Translating with Tree Transducers

**Input**

lo haré de muy buen grado .

**Output**

Grammar

\[ ADV \rightarrow \langle de muy buen grado ; gladly \rangle \]
Translating with Tree Transducers

Input | Output
--- | ---

Grammar

\[ S \rightarrow \langle \text{lo haré ADV , ; I will do it ADV .} \rangle \]

\[ ADV \rightarrow \langle \text{de muy buen grado , ; gladly} \rangle \]

---

Translating with Tree Transducers

Input | Output
--- | ---

Grammar

\[ S \rightarrow \langle \text{lo haré ADV , ; I will do it ADV .} \rangle \]

\[ ADV \rightarrow \langle \text{de muy buen grado , ; gladly} \rangle \]

---

Translating with Tree Transducers

Input | Output
--- | ---

Grammar

\[ S \rightarrow \langle \text{lo haré ADV , ; I will do it ADV .} \rangle \]

\[ ADV \rightarrow \langle \text{de muy buen grado , ; gladly} \rangle \]

---

Translating with Tree Transducers

Input | Output
--- | ---

Grammar

\[ S \rightarrow \langle \text{lo haré ADV , ; I will do it ADV .} \rangle \]

\[ ADV \rightarrow \langle \text{de muy buen grado , ; gladly} \rangle \]

---

Translating with Tree Transducers

Input | Output
--- | ---

Grammar

\[ V P \rightarrow \langle \text{lo haré ADV , ; I will do it ADV .} \rangle \]

\[ ADV \rightarrow \langle \text{de muy buen grado , ; gladly} \rangle \]

---

Translating with Tree Transducers

Input | Output
--- | ---

Grammar

\[ S \rightarrow \langle \text{lo haré ADV , ; I will do it ADV .} \rangle \]

\[ ADV \rightarrow \langle \text{de muy buen grado , ; gladly} \rangle \]
Syntactic Decoding

Tree Transducer Grammars

\[ S \rightarrow NN \quad NNP \]
\[ S \rightarrow \text{No se olvide de subir un canto rodado en Colorado} \]

Synchronous Grammar

- \( NN \rightarrow \text{Colorado} \)
  -\( \text{ Colorado} \)
- \( NN \rightarrow \text{canto rodado} \)
  -\( \text{boulder} \)
- \( S \rightarrow \text{No se olvide de subir un NN on NNP} \)
  -\( \text{Don't forget to climb a NN in NNP} \)

Output

\[ S \rightarrow NN \quad NNP \]
\[ \text{Don't forget to climb a boulder in Colorado} \]
CKY-style Bottom-up Parsing

For each span length:

For each span \([i:j]\):

Apply all grammar rules to \([i:j]\)

Binary rule: \(X \rightarrow YZ\)

Split points: \(i < k < j\)

Operations: \(O(j-i)\)

Time scales with: Grammar constant

No se olvide de subir un canto rodado en Colorado \(j\)
CKY-style Bottom-up Parsing

For each span length:  
For each span [ij]:  
Apply all grammar rules to [ij]

\[ S \rightarrow \text{No se vbs de subir un NN en NNP} \]

No se olvide de subir un canto rodado en Colorado.
CKY-style Bottom-up Parsing

For each span length:
- Apply all grammar rules to [u]

\[ S \rightarrow \text{No se } \text{VP } \text{NP } \text{PP} \]

No se olvide de subir un canto rodado en Colorado

Many untransformed lexical rules can be applied in linear time

Problem: Applying adjacent non-terminals is slow

Eliminating Non-terminal Sequences

**Lexical Normal Form (LNF)**

- (a) lexical rules have at most one adjacent non-terminal
- (b) all unlexicalized rules are binary.

Original rule:

\[ S \rightarrow \text{No se } \text{VB } \text{VB } \text{un } \text{NN } \text{PP} \]

Transformed rules:

- \[ S \rightarrow \text{No se } \text{VB}\text{-VB } \text{un } \text{NN}\text{-PP} \]
- \[ \text{VB}\text{-VB } \rightarrow \text{VB } \text{VB} \]
- \[ \text{NN}\text{-PP } \rightarrow \text{NN } \text{PP} \]

Parsing stages:

- Lexical rules are applied by matching
- Unlexicalized rules are applied by iterating over split points
Flexible Syntax

Previous work

- string-to-string: ITG (Wu 1997)
- string-to-tree: Yamada & Knight 2001
- tree-to-string: Huang et al. 2006

