A Dynamic Oracle for Arc-Eager Dependency Parsing

Yoav Goldberg       Joakim Nivre
Bar Ilan University  Uppsala University

COLING 2012
This talk is about parsing
Syntactic Parsing transforms a sentence into a syntactic tree.
Syntactic Parsing transforms a sentence into a syntactic tree.

A useful signal for various downstream tasks:

- Entity extraction and resolution
- Knowledge Acquisition
- Question Answering (query understanding)
- Voice-commands
- Translation
Syntactic Parsing

- A useful signal for various downstream tasks:
  - Entity extraction and resolution
  - Knowledge Acquisition
  - Question Answering (query understanding)
  - Voice-commands
  - Translation
Approaches to parsing

Global Optimization
▶ Define a scoring function over <sentence,tree> pairs.
▶ Search for best-scoring structure.
▶ Simpler scoring ⇒ easier search.
▶ Accurate (best we have).
▶ Well studied.
▶ Well understood.
▶ Strong theoretical foundations.
▶ Slow, especially with rich scoring functions.

Greedy decoding
▶ Start with an unparsed sentence.
▶ Apply locally-optimal actions until sentence is parsed.
▶ Don’t look back.
▶ Use whatever features you want.
▶ Surprisingly accurate.
▶ Can be extremely fast.
▶ Still less accurate than search-based.
▶ Very little theoretical work.
▶ Not well understood.
Approaches to parsing

Global Optimization

- **Define** a scoring function over <sentence,tree> pairs.
- **Search** for best-scoring structure.
- Simpler scoring $\Rightarrow$ easier search.
Approaches to parsing

Global Optimization

- **Define** a scoring function over $<$sentence,tree$>$ pairs.
- **Search** for best-scoring structure.
- Simpler scoring $\Rightarrow$ easier search.
Approaches to parsing

Global Optimization

- Define a scoring function over \(<\text{sentence}, \text{tree}>\) pairs.
- Search for best-scoring structure.
- Simpler scoring ⇒ easier search.

Greedy decoding
Approaches to parsing

Global Optimization

- Define a scoring function over $<\text{sentence},\text{tree}>$ pairs.
- Search for best-scoring structure.
- Simpler scoring $\Rightarrow$ easier search.

Greedy decoding

- Start with an unparsed sentence.
- Apply *locally-optimal* actions until sentence is parsed.
- Don’t look back.
Approaches to parsing

Global Optimization

- Define a scoring function over <sentence, tree> pairs.
- Search for best-scoring structure.
- Simpler scoring $\Rightarrow$ easier search.

Greedy decoding

- Start with an unparsed sentence.
- Apply **locally-optimal** actions until sentence is parsed.
- Don’t look back.
Approaches to parsing

Global Optimization

- Accurate (best we have).
- Well studied.
- Well understood.
- Strong theoretical foundations.

argmax over combinatorial space

Greedy decoding

while (!done) { do best thing }
Approaches to parsing

**Global Optimization**  
argmax over combinatorial space

- Accurate (best we have).
- Well studied.
- Well understood.
- Strong theoretical foundations.
- Slow, especially with rich scoring functions.

**Greedy decoding**  
while (!done) { do best thing }
Approaches to parsing

Global Optimization    argmax over combinatorial space

- Accurate (best we have).
- Well studied.
- Well understood.
- Strong theoretical foundations.
- Slow, especially with rich scoring functions.

Greedy decoding    while (!done) { do best thing }

- Use whatever features you want.
- Surprisingly accurate.
- Can be extremely fast.
Approaches to parsing

Global Optimization

- Accurate (best we have).
- Well studied.
- Well understood.
- Strong theoretical foundations.
- Slow, especially with rich scoring functions.

Greedy decoding

while (!done) { do best thing }

- Use whatever features you want.
- Surprisingly accurate.
- Can be extremely fast.
- Still less accurate than search-based.
- Very little theoretical work.
- Not well understood.
Main question motivating this work

What’s the best we can do with the greedy approach?
Main question motivating this work

What’s the best we can do with the greedy approach?

Practical:
Better fast parsers!
  ▶ Provide accurate results.
  ▶ …without paying for more machines.
Main question motivating this work

What’s the best we can do with the greedy approach?

