Decoding and Inference with Syntactic Translation Models

March 5, 2013
CFGs

\[ S \rightarrow NP \ VP \]
\[ VP \rightarrow NP \ V \]
\[ V \rightarrow \text{tabeta} \]
\[ NP \rightarrow \text{jon-ga} \]
\[ NP \rightarrow \text{ringo-o} \]
CFGs

\[
S \rightarrow NP \ VP \\
VP \rightarrow NP \ V \\
V \rightarrow tabeta \\
NP \rightarrow jon-ga \\
NP \rightarrow ringo-o
\]
CFGs

S  →  NP  VP
VP  →  NP  V
V  →  tabeta
NP  →  jon-ga
NP  →  ringo-o
CFGs

\[
\begin{align*}
S & \rightarrow NP \ VP \\
VP & \rightarrow NP \ V \\
V & \rightarrow tabeta \\
NP & \rightarrow jon-ga \\
NP & \rightarrow ringo-o
\end{align*}
\]
CFGs

S → NP VP
VP → NP V
V → tabeta
NP → jon-ga
NP → ringo-o
CFGs

\[
\begin{align*}
S & \rightarrow \ NP \ VP \\
VP & \rightarrow \ NP \ V \\
V & \rightarrow \ \text{tabeta} \\
NP & \rightarrow \ \text{jon-ga} \\
NP & \rightarrow \ \text{ringo-o}
\end{align*}
\]

Tuesday, March 5, 13
CFGs

\[
S \rightarrow NP \ VP \\
VP \rightarrow NP \ V \\
V \rightarrow \text{tabeta} \\
NP \rightarrow \text{jon-ga} \\
NP \rightarrow \text{ringo-o}
\]
CFGs

S → NP VP
VP → NP V
V → tabeta
NP → jon-ga
NP → ringo-o

Output: jon-ga ringo-o tabeta
Synchronous CFGs

\[ S \rightarrow NP \ VP \]
\[ VP \rightarrow NP \ V \]
\[ V \rightarrow \text{tabeta} \]
\[ NP \rightarrow \text{jon-ga} \]
\[ NP \rightarrow \text{ringo-o} \]
Synchronous CFGs

\[
\begin{align*}
S & \rightarrow \text{NP VP} : [1] [2] & \text{(monotonic)} \\
\text{VP} & \rightarrow \text{NP V} : [2] [1] & \text{(inverted)} \\
V & \rightarrow \text{tabeta} : \text{ate} \\
\text{NP} & \rightarrow \text{jon-ga} : \text{John} \\
\text{NP} & \rightarrow \text{ringo-o} : \text{an apple}
\end{align*}
\]
Synchronous CFGs

\[
S \rightarrow \text{NP VP} : \begin{array}{c} 1 \\ 2 \end{array} \quad \text{(monotonic)}
\]

\[
VP \rightarrow \text{NP V} : \begin{array}{c} 2 \\ 1 \end{array} \quad \text{(inverted)}
\]

\[
V \rightarrow \text{tabeta} : \text{ate}
\]

\[
\text{NP} \rightarrow \text{jon-ga} : \text{John}
\]

\[
\text{NP} \rightarrow \text{ringo-o} : \text{an apple}
\]
Synchronous generation
Synchronous generation

S ———— S
Synchronous generation
Synchronous generation

S

NP
jon-ga

VP

S

NP
John

VP
Synchronous generation

S
  NP  VP
  jon-ga  NP  V
  
S
  NP  VP
  John  V  NP
Synchronous generation

S
  NP      VP
  |      |
jon-ga  NP
      |      V
      |      |
ring-o

S
  NP      VP
  |      |
John   V
      |
      NP
      |
an apple
Synchronous generation

S
  \----\-
  NP  VP
    \----\-
    jon-ga  NP
             \----\-
             V  ringo-o
             \----\-
             tabeta

S
  \----\-
  NP  VP
    \----\-
    John  V
             \----\-
             ate  NP
             \----\-
             an  apple
Synchronous generation

