr>
<td>improvement</td>
<td>1.7</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Conclusions: Dynamic Programming

- A general framework of DP on monotonic hypergraphs
- Exact $k$-best DP algorithms (monotonic)
- Approximate DP with non-local features (non-monotonic)
  - Forest Reranking for discriminative parsing
  - Forest Rescoring for MT decoding
- Forest-based Translation
  - translates a parse forest of millions of trees
  - even faster than translating top-30 trees (and better)
- 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

Can not be factored anywhere, have to wait till root.
(punctuation or not is ambiguous: ': possessive or right quote?)

(Charniak and Johnson, 2005)