Practical:
Better fast parsers!
  ▶ Provide accurate results.
  ▶ … without paying for more machines.

Academic:
Incremental processing!
  ▶ Human processing probably more similar to greedy than to search.
  ▶ Cool learning problem.
  ▶ If we understand greedy better, we may do better on search-based.
The old world
(up until mid-2012)
An abstract machine composed of a stack and a buffer.

Machine is initialized with the words of a sentence.

A set of actions process the words by moving them from buffer to stack, removing them from the stack, or adding links between them.

A specific set of actions define a transition system.
The Arc-Eager Transition System

- **SHIFT** move first word from buffer to stack.
  
  (pre: Buffer not empty.)
The Arc-Eager Transition System

- **SHIFT** move first word from buffer to stack.
  (pre: Buffer not empty.)

- **LEFTARC** make first word in buffer head of top of stack, pop the stack.
  (pre: Stack not empty. Top of stack does not have a parent.)
The Arc-Eager Transition System

- **SHIFT** move first word from buffer to stack.
  
  (pre: Buffer not empty.)

- **LEFTARC** make first word in buffer head of top of stack, pop the stack.
  
  (pre: Stack not empty. Top of stack does not have a parent.)

- **RIGHTARC** make top of stack head of first in buffer, move first in buffer to stack.
  
  (pre: Buffer not empty.)
The Arc-Eager Transition System

- **SHIFT** move first word from buffer to stack.
  (pre: Buffer not empty.)

- **LEFTARC**$_{label}$ make first word in buffer head of top of stack, pop the stack.
  (pre: Stack not empty. Top of stack does not have a parent.)

- **RIGHTARC**$_{label}$ make top of stack head of first in buffer, move first in buffer to stack.
  (pre: Buffer not empty.)

- **REDUCE** pop the stack
  (pre: Stack not empty. Top of stack has a parent.)
She ate pizza with pleasure.
She ate pizza with pleasure
She ate pizza with pleasure
She ate pizza with pleasure
She ate pizza with pleasure.
Parsing Example

She ate pizza with pleasure
She ate pizza with pleasure
Parsing Example

She ate pizza with pleasure
She ate pizza with pleasure.
She ate pizza with pleasure.
She ate pizza with pleasure
What do we know about the arc-eager transition system?

- Every sequence of actions result in a valid projective structure.
- Every projective tree is derivable by (at least one) sequence of actions.
- Given a tree, finding a sequence of actions for deriving it. ("static oracle")
Parsing Algorithm

sequence ← oracle(sentence, tree)
configuration ← initialize(sentence)
while not configuration.IsFinal() do
    action ← sequence.next()
    configuration ← configuration.apply(action)
return configuration.tree
sequence ← oracle(sentence, tree)
configuration ← initialize(sentence)
while not configuration.IsFinal() do
    action ← sequence.next()
    configuration ← configuration.apply(action)
return configuration.tree
Parsing Algorithm

start with weight vector $w$
configuration $\leftarrow$ initialize(sentence)
while not configuration.IsFinal() do
  action $\leftarrow$ predict($w, \phi(\text{configuration})$)
  configuration $\leftarrow$ configuration.apply(action)
return configuration.tree

summarize the configuration as a feature vector
Parsing Algorithm

start with weight vector $w$
configuration $\leftarrow$ initialize(sentence)
while not configuration.IsFinal() do
    action $\leftarrow$ predict($w$, $\phi$(configuration))
    configuration $\leftarrow$ configuration.apply(action)
return configuration.tree

summarize the configuration as a feature vector

predict the action based on the features
 Parsing Algorithm

```
summarize the configuration as a feature vector

start with weight vector \( w \)
configuration \( \leftarrow \) initialize(sentence)
while not configuration.IsFinal() do
    action \( \leftarrow \) predict(\( w \), \( \phi \)(configuration))
    configuration \( \leftarrow \) configuration.apply(action)
return configuration.tree
```

predict the action based on the features

need to learn the correct weights
Training Algorithm

Learning a parser (online)

\[ w \leftarrow 0 \]