Hiero Rules

\[ S \rightarrow (S_{11}X_{12}) \]
\[ S \rightarrow (X_{12}) \]
\[ X \rightarrow (yu X_{12} you X_{12} have X_{12} with X_{12}) \]
\[ X \rightarrow (X_{12} de X_{12} the X_{12} that X_{12}) \]
\[ X \rightarrow (X_{12} zhiyi, one of X_{12}) \]
\[ X \rightarrow (Aozhou, Australia) \]
\[ X \rightarrow (shi, is) \]
\[ X \rightarrow (shoushu guojia, few countries) \]
\[ X \rightarrow (bangjiao, diplomatic relations) \]
\[ X \rightarrow (Beijing, North Korea) \]

From [Chiang et al., 2005]

STSG extraction

1. Phrases
   * respect word alignments
   * are syntactic constituents on both sides
2. Phrase pairs form rules
3. Subtract phrases to form rules

STSG extraction

1. Phrases
   * respect word alignments
   * are syntactic constituents on both sides
2. Phrase pairs form rules
3. Subtract phrases to form rules
STSG extraction

1. Phrases
   - respect word alignments
   - are syntactic constituents on both sides
2. Phrase pairs form rules
3. Subtract phrases to form rules

Why is tree-to-tree hard?

too few rules
too few derivations

Extracting more rules

Allow more derivations

STSG: allow only matching substitutions
Hiero-like allow any substitutions
Let the model learn to choose:
  - matching substitutions
  - mismatching substitutions
  - monotone phrase-based

Allow more derivations

Hiero-like decoding

STSG decoding

fuzzy STSG decoding

fire subst:NP→NP
fire subst:NN→NP
fire subst:match
Exploiting GPUs

Lots to Parse

=2.6 billion words
Lots to Parse

=6 months (CPU)

Lots to Parse

=3.6 days (GPU)

CPU Parsing

>98% sparsity

[Slide credit: Slav Petrov]

CPU Parsing

Skip Spans

Skip Rules

CPU Parsing

CPU

CPU Parsing

CPU
The Future of Hardware

16384 Threads
The Future of Hardware

Warps

Warp

Warp Divergence

Warp

Warp Divergence
Warps

Warp Divergence

Warps

Coalescence

Designing GPU Algorithms

Dense, Uniform Computation

Designing GPU Algorithms

CPU
Irregular, Sparse

GPU
Regular, Dense

Designing GPU Algorithms

CPU
Irregular, Sparse

GPU
Regular, Dense

CKY Algorithm

[Canny, Hall, and Klein, 2013]
for each sentence:
  for each span (begin, end):
    for each split:
      for each rule ($P \rightarrow LR$):
        $score[begin, end, P] + \text{ruleScore}[P \rightarrow LR] \times score[begin, split, L] \times score[split, end, R]$

for each sentence:
  for each span (begin, end):
    for each split:
      applyGrammar(begin, split, end)

for each parse item in sentence:
  applyGrammar(item)

for each parse item in sentence:
  applyGrammar(item)

GPU Parsing Pipeline

<table>
<thead>
<tr>
<th>CPU Queue</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i, k, j)</td>
<td></td>
</tr>
<tr>
<td>(0, 1, 3)</td>
<td></td>
</tr>
<tr>
<td>(0, 2, 3)</td>
<td></td>
</tr>
<tr>
<td>(1, 2, 4)</td>
<td></td>
</tr>
<tr>
<td>(1, 3, 4)</td>
<td></td>
</tr>
</tbody>
</table>

Parsing Speed

CPU | GPU
---|---
190 s/sec | 10 s/sec

Sentences per second

Exploiting Sparsity

Grammar

S
NP VP

CPU Queuing

GPU Application

Exploiting Sparsity

Grammar

S
NP VP

GPU Application

Exploiting Sparsity

Warp (1, 2, 4)
(0, 1, 3)
(0, 2, 3)
(1, 2, 4)
(1, 3, 4)
(2, 3, 5)
(2, 4, 5)
(3, 4, 6)
...

Exploiting Sparsity

Warp Divergence

Exploiting Sparsity

Grammar

S
NP VP

GPU Application
Exploiting Sparsity

CPU

GPU

Parsing Speed

NP Queue

(i, k, j)

(0, 1, 3)

(0, 2, 3)

...