Incremental Parsing with TAG

Miriam Käshammer

Course: Incremental Processing
Department of Computational Linguistics, Universität des Saarlandes

May 28, 2011
Tree Adjoining Grammar (TAG)

- tree-rewriting formalism
- a set of (elementary) trees with two operations

Auxiliary trees:

- \( VP \)
  - \( ADV \)
    - \( often \)
  - \( VP^* \)

Initial trees:

- \( S \)
  - \( NP \downarrow \)
    - \( Peter \)
  - \( VP \)
    - \( V \)
      - \( laughs \)
Tree Adjoining Grammar

Operations

Substitution  Replacing a leaf with an initial tree
Adjunction  Replacing an internal node with an auxiliary tree

```
NP  S
Peter  S
   NP  V
   NP  laughs
  VP  V
 ADV  VP*  
   VP  
  often
```

⇒

```
NP  S
  VP  
 ADV  VP
   VP
      V
   often
       laughs
```
Tree Adjoining Grammar

Some linguistic principles:

- Lexicalization: Each elementary tree has at least one non-empty lexical item, its anchor. (LTAG)
- Predicate argument co-occurrence: Elementary trees of predicates contain slots for the arguments they subcategorize for.

```
S
   |   NP   VP
   |   NP
Peter
   | VP
      | VP*
      | V
      | laughs
      | often
```
Why is TAG interesting?

- **Mildly context-sensitive formalism**
  1. It generates (at least) all context-free languages.
  2. It captures a limited amount of cross-serial dependencies, e.g. the copy language \( \{ww\,|\,w \in \{a, b\}^*\} \).
  3. It can be parsed in polynomial time. \( O(n^6) \)
  4. It has constant growth property.

  \[ \Rightarrow \text{appropriate to describe natural languages} \]

- **Important characteristics**
  1. **Extended domain of locality** - elementary trees can be arbitrarily large.
  2. **Factoring of recursion** - adjunction operations allows to put recursive structures into separate elementary trees.
Why incremental parsing?

- Psycholinguistic evidence
  - Humans build up semantic representation before reaching the end of the sentence.
  - Interpretation is based on fast, left-to-write construction of syntactic relations.

- Boost in speed
[Sturt and Lombardo, 2005] argue that models of human parsing incorporate an operation similar to adjunction in TAG.

Traditional LTAG does not allow full connectedness.

Peter often ...

```
NP
  | VP
  |   | ADV
  |   | VP*
  |   |   | often

Peter
```
Where is incrementality encoded?

Components of a parser:

- a (competence) grammar
- a parsing strategy
- a memory organizing strategy
- an oracle
Where is incrementality encoded?

Components of a parser:

- a (competence) grammar
- a parsing strategy
- a memory organizing strategy
- an oracle
Where is incrementality encoded?

Components of a parser:

⇒ a (competence) grammar
  - a parsing strategy
  - a memory organizing strategy
  - an oracle
Approaches

LTAG-spinal - “Incremental” parser
Incremental LTAG Parsing
L. Shen and A. K. Joshi (2005)

DVTAG - Strictly incremental parser
Dynamic TAG and lexical dependencies
A. Mazzei, V. Lombardo and P. Sturt (2007)
Outline

1. Introduction
2. “Incremental” TAG
3. Strictly incremental TAG
4. Conclusion
Outline

1. Introduction
   - TAG
   - Motivation
   - Overview

2. “Incremental” TAG
   - Grammar Formalism
   - Parsing Algorithm
   - Training
   - Evaluation

3. Strictly incremental TAG
   - Dynamics
   - Formalism
   - Wide-coverage Grammar

4. Conclusion
LTAG-spinal

Variant of LTAG

- initial tree only contains its *spine*
- auxiliary tree only contains its *spine* and a foot node directly linked to the spine

**Definition: Spine**

The *spine* of an elementary tree is the path from the root node to the anchor of the tree.
LTAG-spinal formalism

Operations

Adjunction (A) Same as in LTAG.

Attachment (T) Attachment of an initial tree $\alpha$ to a node $n$ of another tree $\alpha'$: add the root of $\alpha$ to $n$ as a new child.