Output: (jon-ga ringo-o tabeta : John ate an apple)
Translation as parsing

Parse source

jon-ga  ringo-o  tabeta
Translation as parsing

Parse source

NP

jon-ga  ringo-o  tabeta
Translation as parsing

Parse source

NP  |  NP
---  |  ---
jon-ga  |  ringo-o  tabeta
Translation as parsing

Parse source

NP  |  NP  |  V
jon-ga  |  ringo-o  |  tabeta
Translation as parsing

Parse source

VP

NP  NP  V
jon-ga  ringo-o  tabeta
Translation as parsing

Parse source

S
  VP
    NP   NP    V
    jon-ga ringo-o tabeta
Translation as parsing

Parse source

Project to target

```
S
  /\  \
/  \ /  \ 
NP VP V NP
jon-ga ringo-o tabeta

S
  /\  \
/  \ /  \ 
NP VP V NP
John ate an apple
```
A closer look at parsing

- Parsing is usually done with dynamic programming
  - **Share common computations and structure**
  - Represent exponential number of alternatives in polynomial space
- With SCFGs there are two kinds of ambiguity
  - source parse ambiguity
  - translation ambiguity
  - parse forests can represent both!
A closer look at parsing

- Any monolingual parser can be used (most often: CKY / CKY variants)
- Parsing complexity is $O(|n^3|)$
  - cubic in the length of the sentence ($n^3$)
  - cubic in the number of non-terminals ($|G|^3$)
- adding nonterminal types increases parsing complexity substantially!
- With few NTs, exhaustive parsing is tractable
Parsing as deduction

“If $A$ and $B$ are true with weights $u$ and $v$, and phi is also true, then $C$ is true with weight $w$.”
Example: CKY

Inputs:

\[ f = \langle f_1, f_2, \ldots, f_e \rangle \]

\[ G \] Context-free grammar in Chomsky normal form.

Item form:

\[ [X, i, j] \] A subtree rooted with NT type \( X \) spanning \( i \) to \( j \) has been recognized.
Example: CKY

Goal:

\[ [S, 0, \ell] \]

Axioms:

\[
\frac{[X, i - 1, i] : w}{(X \xrightarrow{w} f_i) \in G}
\]

Inference rules:

\[
\frac{[X, i, k] : u \quad [Y, k, j] : v}{[Z, i, j] : u \times v \times w \quad (Z \xrightarrow{w} XY) \in G}
\]
I saw her duck.
I saw her duck
I saw her duck.
I saw her duck.
I saw her duck.
I saw her duck.
S → PRP VP
VP → V NP
VP → V SBAR
SBAR → PRP V
NP → PRP NN
V → saw
NN → duck
V → duck
PRP → I
PRP → her

I saw her duck
I saw her duck.
I saw her duck.
I saw her duck
I saw her duck.
I saw her duck.
Semantics of hypergraphs

- Generalization of directed graphs
- Special node designated the “goal”
- Every edge has a single head and 0 or more tails (the \textit{arity} of the edge is the number of tails)
- Node labels correspond to LHS’s of CFG rules
- A \textit{derivation} is the generalization of the graph concept of \textit{path} to hypergraphs
- Weights multiply along edges in the derivation, and add at nodes (cf. \textit{semiring parsing})
Edge labels

• Edge labels may be a mix of terminals and substitution sites (non-terminals)

• In translation hypergraphs, edges are labeled in both the source and target languages

• The number of substitution sites must be equal to the arity of the edge and must be the same in both languages

• The two languages may have different orders of the substitution sites

• There is no restriction on the number of terminal symbols
Edge labels

\{(la lectura de ayer : yesterday's reading), (la lectura de ayer : reading from yesterday)\}
Inference algorithms

- Viterbi \( O(|E| + |V|) \)
  - Find the maximum weighted derivation
  - Requires a partial ordering of weights
- Inside - outside \( O(|E| + |V|) \)
  - Compute the marginal (sum) weight of all derivations passing through each edge/node
- \( k \)-best derivations \( O(|E| + |D_{max}|k \log k) \)
  - Enumerate the \( k \)-best derivations in the hypergraph
- See IWPT paper by Huang and Chiang (2005)
Things to keep in mind

Bound on the number of edges:
\[ |E| \in O(n^3|G|^3) \]

Bound on the number of nodes:
\[ |V| \in O(n^2|G|) \]
Decoding Again

- Translation hypergraphs are a “lingua franca” for translation search spaces
- Note that FST lattices are a special case
- Decoding problem: how do I build a translation hypergraph?
Representational limits

Consider this very simple SCFG translation model:

“Glue” rules:

\[
S \rightarrow S \quad S \quad : \quad 1 \quad 2 \\
S \rightarrow S \quad S \quad : \quad 2 \quad 1
\]
Consider this very simple SCFG translation model:

"Glue" rules:

\[
S \rightarrow S \ S \ : \ 1 \ 2
\]

\[
S \rightarrow S \ S \ : \ 2 \ 1
\]

"Lexical" rules:

\[
S \rightarrow \text{tabeta} \ : \ \text{ate}
\]

\[
S \rightarrow \text{jon-ga} \ : \ \text{John}
\]

\[
S \rightarrow \text{ringo-o} \ : \ \text{an apple}
\]
Representational limits

- Phrase-based decoding runs in exponential time
- All permutations of the source are modeled (traveling salesman problem!)
- Typically distortion limits are used to mitigate this
- But parsing is polynomial...what’s going on?
Representational limits

Binary SCFGs cannot model this (however, ternary SCFGs can):

A -- B -- C -- D
  B       D       A

Tuesday, March 5, 13
Representational limits

Binary SCFGs cannot model this (however, ternary SCFGs can):

But can’t we binarize *any* grammar?
Representational limits

Binary SCFGs cannot model this (however, ternary SCFGs can):

But can’t we binarize any grammar?

**No.** Synchronous CFGs cannot generally be binarized!
Does this matter?

• The “forbidden” pattern is observed in real data (Melamed, 2003)
• Does this matter?
  • Learning
    • Phrasal units and higher rank grammars can account for the pattern
  • Sentences can be simplified or ignored
• Translation
  • The pattern does exist, but how often must it exist (i.e., is there a good translation that doesn’t violate the SCFG matching property)?
Tree-to-string

• How do we generate a hypergraph for a tree-to-string translation model?
  • Simple linear-time (given a fixed translation model) top-down matching algorithm
  • Recursively cover “uncovered” sites in tree
  • Each node in the input tree becomes a node in the translation forest
  • For details, Huang et al. (AMTA, 2006) and Huang et al. (EMNLP, 2010)
\[
S(x_1:NP \ x_2:VP) \rightarrow x_1 \ x_2 \\
VP(x_1:NP \ x_2:V) \rightarrow x_2 \ x_1
\]

*tabeta* \rightarrow ate

*ringo-o* \rightarrow an apple

*jon-ga* \rightarrow John

Tree-to-string grammar
S(x₁:NP  x₂:VP) → x₁  x₂

VP(x₁:NP  x₂:V) → x₂  x₁

tabeta → ate

ringo-o → an apple

jon-ga → John
\[ S(x_1:NP \ x_2:VP) \rightarrow x_1 \ x_2 \]

\[ VP(x_1:NP \ x_2:V) \rightarrow x_2 \ x_1 \]

\textit{tabeta} \rightarrow \textit{ate}

\textit{ringo-o} \rightarrow \text{an apple}

\textit{jon-ga} \rightarrow \textit{John}
S(x₁:NP  x₂:VP) → x₁ x₂

VP(x₁:NP  x₂:V) → x₂ x₁

tabeta → ate

ringo-o → an apple

jon-ga → John
S(x₁:NP x₂:VP) → x₁ x₂
VP(x₁:NP x₂:V) → x₂ x₁

tabeta → ate
ringo-o → an apple
jon-ga → John

Tuesday, March 5, 13
S(x_1:NP  x_2:VP) → x_1  x_2
VP(x_1:NP  x_2:V) → x_2  x_1

\textit{tabeta} → \textit{ate}

\textit{ringo-o} → an apple

\textit{jon-ga} → John
\[ S(x_1:NP \ x_2:VP) \rightarrow x_1 \ x_2 \]
\[ VP(x_1:NP \ x_2:V) \rightarrow x_2 \ x_1 \]
\[ tabeta \rightarrow ate \]
\[ ringo-o \rightarrow an \ apple \]
\[ jon-ga \rightarrow John \]
Language Models
Hypergraph review

la lectura : reading

ayer : yesterday

de : 's

de : from

Source label  Target label

Goal node
Hypergraph review

la lectura : reading

ayer : yesterday

Substitution sites / variables / non-terminals
Hypergraph review

For LM integration, we ignore the source!
Hypergraph review

For LM integration, we ignore the source!
Hypergraph review

How can we add the LM score to each string derived by the hypergraph?
LM Integration

- If LM features were purely local ...
  - “Unigram” model
  - Discriminative LM
- ... integration would be a breeze
  - Add an “LM feature” to every edge
- But, LM features are non-local!
Why is it hard?