\[ \text{for sentence, tree pair in corpus do} \]
\[ \quad \text{sequence } \leftarrow \text{oracle(sentence, tree)} \]
\[ \quad \text{configuration } \leftarrow \text{initialize(sentence)} \]
\[ \text{while not configuration.IsFinal() do} \]
\[ \quad \text{action } \leftarrow \text{sequence.next()} \]
\[ \quad \text{features } \leftarrow \phi(\text{configuration}) \]
\[ \quad \text{predicted } \leftarrow \text{predict}(w, \phi(\text{configuration})) \]
\[ \text{if predicted } \neq \text{action then} \]
\[ \quad w.\text{update}(\phi(\text{configuration}), \text{action, predicted}) \]
\[ \quad \text{configuration } \leftarrow \text{configuration.apply(action)} \]
\[ \text{return } w \]
The new world
(2012 - )
How can we improve the training procedure?
How can we improve the training procedure?

re-examine the process.
Another look at training

Algorithm: Training with a static oracle

\[ w \leftarrow 0 \]

\begin{algorithm}
  for sentence, tree pair in corpus do
  
  sequence \leftarrow oracle(sentence, tree)
  
  conf \leftarrow initialize(sentence)
  
  while not conf.IsFinal() do
    
    action \leftarrow sequence.next()
    
    predicted \leftarrow predict(w, \phi(conf))
    
    if predicted \neq action then
      
      w.update(\phi(conf), action, predicted)
      
      conf \leftarrow conf.apply(action)
    
  return w
\end{algorithm}
Another look at training

Algorithm: Training with a static oracle

\[ w \leftarrow 0 \]
\[ \text{for sentence,tree pair in corpus do} \]
\[ \text{sequence } \leftarrow \text{oracle(sentence, tree)} \]
\[ \text{conf } \leftarrow \text{initialize(sentence)} \]
\[ \text{while not conf.IsFinal() do} \]
\[ \text{action } \leftarrow \text{sequence.next()} \]
\[ \text{predicted } \leftarrow \text{predict}(w, \phi(\text{conf})) \]
\[ \text{if predicted } \neq \text{action then} \]
\[ w.\text{update}(\phi(\text{conf}), \text{action, predicted}) \]
\[ \text{conf } \leftarrow \text{conf.apply(action)} \]
\[ \text{return } w \]
Algorithm: Training with a static oracle

\[ w \leftarrow 0 \]

\[ \text{for sentence, tree pair in corpus do} \]

\[ \text{sequence} \leftarrow \text{oracle}(\text{sentence, tree}) \]

\[ \text{conf} \leftarrow \text{initialize}(\text{sentence}) \]

\[ \text{while not conf.IsFinal()} \]

\[ \text{action} \leftarrow \text{sequence.next()} \]

\[ \text{predicted} \leftarrow \text{predict}(w, \phi(\text{conf})) \]

\[ \text{if predicted} \neq \text{action} \]

\[ w \text{.update}(\phi(\text{conf}), \text{action}, \text{predicted}) \]

\[ \text{conf} \leftarrow \text{conf.apply}(\text{action}) \]

\[ \text{return } w \]

Oracle produces a static, single sequence of actions to follow. Often, many sequences lead to the gold tree (spurious ambiguity).
Spurious Ambiguity

he wrote her a letter
Spurious Ambiguity

he wrote her a letter

SH
Spurious Ambiguity

Which sequence is better?

SH LEFT
Spurious Ambiguity

Which sequence is better?
Spurious Ambiguity

Which sequence is better?

SH LEFT SH RIGHT
Spurious Ambiguity

Which sequence is better?
Spurious Ambiguity

Which sequence is better?
Spurious Ambiguity

Which sequence is better?
Spurious Ambiguity

Which sequence is better?
Spurious Ambiguity

Which sequence is better?

SH LEFT SH RIGHT  RE SH LEFT RIGHT

SH LEFT RE RIGHT
Spurious Ambiguity

Which sequence is easier to learn?

SH LEFT SH RIGHT
RE SH LEFT RIGHT
SH LEFT RE RIGHT
Currently, oracles always prefer SH in a SH/RE conflict.

