Incremental Dependency Parsing

Michael Fell

9 June 2011
Overview

- Incremental Dependency Parsing
  - two algorithms
  - evaluation

- General criticism on present approaches
  - possible improvements

- Summary
The man loves cake.

(Dependency Graph)
Dependency Graph

Labeled, directed graph \((W, A)\)

- \(W\): words in the sentence
- \(A\): dependency relation between words
Dependency Graph

Labeled, directed graph \((W, A)\)
- \(W\): words in the sentence
- \(A\): dependency relation between words

Well-formedness criteria:
- connected
- acyclic
- unique label
- single head
- projective
Incremental Dependency Parsing

Dependency parsing
- is robust and performs well
- omits phrasal nodes

What about doing it incrementally?
Incremental Dependency Parsing

Dependency parsing
- is robust and performs well
- omits phrasal nodes

What about doing it incrementally?

One possibility:
Left-to-right bottom-up dependency parsing
Bottom-up Dependency Parsing

stack  input  dep. graph
Bottom-up Dependency Parsing

Shift: (S)

Stack

Input

Dep. Graph

abc
bc
G
G
Bottom-up Dependency Parsing

Shift: \[(S)\]

Left-Reduce (LR):
Bottom-up Dependency Parsing

Shift: (S)

Right-Reduce (RR):

Left-Reduce (LR):
Bottom-up Dependency Parsing

Example derivation of \( a \ b \ c \)

\[ \text{abc} \quad \emptyset \]
Bottom-up Dependency Parsing

Example derivation of $abc$

(1)

\[
\begin{array}{c}
\text{abc} \\
\text{bc} \\
\text{S} \\
\hline
\text{a}
\end{array}
\]

$\emptyset$

$\emptyset$
Bottom-up Dependency Parsing

Example derivation of \text{abc} \rightarrow \text{bc} \rightarrow \text{a}

(1)
\text{S} \rightarrow \text{abc} \quad \emptyset

(2)
\text{S} \rightarrow \text{bc} \quad \emptyset
\text{S} \rightarrow \text{c} \quad \emptyset
Bottom-up Dependency Parsing

Example derivation of \(abc\)

(1) \(\begin{array}{c}
\text{S} \\
a \\
\end{array}\) \(\begin{array}{c}
\text{abc} \\
\emptyset
\end{array}\)

(2) \(\begin{array}{c}
\text{S} \\
b \\
\end{array}\) \(\begin{array}{c}
\text{bc} \\
\emptyset
\end{array}\)

(3) \(\begin{array}{c}
\text{LR} \\
b \\
\end{array}\) \(\begin{array}{c}
c \\
\emptyset
\end{array}\)

\(\begin{array}{c}
a \\
\end{array}\) \(\begin{array}{c}
b \\
\end{array}\)
Bottom-up Dependency Parsing

Example derivation of $abc$

(1) $\emptyset$

(2) $\emptyset$

(3) $\emptyset$

(4) $\emptyset$
Bottom-up dependency parsing

Example derivation of $abc$

1. $\emptyset$
2. $bc$
3. $bc$
4. $a \ b$
5. $a \ b \ c$
Bottom-up dependency parsing

Dependency graphs with 3 nodes:

(1) \[ a \rightarrow b \rightarrow c \]  
(2) \[ a \rightarrow b \rightarrow c \]  
(3) \[ a \rightarrow b \rightarrow c \]  
(4) \[ a \rightarrow b \rightarrow c \]  
(5) \[ a \leftarrow b \leftarrow c \]  
(6) \[ a \leftarrow b \leftarrow c \]  
(7) \[ a \leftarrow b \leftarrow c \]

We have derived (4). (2), (3) and (5) can also be derived.
Bottom-up dependency parsing

Dependency graphs with 3 nodes:

We have derived (4). (2), (3) and (5) can also be derived.

(1) and (6), (7) can’t be derived
Bottom-up Dependency Parsing

Dependency graphs with 3 nodes:

We have derived (4). (2), (3) and (5) can also be derived.

(1) and (6), (7) can’t be derived

(1): b is combined via Right-Reduction $\rightarrow$ b has a head $\rightarrow$ b erased from stack
Bottom-up Dependency Parsing

Dependency graphs with 3 nodes:

We have derived (4). (2), (3) and (5) can also be derived.