Conjunction (C) Special operation to build coordination structures.
LTAG-spinal formalism

Relation to LTAG

- LTAG-spinal is more powerful than CFG. [Shen and Joshi, 2005a]
- LTAG-spinal with attachment constraints is weakly equivalent to traditional LTAG. [Shen et al., 2007]
Relation to LTAG

- LTAG-spinal is more powerful than CFG. [Shen and Joshi, 2005a]
- LTAG-spinal with attachment constraints is weakly equivalent to traditional LTAG. [Shen et al., 2007]
- LTAG-spinal trees generalize over predicates with different subcategorization frames.
The Parsing Algorithm

- Four types of **parser operations**:
  - Attachment, adjunction, conjunction
  - Generation: generate a possible spine for a given word according to the context and the lexicon (Supertagging)

- Variant of the shift-reduce algorithm, using a **stack of disconnected treelets** to represent the left context
  - **Shift**: Read a word, generate a list of possible elementary trees for this word. For each elementary tree, push it into the stack.
  - **Reduce**: Pop the top two treelets from the stack, combine them by attachment, adjunction or conjunction and push the combined tree into the stack.

- Beam-search to prune the search space
Example

DT    NN    WDT    VBZ    JJ    CC    JJ    TO    PRP
a    parser    which    seems    new    and    interesting    to    me

G: generate    T: attach    A: adjoin    C: conjoin

Graph taken from http://libinshen.net/Documents/ijc04_slides.ps
Example

A parser which seems new and interesting to me

G: generate   T: attach   A: adjoin   C: conjoin
Example

\[
\begin{array}{ccccccccccc}
\text{G: generate} & \text{T: attach} & \text{A: adjoin} & \text{C: conjoin} \\
\end{array}
\]
Example

a parser which seems new and interesting to me

G: generate T: attach A: adjoin C: conjoin
Example

```
G: generate    T: attach    A: adjoin    C: conjoin
```

```
a parser which seems new and interesting to me
```
Example

```
a parser which seems new and interesting to me
```

G: generate  T: attach  A: adjoin  C: conjoin
Example

a parser which seems new and interesting to me

G: generate T: attach A: adjoin C: conjoin
Example

a parser which seems new and interesting to me

G: generate  T: attach  A: adjoin  C: conjoin
Example

a parser which seems new and interesting to me

G: generate  T: attach  A: adjoin  C: conjoin
Example

a parser which seems new and interesting to me

G: generate    T: attach    A: adjoin    C: conjoin
Example

a parser which seems new and interesting to me

G: generate      T: attach      A: adjoin      C: conjoin

Graph taken from http://libinshen.net/Documents/ijc04_slides.ps
Flex Model vs. Eager Model

Pseudo-ambiguity in the shift-reduce derivation:
A adjoins to B, B adjoins to C

- \((A \rightarrow B) \rightarrow C\)
- \((A \rightarrow (B \rightarrow C))\)
Flex Model vs. Eager Model

Pseudo-ambiguity in the shift-reduce derivation:
A adjoins to B, B adjoins to C

• \((A \rightarrow B) \rightarrow C\)
• \((A \rightarrow (B \rightarrow C))\)

Flex Model
• Both derivations are allowed.

Eager Model
• Only \((A \rightarrow B) \rightarrow C\) is allowed.
Features and Learning

**Features** extracted from gold-standard parses have the following format:

\[(\text{operation}, \text{main_spine}, \text{child_spine}, \text{spine_node}, \text{context})\]

- **spine_node**: node on the **main_spine** onto which the **child_spine** is attached/adjointed/conjoined
- **context**: dependent on the type of operation; includes amongst others the (0,2) window in the sentence

**Weights** for the features are learned using a perceptron-like algorithm as proposed in [Collins, 2002].
Evaluation

- LTAG-spinal treebank (see [Shen and Joshi, 2005b])
- Training, development and test data from WSJ
- Syntactic dependency for evaluation against PTB

<table>
<thead>
<tr>
<th>model</th>
<th>beam</th>
<th>sen/sec</th>
<th>f-score %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flex</td>
<td>10</td>
<td>0.37</td>
<td>89.3</td>
</tr>
<tr>
<td>Eager</td>
<td>10</td>
<td>0.79</td>
<td>88.7</td>
</tr>
</tbody>
</table>

With an extension (Combined Parses) and beam=100: 94.2%
What we have seen so far

Parser for LTAG-spinal

- Incremental?
  - Input is processed incrementally, but only partially
  - Structure is not fully connected (usage of a stack)
  - Look-ahead of 2 words