This is a reasonable choice: preferring RE often leads to somewhat worse scores.

But maybe we should let the classifier choose in a context-dependent way?

Which sequence is easier to learn?
Replacing the static oracle

Algorithm: Training with a dynamic oracle

\( w \leftarrow 0 \)

\textbf{for} sentence, tree pair in corpus \textbf{do}

\hspace{1em} \texttt{sequence} \leftarrow \texttt{oracle(sentence, tree)}

\hspace{1em} \texttt{conf} \leftarrow \texttt{initialize(sentence)}

\hspace{1em} \textbf{while} not \texttt{conf.IsFinal()} \textbf{do}

\hspace{2em} \texttt{predicted} \leftarrow \texttt{predict}(w, \phi(\texttt{conf}))

\hspace{2em} \texttt{action} \leftarrow \texttt{sequence.next()} \\
\hspace{2em} \textbf{if} predicted \neq \texttt{action} \textbf{then}

\hspace{3em} w.update(\phi(\texttt{conf}), \texttt{action}, \texttt{predicted})

\hspace{3em} \texttt{conf} \leftarrow \texttt{conf.apply(\texttt{action})}

\textbf{return} w
Replacing the static oracle

Algorithm: Training with a dynamic oracle

\[
\begin{align*}
w & \leftarrow 0 \\
\text{for} \text{ sentence, tree pair in corpus do} \\
\text{conf} & \leftarrow \text{initialize(sentence)} \\
\textbf{while} \text{ not conf.IsFinal() do} \\
\text{predicted} & \leftarrow \text{predict}(w, \phi(\text{conf})) \\
\text{action} & \leftarrow \text{predicted} \\
\text{if} \text{ not is\_allowed(predicted, conf, tree) then} \\
\text{action} & \leftarrow \text{highest scoring allowed action} \\
w & \text{.update}(\phi(\text{conf}), \text{action, predicted}) \\
\text{conf} & \leftarrow \text{conf.apply(action)} \\
\text{return} \ w
\end{align*}
\]
Replacing the static oracle

Algorithm: Training with a dynamic oracle

\[ w \leftarrow 0 \]

\textbf{for} sentence, tree pair in corpus \textbf{do} \\
\hspace{1cm} \text{conf} \leftarrow \text{initialize}(sentence) \\
\hspace{1cm} \text{while not conf.isFinal()} \textbf{do} \\
\hspace{2cm} \text{predicted} \leftarrow \text{predict}(w, \phi(\text{conf})) \\
\hspace{2cm} \text{action} \leftarrow \text{predicted} \\
\hspace{2cm} \text{if not is\_allowed(predicted, conf, tree) then} \\
\hspace{3cm} \text{action} \leftarrow \text{highest scoring allowed action} \\
\hspace{3cm} \text{w.update}(\phi(\text{conf}), \text{action}, \text{predicted}) \\
\hspace{2cm} \text{conf} \leftarrow \text{conf.apply}(\text{action}) \\
\hspace{1cm} \text{return } w \\

Instead of producing a single sequence upfront, the oracle is dynamically queried at each step.
Replacing the static oracle

Algorithm: Training with a dynamic oracle

\[ w \leftarrow 0 \]

\textbf{for} sentence, tree pair in corpus \textbf{do}

\[ \text{Instead of producing a single sequence upfront, the oracle is dynamically queried at each step.} \]

\textbf{while} not conf.IsFinal() \textbf{do}

 predicted \leftarrow \text{predict}(w, \phi(\text{conf}))

 action \leftarrow \text{predicted}

 if not is_allowed(predicted, conf, tree) then

 action \leftarrow \text{highest scoring allowed action}

 w.update(\phi(\text{conf}), action, predicted)

 conf \leftarrow \text{conf.apply(action)}

 return w

Great! but... where do we find such an oracle?
The static oracle is a function from a tree to a sequence of actions.

What we really want is a function from a configuration to a set of actions.

The function should answer
“if we take action A at configuration C, can we still reach the gold tree?”
Reachability

Easy question: Arc Reachability

I am at configuration C. Is there a sequence of actions that will result in the addition of arc \((i, j)\)?