(1) and (6), (7) can’t be derived

(1): b is combined with a via Right-Reduction $\rightarrow$ b has a head $\rightarrow$ b erased from stack
(6), (7): no connecting arc between a and b $\rightarrow$ To connect them, we needed to put c onto the stack, too. (hence lose incrementality)
Bottom-up Dependency Parsing

Is there a way to parse (1) and (6), (7) incrementally?

→ (6), (7): no!
Is there a way to parse (1) and (6), (7) incrementally?

→ (6), (7): no!

→ (1): yes, read input from right to left

→ incrementality?
Is there a way to parse (1) and (6), (7) incrementally?

→ (6), (7): no!
→ (1): yes, read input from right to left
   → incrementality?

(1) can be processed incrementally
Incremental Dependency Parsing

Bottom-up and Top-down in Dependency Parsing

BU: D H x

Dependent D is attached to its head H before H is attached to its head
Incremental Dependency Parsing

Bottom-up and Top-down in Dependency Parsing

**BU:**

Dependent D is attached to its head H before H is attached to its head

**TD:**

Head H is attached to a dependent D before D is attached to its dependent(s)
Incremental Dependency Parsing

Bottom-up and Top-down in Dependency Parsing

**BU:**

Dependent $D$ is attached to its head $H$ before $H$ is attached to its head

**TD:**

Head $H$ is attached to a dependent $D$ before $D$ is attached to its dependent(s)

Insight:

→ We can process left-dependents incrementally via BU
→ process right-dependents incrementally via TD parsing
Incremental Dependency Parsing

Bottom-up and Top-down in Dependency Parsing

BU:  

Dependent D is attached to its head H before H is attached to its head

TD:  

Head H is attached to a dependent D before D is attached to its dependent(s)

Insight:

→ We can process left-dependents incrementally via BU
→ process right-dependents incrementally via TD parsing

→ Arc-Eager Dependency Parsing
Arc-Eager Dependency Parsing

Shift: (S)

stack: S

input: abc

dep. graph: G

stack: S

input: bc

dep. graph: G
Arc-Eager Dependency Parsing

Shift: (S)

Left-Arc (LA):
Arc-Eager Dependency Parsing

**Shift:** (S)

**Right-Arc (RA):**

**Left-Arc (LA):**
Arc-Eager Dependency Parsing

**Shift:**
(S)

- **Stack:** $S$
- **Input:** abc
- **Dep. Graph:** $G$

**Right-Arc (RA):**

- **Stack:** $a$
- **Input:** $bc$
- **Dep. Graph:** $G$
- **Result:** $G \cup a \ b$

**Left-Arc (LA):**

- **Stack:** $a$
- **Input:** $bc$
- **Dep. Graph:** $G$

**Reduce (R):**

- **Stack:** $S$
- **Input:** $bc$
- **Dep. Graph:** $G \cup a \ b$
- **Result:** $G$
Bottom-up vs. Arc-Eager

Shift: $S_{BU}$

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Dep. Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>abc</td>
<td>G</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>bc</td>
<td>G</td>
</tr>
</tbody>
</table>

Shift: $S_{AE}$

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Dep. Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>abc</td>
<td>G</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>bc</td>
<td>G</td>
</tr>
</tbody>
</table>
Bottom-up vs. Arc-Eager

**Shift:**

(Bottom-up)

\[ (S_{BU}) \]

<table>
<thead>
<tr>
<th>Stack:</th>
<th>Input:</th>
<th>Dep. Graph:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>abc</td>
<td>G</td>
</tr>
<tr>
<td>( a )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( b )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Left-Reduce (LR):**

<table>
<thead>
<tr>
<th>Stack:</th>
<th>Input:</th>
<th>Dep. Graph:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
<td>a</td>
<td>G ( \cup ) a b</td>
</tr>
<tr>
<td>( a )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( b )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Shift:**

(Arc-Eager)

\[ (S_{AE}) \]

<table>
<thead>
<tr>
<th>Stack:</th>
<th>Input:</th>
<th>Dep. Graph:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>abc</td>
<td>G</td>
</tr>
<tr>
<td>( a )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( b )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Left-Arc (LA):**

<table>
<thead>
<tr>
<th>Stack:</th>
<th>Input:</th>
<th>Dep. Graph:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
<td>c</td>
<td>G ( \cup ) a b</td>
</tr>
<tr>
<td>( a )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( b )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bottom-up vs. Arc-Eager

**Shift:** (S_{BU})

- **Stack:**
  - $S$
  - $a$
  - $b$
- **Input:**
  - $abc$
  - $bc$
- **Dep. Graph:**
  - $G$