Two problems:

1. What is the content of the variables?
Why is it hard?

Two problems:

1. What is the content of the variables?
Why is it hard?

Two problems:

1. What is the content of the variables?
Why is it hard?

Two problems:

1. What is the content of the variables?
Why is it hard?

Two problems:

1. What is the content of the variables?

2. What will be the **left context** when this string is substituted somewhere?
Why is it hard?

Two problems:

1. What is the content of the variables?
2. What will be the left context when this string is substituted somewhere?
Why is it hard?

Two problems:

1. What is the content of the variables?
2. What will be the *left context* when this string is substituted somewhere?
Why is it hard?

Two problems:

1. What is the content of the variables?

2. What will be the **left context** when this string is substituted somewhere?
Naive solution

- Extract the all (k-best?) translations from the translation model
- Score them with an LM
- What’s the problem with this?
Outline of DP solution

• Use $n$-order Markov assumption to help us
  • In an $n$-gram LM, words more than $n$ words away will not affect the local (conditional) probability of a word in context
  • This is not generally true, just the Markov assumption!

• General approach
  • Restructure the hypergraph so that LM probabilities decompose along edges.
  • Solves both “problems”
    • we will not know the full value of variables, but we will know “enough”.
    • defer scoring of left context until the context is established.
Hypergraph restructuring
Hypergraph restructuring

- Note the following three facts:
  - If you know $n$ or more consecutive words, the conditional probabilities of the $n$th, $(n+1)$th, ... words can be computed.
  - Therefore: add a feature weight to the edge for words.
Hypergraph restructuring

Note the following three facts:

- If you know \( n \) or more consecutive words, the conditional probabilities of the \( n \)th, \((n+1)\)th, ... words can be computed.
- Therefore: add a feature weight to the edge for words.
- \((n-1)\) words of context to the left is enough to determine the probability of any word
- Therefore: split nodes based on the \((n-1)\) words on the right side of the span dominated by every node
Hypergraph restructuring

• Note the following three facts:
  • If you know \( n \) or more consecutive words, the conditional probabilities of the \( n \)th, \((n+1)\)th, ... words can be computed.
  • Therefore: add a feature weight to the edge for words.
  • \((n-1)\) words of context to the left is enough to determine the probability of any word
  • Therefore: split nodes based on the \((n-1)\) words on the right side of the span dominated by every node
  • \((n-1)\) words on the left side of a span cannot be scored with certainty because the context is not known
  • Therefore: split nodes based on the \((n-1)\) words on the left side of the span dominated by every node
Hypergraph restructuring

- Note the following three facts:
  - If you know \( n \) or more consecutive words, the conditional probabilities of the \( n \)th, \((n+1)\)th, \( ... \) words can be computed.

Split nodes by the \((n-1)\) words on both sides of the convergent edges.

- \((n-1)\) words on the **left** side of a span cannot be scored with certainty because the context is not known
- Therefore: split nodes based on the \((n-1)\) words on the **left** side of the span dominated by every node
Hypergraph restructuring

- Algorithm (“cube intersection”):
  - For each node $v$ (proceeding in topological order through the nodes)
    - For each edge $e$ with head-node $v$, compute the $(n-1)$ words on the left and right; call this $q_e$
      - Do this by substituting the $(n-1)x2$ word string from the tail node corresponding to the substitution variable
      - If node $vq_e$ does not exist, create it, duplicating all outgoing edges from $v$ so that they also proceed from $vq_e$
      - Disconnect $e$ from $v$ and attach it to $vq_e$
  - Delete $v$
Hypergraph restructuring

0.6  the man
0.4  the husband
0.1  la mancha
0.7  the stain
0.2  the gray stain

0.6  2's 1
0.4  1 from 2
Hypergraph restructuring

-LM Viterbi:
the stain's the man
Hypergraph restructuring

Let’s add a bi-gram language model!