- Implemented: efficient statistical parsing
- Generative power of grammar is stronger than CFG
- Motivation?
Outline

1. Introduction
   - TAG
   - Motivation
   - Overview

2. “Incremental” TAG
   - Grammar Formalism
   - Parsing Algorithm
   - Training
   - Evaluation

3. Strictly incremental TAG
   - Dynamics
   - Formalism
   - Wide-coverage Grammar

4. Conclusion
Dynamics

Dynamic Grammar

- The syntactic analysis is viewed as a dynamic process
  ⇒ A sequence of transitions between adjacent syntactic states \( S_{i-1} \) and \( S_i \).

- A syntactic state contains all the syntactic information about the fragment already processed.
Dynamics and Incrementality

**Incremental processing:**
- Each input word $w_i$ defines a transition from $S_{i-1}$ (left-context) to $S_i$.
- States as partial syntactic structures

**Strong connectivity:**
- Impose that transitions do not produce disconnected trees

Parsing strategy is part of the grammar
⇒ Incrementality-in-competence
Dynamics in TAG - Intuition (I)
At step $i$, the elementary tree anchored in $w_i$ is combined with the partial structure spanning the words $w_1...w_{i-1}$.

$\Rightarrow$ Updated left-context spanning the words $w_1...w_i$
Dynamic Version of TAG
Elementary trees similar to LTAG trees, BUT
Dynamic Version of TAG
Elementary trees similar to LTAG trees, BUT

- Lexical items (but not the left-anchor) can be **underspecified**. The preterminal category is paired with a finite list of values.
  $\Rightarrow$ **predicted nodes** to fulfill full connectivity
Dynamic Version of TAG

Elementary trees similar to LTAG trees, BUT

- Lexical items (but not the left-anchor) can be underspecified. The preterminal category is paired with a finite list of values. ⇒ predicted nodes to fulfill full connectivity
- Distinction between left/right auxiliary trees
Head feature

- Feature that indicates the lexical head of each node in the elementary tree
- Needed to compute a *dependency tree*
Some DVTAG terminology

Dotted tree
A pair $\langle \gamma, i \rangle$ where $\gamma$ is a tree and $i$ is an integer such that $i \in 0 \ldots |YIELD(\gamma)|$.
Moreover, all the symbols on the yield that are on the left of the dot are terminal symbols.

Fringe of $\langle \gamma, i \rangle$
Set of nodes that are accessible for operations
DVTAG Operations

Combination of a left-context $\langle \Lambda, i \rangle$ with an (unanalysed) elementary tree $\langle \gamma, 0 \rangle$

- 2 substitution operations
- 4 adjoining operations
- a shift operation

“Normal” operations
- Substitution $Sub^\rightarrow$
- Adjoining from the left $\nabla^L$
- Adjoining from the right $\nabla^R$
- Shift $Shi$

Inverse operations
- Inv. substitution $Sub^\leftarrow$
- Inv. adj. from the left $\nabla^L$
- Inv. adj. from the right $\nabla^R$
DVTAG Operations

**Shift**

$Shi(⟨\gamma, i⟩)$ takes as input a single dotted tree $⟨\gamma, i⟩$ and returns the dotted tree $⟨\gamma, i + 1⟩$. It can be applied only if a terminal symbol belongs to the fringe of $⟨\gamma, i⟩$.
DVTAG Operations

Substitution

$Sub^\rightarrow (\langle \alpha, 0 \rangle, \langle \gamma, i \rangle)$:
If there is a substitution node $N$ in the fringe of $\langle \gamma, i \rangle$ such that $\text{label}(N) = \text{label}(\text{root}(\alpha))$, the operation returns a new dotted tree $\langle \delta, i + 1 \rangle$ such that $\delta$ is obtained by grafting $\alpha$ into $N$. 
DVTAG Operations

Substitution and inverse Substitution

\[ \text{Sub}^{-1}(\langle \alpha, 0 \rangle, \langle \gamma, 1 \rangle) \]

\[ \text{Sub}^{-1}(\langle \zeta, 0 \rangle, \langle \gamma, 1 \rangle) \]
DVTAG Operations

Adjoining from the left

$$\nabla_L^\rightarrow (\langle \beta, 0 \rangle, \langle \gamma, i \rangle, \text{add})$$ where $\beta$ is a left auxiliary tree:

If there is a non-terminal node $N$ at position $\text{add}$ in the fringe of $\langle \gamma, i \rangle$ such that $\text{label}(N) = \text{label}(\text{root}(\beta))$, the operation returns a new dotted tree $\langle \delta, i + 1 \rangle$ such that $\delta$ is obtained by grafting $\beta$ into $N$. 