**Left-Reduce (LR):**

- **Stack:**
  - $a$
  - $b$
  - $c$
- **Input:**
  - $bc$
  - $c$
- **Dep. Graph:**
  - $G$
  - $G \cup a \ b$

**Shift:** (S_{AE})

- **Stack:**
  - $S$
  - $a$
  - $b$
- **Input:**
  - $abc$
  - $bc$
- **Dep. Graph:**
  - $G$

**Left-Arc (LA):**

- **Stack:**
  - $a$
  - $S$
- **Input:**
  - $bc$
- **Dep. Graph:**
  - $G$
  - $G \cup a \ b$
**Bottom-up vs. Arc-Eager**

---

**Shift: (S_{BU})**

- Stack: \( S \)
- Input: \( abc \)
- Dep. Graph: \( G \)

---

**Left-Reduce (LR):**

- Stack: \( S \)
- Input: \( ab \)
- Dep. Graph: \( G \)

---

**Shift: (S_{AE})**

- Stack: \( S \)
- Input: \( ab \)
- Dep. Graph: \( G \)

---

**Left-Arc (LA):**

- Stack: \( S \)
- Input: \( ab \)
- Dep. Graph: \( G \)

---

Bottom-up versus Arc-Eager highlights the differences in parsing strategies between stack-based (Bottom-up) and arc-based (Arc-Eager) approaches in formal language theory.
Bottom-up vs. Arc-Eager

Right-Reduce (RR):

```
S
a
b
```

```
\text{RR}
```

```
c
G \cup a \ b
```

Right-Arc (RA):

```
S
a
b
```

```
\text{RA}
```

```
b c
G \cup a \ b
```

```
c
G \cup a \ b
```

37
Bottom-up vs. Arc-Eager

Right-Reduce (RR):

\[
\begin{array}{c}
S \\
\Downarrow S \\
\Downarrow RR \\
S
\end{array}
\]

bc 

\[
G
\]

c 

\[
G \cup a \ b
\]

Right-Arc (RA):

\[
\begin{array}{c}
S \\
\Downarrow RA \\
S
\end{array}
\]

bc 

\[
G
\]

c 

\[
G \cup a \ b
\]
Bottom-up vs. Arc-Eager

Right-Reduce (RR):

\[
\begin{align*}
  &a \quad S \\
  &b \quad a \quad S \\
  &a \quad S \\
  &c \\
  &G \\
  &G \cup a \ b
\end{align*}
\]

Right-Arc (RA):

\[
\begin{align*}
  &a \quad S \\
  &b \quad a \quad S \\
  &a \quad S \\
  &c \\
  &G \\
  &G \cup a \ b
\end{align*}
\]
Arc-Eager Dependency Parsing can fully simulate Bottom-up Dependency Parsing!

→ Arc-Eager Dependency Parsing can fully simulate Bottom-up Dependency Parsing!
Bottom-up vs. Arc-Eager

Right-Reduce (RR):

\[
\begin{align*}
S & \quad \text{bc} \\
S & \quad \text{c} \\
S & \quad \text{c} \\
S & \quad \text{G} \\
\end{align*}
\]

Right-Arc (RA):

\[
\begin{align*}
S & \quad \text{bc} \\
S & \quad \text{c} \\
S & \quad \text{c} \\
S & \quad \text{G} \\
\end{align*}
\]

→ Arc-Eager Dependency Parsing can fully simulate Bottom-up Dependency Parsing

→ We can also derive new graphs with AE! (see next slide)
Arc-Eager Dependency Parsing

(1) $a \rightarrow b \rightarrow c$

(1) is not derivable with BU parsing, but it is with AE:

```
  abc  ∅
```

Arc-Eager Dependency Parsing

(1) \( a \rightarrow b \rightarrow c \)

(1) is not derivable with BU parsing, but it is with AE:

\[
\begin{array}{c}
\text{abc} \\
\text{bc} \\
\text{a} \\
\end{array}
\]
Arc-Eager Dependency Parsing

(1) \( a \rightarrow b \rightarrow c \)

(1) is not derivable with BU parsing, but it is with AE:

\[
\begin{array}{c}
\text{abc} \\
\text{bc} \\
\text{c} \\
\end{array}
\quad
\begin{array}{c}
\varnothing \\
\varnothing \\
\text{a b} \\
\end{array}
\]
Arc-Eager Dependency Parsing

(1) is not derivable with BU parsing, but it is with AE:
Arc-Eager Parsing: Evaluation