0.6  the man
0.4  the husband
0.1  la mancha
0.7  the stain
0.2  the gray stain

2's 1  0.6
1 from 2  0.4
Hypergraph restructuring

Let's add a bi-gram language model!
Hypergraph restructuring

\[ p(\text{mancha}|\text{la}) \]

- 0.1 \quad \text{la mancha}
- 0.7 \quad \text{the stain}
- 0.2 \quad \text{the gray stain}

\[ \begin{array}{ccc}
0.6 & \text{the man} & \times \\
0.4 & \text{the husband} & \times \\
\end{array} \]

\[ \begin{array}{ccc}
2 \quad \text{'s} & 1 & 0.6 \\
1 \quad \text{from} & 2 & 0.4 \\
\end{array} \]
Hypergraph restructuring

\[ p(\text{mancha}|\text{la}) \]

\[
\begin{align*}
0.6 & \quad \text{the man} \\
0.4 & \quad \text{the husband}
\end{align*}
\]

\[
\begin{align*}
0.1 & \quad \text{la mancha} \\
0.7 & \quad \text{the stain} \\
0.2 & \quad \text{the gray stain}
\end{align*}
\]

\[
\begin{align*}
2 & \quad \text{is} \\
1 & \quad \text{from} \\
0.6 & \\
0.4
\end{align*}
\]
Hypergraph restructuring

\[ p(\text{stain}|\text{the}) = \begin{cases} 
0.6 & \text{the man} \\
0.4 & \text{the husband} \\
0.1 & \text{la mancha} \\
0.7 & \text{the stain} \\
0.2 & \text{the gray stain} 
\end{cases} \]
Hypergraph restructuring

\[
p(\text{stain}|\text{the})
\]

- 0.6
- 0.4

- the man
- the husband

- la mancha

- the stain
- the gray stain

- 0.1
- 0.7
- 0.2

- 2 's 1
- 1 from 2

- 0.6
- 0.4
Hypergraph restructuring

p(gray|the) \times p(stain|gray)

0.1
la mancha

0.2
the gray stain

0.4
the stain

0.6
the man

0.7
the husband

0.6
2's 1

1 from 2
0.4
X

X

X

X

X
Hypergraph restructuring

\[ p(\text{gray|the}) \times p(\text{stain|gray}) \]

\[ \text{the man} \]
\[ \text{the husband} \]
\[ \text{la mancha} \]
\[ \text{the stain} \]
\[ \text{the gray stain} \]
Hypergraph restructuring

[Diagram with nodes and edges labeled with terms and values]

- la mancha
- the stain
- the gray stain
- the man
- the husband

Values:
- 0.6
- 0.4
- 0.1
- 0.7
- 0.2
Hypergraph restructuring
Hypergraph restructuring

- the man
- the husband
- la mancha
- the stain
- the gray stain

Tuesday, March 5, 13
Hypergraph restructuring

Every node “remembers” enough for edges to compute LM costs
Complexity

• What is the run-time of this algorithm?
Complexity

• What is the run-time of this algorithm?

\[ O(|V||E||\Sigma|^{2(n-1)}) \]

Going to longer n-grams is exponentially expensive!
Cube pruning

• Expanding every node like this exhaustively is impractical
  • Polynomial time, but really, really big!

• Cube pruning: minor tweak on the above algorithm
  • Compute the k-best expansions at each node
  • Use an **estimate** (usually a unigram probability) of the unscored left-edge to rank the nodes
Cube pruning

- Why “cube” pruning?
  - Cube-pruning only involves a “cube” when arity-2 rules are used!
  - More appropriately called “square” pruning with arity-1
  - Or “hypercube” pruning with arity > 2!
Cube Pruning

monotonic grid?

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
</table>
| (VP \text{held} \ast \text{meeting}) \_
\_3,6 | 1.0 | 2.0 | 4.0 | 9.0 |
| (VP \text{held} \ast \text{talk}) \_
\_3,6 | 1.1 | 2.1 | 4.1 | 9.1 |
| (VP \text{hold} \ast \text{conference}) \_
\_3,6 | 3.5 | 4.5 | 6.5 | 11.5 |

Huang and Chiang

Forest Rescoring 12
Cube Pruning

non-monotonic grid due to LM combo costs

(Hold ⋆ meeting)

(PP 3,6

VP 3,6

(PP with ⋆ Sharon)

(PP along ⋆ Sharon)

(PP with ⋆ Shalong)