Diagram:

- Initial tree $\langle \beta, 0 \rangle$ with label $\beta$.
- Adjoining a new node $\langle \gamma, 1 \rangle$ labeled $\gamma$ at position $i$.
- The operation $\nabla_L^\rightarrow$ grafts $\beta$ into the tree at node $N$.
- Resulting tree $\langle \delta, 2 \rangle$ with label $\delta$. 

Diagram shows the structure before and after the operation.
Example derivation

\[
\begin{align*}
\langle \cdot, 0 \rangle & \xrightarrow{\text{Bill}} Shi(\langle \alpha_{\text{Bill}}, 0 \rangle) \\
& \xrightarrow{\text{often}} \nabla_L(\langle \beta_{\text{often}}, 0 \rangle, \langle \Lambda_1, 1 \rangle, 2)
\end{align*}
\]
Example derivation

\[
\text{pleases} \xrightarrow{\text{Shi}(\langle \Lambda_2, 2 \rangle)} \quad \text{S}(\text{pleases})
\]

\[
\text{Bill} \quad \text{V}(\text{pleases}) \quad \text{NP}(\text{Bill})
\]

\[
\text{often} \quad \text{V}(\text{pleases}) \quad \text{NP}(\text{Bill})
\]

\[
\text{often} \quad \text{V}(\text{pleases}) \quad \text{NP}(\text{Bill})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{Sub}^{-1}(\langle \alpha_{\text{Sue}}, 0 \rangle, \langle \Lambda_3, 3 \rangle)
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]

\[
\text{S}(\text{pleases}) \quad \text{VP}(\text{pleases})
\]
A wide-coverage DVTAG

Ways to build a grammar:

1. Manually write it (XTAG, FTAG)
2. Automatically extract it from treebanks

Anticipated problem: Size of grammar because of predicted nodes
Grammar seizes in comparison

<table>
<thead>
<tr>
<th></th>
<th># of tree templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAG</td>
<td>1,200</td>
</tr>
<tr>
<td>DVTAG from XTAG</td>
<td>6,000,000</td>
</tr>
</tbody>
</table>
Grammar seizes in comparison

<table>
<thead>
<tr>
<th></th>
<th># of tree templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAG</td>
<td>1,200</td>
</tr>
<tr>
<td>DVTAG from XTAG</td>
<td>6,000,000</td>
</tr>
<tr>
<td>DVTAG from PTB</td>
<td>12,000</td>
</tr>
</tbody>
</table>
# Grammar seizes in comparison

<table>
<thead>
<tr>
<th></th>
<th># of tree templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAG</td>
<td>1,200</td>
</tr>
<tr>
<td>DVTAG from XTAG</td>
<td>6,000,000</td>
</tr>
<tr>
<td>DVTAG from PTB</td>
<td>12,000</td>
</tr>
<tr>
<td>LTAG-spinal</td>
<td>1,200</td>
</tr>
</tbody>
</table>

Numbers are taken from [Mazzei, 2005] and [Shen et al., 2007], and rounded.
Outline

1. Introduction
   - TAG
   - Motivation
   - Overview

2. “Incremental” TAG
   - Grammar Formalism
   - Parsing Algorithm
   - Training
   - Evaluation

3. Strictly incremental TAG
   - Dynamics
   - Formalism
   - Wide-coverage Grammar

4. Conclusion
Conclusion

LTAG-spinal
- Parsing strategy specifies the “incremental” nature.
- In fact, not very incremental (stack, look-ahead)
- Efficient, implemented parser available

DVTAG
- The (competence) grammar determines the parsing strategy
- Natively fulfills a strict version of incrementality
- Resembles left-corner strategy (⇒ center embeddings)
- Grammars grow very large in size
References I


References II


Discussion

- Why would we want incrementality in competence?
- For NLP applications the “incremental” parser might be enough.
- Can psycholinguistic findings/memory profiles be explained with these grammars?