- small Swedish treebank (5685 sentences)
- evaluating incrementality: number of connected components on stack during parse ($\leq 1$ means strictly incremental)
Arc-Eager Parsing: Evaluation

- small Swedish treebank (5685 sentences)
- evaluating incrementality: number of connected components on stack during parse ($\leq 1$ means strictly incremental)

<table>
<thead>
<tr>
<th>Connected components</th>
<th>Parser configurations Number</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1251</td>
<td>7.6</td>
</tr>
<tr>
<td>1</td>
<td>10148</td>
<td>61.3</td>
</tr>
<tr>
<td>2</td>
<td>2739</td>
<td>16.6</td>
</tr>
<tr>
<td>3</td>
<td>1471</td>
<td>8.9</td>
</tr>
<tr>
<td>4</td>
<td>587</td>
<td>3.5</td>
</tr>
<tr>
<td>5</td>
<td>222</td>
<td>1.3</td>
</tr>
<tr>
<td>6</td>
<td>98</td>
<td>0.6</td>
</tr>
<tr>
<td>7</td>
<td>26</td>
<td>0.2</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>0.0</td>
</tr>
<tr>
<td>$\leq 1$</td>
<td>11399</td>
<td>68.9</td>
</tr>
<tr>
<td>$\leq 3$</td>
<td>15609</td>
<td>94.3</td>
</tr>
<tr>
<td>$\leq 8$</td>
<td>16545</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Arc-Eager Parsing: Evaluation

- small Swedish treebank (5685 sentences)
- evaluating incrementality: number of connected components on stack during parse ($\leq 1$ means strictly incremental)

```
<table>
<thead>
<tr>
<th>Connected components</th>
<th>Parser configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number</td>
</tr>
<tr>
<td>0</td>
<td>1251</td>
</tr>
<tr>
<td>1</td>
<td>10148</td>
</tr>
<tr>
<td>2</td>
<td>2739</td>
</tr>
<tr>
<td>3</td>
<td>1471</td>
</tr>
<tr>
<td>4</td>
<td>587</td>
</tr>
<tr>
<td>5</td>
<td>222</td>
</tr>
<tr>
<td>6</td>
<td>98</td>
</tr>
<tr>
<td>7</td>
<td>26</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

$\leq 1$ | 11399 | 68.9 |
$\leq 3$ | 15609 | 94.3 |
$\leq 8$ | 16545 | 100.0 |
```
Intermediate Summary

- Dependency parsing works well in practice
- Incremental dependency parsing possible in many cases
- Improving the parsing technique is essential
  - Arc-Eager performs better than Bottom-up dep. parsing
  - Well-formed parsing results show high incrementality
Intermediate Summary

- Dependency parsing works well in practice
- Incremental dependency parsing possible in many cases
- Improving the parsing technique is essential
  - Arc-Eager performs better than Bottom-up dep. parsing
  - Well-formed parsing results show high incrementality

- … but, what about those structures (6) and (7) we couldn’t parse incrementally?
Robust Incrementality

Das Buch, mit dem der Lehrer
The book with which the teacher
Robust Incrementality

Drawbacks of storing components on a stack
- psycholinguistic plausibility: why not integrate directly?
Robust Incrementality

- psycholinguistic plausibility: why not integrate directly?
- practicality: delay of output as stored components are not part of it
Robust Incrementality

- Argument Dependency Model
  - dependencies between a verb’s arguments
  - proto roles (proto-agent, proto-patient)
    - e.g.: noun(animate & nominative)
      \[\rightarrow\] noun(proto-agent)
      \[\rightarrow\] dependency rel. SUBJ governs the noun (phrase)
      \[\rightarrow\] unless contradictory constraints override this

Das Buch, mit dem der Lehrer
The book with which the teacher
Robust Incrementality

NONSPEC node

- connect structures to NONSPEC node while verb has not been found
- NONSPEC can change into any other node and even divide into several nodes
- May even be in the resulting graph
Robust Incrementality: Evaluation

- corpus with
  - uniform sentence pattern
  - verb-final subclauses

→ 97.3% accurate dependency graphs, but...
Summary

- Incremental Dependency Parsing is possible and efficient
- Verb-end structures pose problems to strict incrementality
- Pseudo-strict incrementality with abstract NONSPEC node suggested
- Integrates dep. relations on-the-fly
- still seems a lot like a renamed stack to me (which can be output) → too vague
Thank you!
References

• Joakim Nivre (2004). Incrementality in Deterministic Dependency Parsing
• Wolfgang Menzel (2009). Towards radically incremental parsing of natural language