<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>
Cube Pruning

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP (_3,6) held * meeting) (_3,6)</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) (_3,6)</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) (_3,6)</td>
<td>3.5</td>
<td>4.5 + 0.6</td>
<td>6.5 + 10.5</td>
</tr>
</tbody>
</table>

non-monotonic grid due to LM combo costs

bigram (meeting, with)
Cube Pruning

non-monotonic grid
due to LM combo costs

\[
\begin{align*}
(VP_{3,6}^{\text{held} \star \text{meeting}}) & : \begin{array}{c|c|c|c}
1.0 & 2.5 & 9.0 & 9.5 \\
\hline
1.1 & 2.4 & 9.5 & 9.4 \\
\hline
3.5 & 5.1 & 17.0 & 12.1 \\
\end{array} \\

(VP_{3,6}^{\text{held} \star \text{talk}}) & : \\

(VP_{3,6}^{\text{hold} \star \text{conference}}) & :
\end{align*}
\]
### Cube Pruning

**k-best parsing**  
(Huang and Chiang, 2005)

- a priority queue of candidates
- extract the best candidate

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(VP&lt;sub&gt;3,6&lt;/sub&gt; held * meeting)</td>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
</tr>
<tr>
<td>(VP&lt;sub&gt;3,6&lt;/sub&gt; held * talk)</td>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>(VP&lt;sub&gt;3,6&lt;/sub&gt; hold * conference)</td>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
</tbody>
</table>

Huang and Chiang

Forest Rescoring 15
Cube Pruning

$k$-best parsing
(Huang and Chiang, 2005)

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

<table>
<thead>
<tr>
<th>(VP \text{\text{\textit{held}} \star \text{\text{\textit{meeting}}}})</th>
<th>(VP \text{\text{\textit{held}} \star \text{\text{\textit{talk}}}})</th>
<th>(VP \text{\text{\textit{hold}} \star \text{\text{\textit{conference}}}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
</tr>
<tr>
<td>9.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
</tr>
<tr>
<td>9.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
</tr>
<tr>
<td>12.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Huang and Chiang

Forest Rescoring
**Cube Pruning**

*k*-best parsing  
(Huang and Chiang, 2005)

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

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

Huang and Chiang

Forest Rescoring 17
Cube pruning

- Widely used for phrase-based and syntax-based MT
- May be applied in conjunction with a bottom-up decoder, or as a second “rescoring” pass
- Nodes may also be grouped together (for example, all nodes corresponding to a certain source span)
- Requirement for topological ordering means translation hypergraph may not have cycles
Alignment
Hypergraphs as Grammars

• A hypergraph is isomorphic to a (synchronous) CFG

• LM integration can be understood as the intersection of an LM and an CFG

• Cube pruning approximates this intersection

• Is the algorithm optimal?
Constrained decoding

• Wu (1997) gives an algorithm that is a generalization of CKY to SCFGs

• Requires an approximation of CNF

• Alternative solution:
  • Parse in one language
  • Then parse the other side of the string pair the with “hypergraph” grammar
Input: <dianzi shiang de mao, a cat on the mat>

With thanks and apologies to Zhifei Li.
Input: <dianzi shiang de mao , a cat on the mat>

With thanks and apologies to Zhifei Li.
Input: <<dianzi shiang de mao , a cat on the mat>}

**Isomorphic CFG**

\[
[X34] \rightarrow a \text{ cat}
\]
Input: <"dianzi shiang de mao , a cat on the mat">

Isomorphic CFG

\[ X^{34} \rightarrow \text{a cat} \]
\[ X^{02} \rightarrow \text{the mat} \]
Input: <\textit{dianzi shiang de mao}, a cat on the mat>
Input: <dianzi shiang de mao, a cat on the mat>

Isomorphic CFG

[X34] → a cat
[X02] → the mat
[X04a] → [X34] on [X02]
[X04a] → [X34] of [X02]
[X04b] → [X02] 's [X34]
[X04b] → [X02] [X34]
Isomorphic CFG

Input: <dianzi shiang de mao, a cat on the mat>

[S] → [X04a]
[X04b] → [X02] [X34]
[X04a] → [X34] of [X02]
[X04b] → [X02] 's [X34]
[X02] → the mat
[X34] → a cat
[X04a] → [X34] on [X02]
Input: <dianzi shiang de mao, a cat on the mat>

Isomorphic CFG

[X34] → a cat
[X02] → the mat
[X04a] → [X34] on [X02]
[X04a] → [X34] of [X02]
[X04b] → [X02] 's [X34]
[X04b] → [X02] [X34]
[S] → [X04a]
[S] → [X04b]
Isomorphic CFG

[X34] → a cat
[X02] → the mat
[X04a] → [X34] on [X02]
[X04a] → [X34] of [X02]
[X04b] → [X02] ’s [X34]
[X04b] → [X02] [X34]
[S] → [X04a]
[S] → [X04b]
Input: <dianzi shiang de mao , a cat on the mat>

Isomorphic CFG

[X34] → a cat
[X02] → the mat
[X04a] → [X34] on [X02]
[X04a] → [X34] of [X02]
[X04b] → [X02] 's [X34]
[X04b] → [X02] [X34]
[S] → [X04a]
[S] → [X04b]

[X34]

a cat on the mat
**Isomorphic CFG**

\[
\begin{align*}
[X34] & \rightarrow \text{a cat} \\
[X02] & \rightarrow \text{the mat} \\
[X04a] & \rightarrow [X34] \text{ on } [X02] \\
[X04a] & \rightarrow [X34] \text{ of } [X02] \\
[X04b] & \rightarrow [X02] \text{'s } [X34] \\
[X04b] & \rightarrow [X02] \text{ } [X34] \\
[S] & \rightarrow [X04a] \\
[S] & \rightarrow [X04b] \\

[X34] \rightarrow \text{a cat} \\
[X02] \rightarrow \text{the mat}
\end{align*}
\]
Input: <dianzi shiang de mao , a cat on the mat>

**Isomorphic CFG**

\[
\begin{align*}
[X34] & \rightarrow \text{a cat} \\
[X02] & \rightarrow \text{the mat} \\
[X04a] & \rightarrow [X34] \text{ on } [X02] \\
[X04a] & \rightarrow [X34] \text{ of } [X02] \\
[X04b] & \rightarrow [X02] \text{ 's [X34]} \\
[X04b] & \rightarrow [X02] [X34] \\
[S] & \rightarrow [X04a] \\
[S] & \rightarrow [X04b]
\end{align*}
\]
Isomorphic CFG

\[
\begin{align*}
[X34] & \rightarrow \text{a cat} \\
[X02] & \rightarrow \text{the mat} \\
[X04a] & \rightarrow [X34] \text{ on } [X02] \\
[X04a] & \rightarrow [X34] \text{ of } [X02] \\
[X04b] & \rightarrow [X02] \text{ 's } [X34] \\
[X04b] & \rightarrow [X02] [X34] \\
[S] & \rightarrow [X04a] \\
[S] & \rightarrow [X04b]
\end{align*}
\]
In the first experiment, we compare performance of the two-parse algorithm and the O\((n^2)\) ITG parsing algorithm on an Arabic-English phrasal ITG alignment task. The corpus consisted of \(0.9k\) sentence pairs, each containing \(M\) Arabic tokens and \(M\) English tokens, drawn from the NIST MT evaluation newswire training data. Sentences were filtered to a length of maximally \(N\) tokens on either side. For \(G\), we used a variant of the phrasal ITG described by Zhang et al. The restriction that phrases contain exactly a single alignment point was relaxed; instead, the grammar was restricted to contain all phrases consistent with the word-based alignment up to a maximal phrase size of \(P\). This resulted in a synchronous grammar with \(0.9M\) rules.

Figure 6.0 plots the average run-time of the two algorithms as a function of the Arabic sentence length. Table 6- shows the overall average run-times. Both presentations make clear that the two-parse approach is dramatically more efficient. In total, aligning the \(0.9k\) sentence pairs in the corpus completed in less than \(0.5\) hours with the two-parse algorithm but required more than \(0.5\) week with the baseline algorithm.

Table 6-: Comparison of synchronous parsing algorithms on Arabic-English

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>avg. run-time [sec/word]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITG alignment</td>
<td></td>
</tr>
<tr>
<td>Two-parse algorithm</td>
<td>96.2</td>
</tr>
</tbody>
</table>

In the second experiment, we evaluate an alternative approach to computing a synchronous parse forest that is based on cube pruning (Huang and Chiang). A note on implementation: our ITG aligner was minimal; it only computed the probability of the sentence pair using the inside algorithm. With the two-parse aligner, we stored the complete item chart during both the first and second parses. Therefore, the implementation was biased in favor of the baseline ITG parsing algorithm.

Figure 6.0: Average synchronous parser run-time in seconds per sentence as a function of Arabic sentence length in words.