Solution

It is very easy to construct such a sequence or prove that it cannot exist.
Arc Reachability in Arc-eager

Reachability of arc (H,M)

Trivial case: (H,M) was already derived – it’s reachable.
Arc Reachability in Arc-eager
Reachability of arc (H, M)

In order to add (H, M) we need:

- one item on top of stack, and the other first on buffer.
- Modifier should not have a head

\[ \ldots \quad F \quad G \quad H \quad I \quad J \quad K \quad L \quad M \quad N \quad \ldots \]
Arc Reachability in Arc-eager

Reachability of arc (H,M)

If one or more items are reduced, can’t add.

... F G H I J K L M N ...
Arc Reachability in Arc-eager
Reachability of arc (H,M)

If both items are on stack, can’t add.
Arc Reachability in Arc-eager
Reachability of arc (H,M)

One item on the buffer and other on the stack:
1. \texttt{SHIFT} until M if first in buffer.

\[
\begin{array}{c}
\ldots & F & G & H & I & J & K & L & M & N & \ldots \\
\end{array}
\]
Arc Reachability in Arc-eager

Reachability of arc (H,M)

One item on the buffer and other on the stack:
1. **SHIFT** until M if first in buffer.
2. **LEFT** or **REDUCE** until H on top of stack.
Arc Reachability in Arc-eager

Reachability of arc (H,M)

One item on the buffer and other on the stack:
1. **SHIFT** until M if first in buffer.
2. **LEFT** or **REDUCE** until H on top of stack.
Arc Reachability in Arc-eager

Reachability of arc (H,M)

One item on the buffer and other on the stack:
1. \textit{SHIFT} until M if first in buffer.
2. \textit{LEFT} or \textit{REDUCE} until H on top of stack.
Arc Reachability in Arc-eager
Reachability of arc (H,M)

One item on the buffer and other on the stack:
1. \textit{SHIFT} until M if first in buffer.
2. \textit{LEFT} or \textit{REDUCE} until H on top of stack.
Arc Reachability in Arc-eager
Reachability of arc (H,M)

One item on the buffer and other on the stack:
1. **SHIFT** until M if first in buffer.
2. **LEFT** or **REDUCE** until H on top of stack.
Arc Reachability in Arc-eager
Reachability of arc (H,M)

One item on the buffer and other on the stack:
1. \textsc{Shift} until M if first in buffer.
2. \textsc{Left} or \textsc{Reduce} until H on top of stack.
3. Add (H,M).
Arc Reachability in Arc-eager
Reachability of arc (H,M)
Arc Reachability in Arc-eager
Reachability of arc (H,M)

Both items are on the buffer:
1. **SHIFT** until one of them is on the stack.
Arc Reachability in Arc-eager

Reachability of arc (H,M)

Both items are on the buffer:
1. SHIFT until one of them is on the stack.
2. And we already did this case..

\[ \ldots \ F \ G \ H \ I \ J \ K \ L \ M \ N \ \ldots \]
Arc Reachability in Arc-eager
Reachability of arc (H,M)

Both items are on the buffer:
1. **SHIFT** until one of them is on the stack.
2. And we already did this case..
Arc Reachability in Arc-eager
Reachability of arc (H,M)

Both items are on the buffer:
1. **SHIFT** until one of them is on the stack.
2. And we already did this case..

\[ \ldots \ F \ G \ H \ I \ J \ K \ L \ M \ N \ \ldots \]
Arc Reachability in Arc-eager

Reachability of arc (H,M)

Both items are on the buffer:
1. \texttt{SHIFT} until one of them is on the stack.
2. And we already did this case..

\[
\begin{array}{cccccccc}
\ldots & F & G & H & I & J & K & L & M & N & \ldots \\
\end{array}
\]
Arc Reachability in Arc-eager

Reachability of arc (H,M)

Both items are on the buffer:
1. SHIFT until one of them is on the stack.
2. And we already did this case..

\[\ldots \quad F \quad G \quad H \quad I \quad J \quad K \quad L \quad M \quad N \quad \ldots\]
Arc Reachability in Arc-eager

Reachability of arc (H,M)

Both items are on the buffer:
1. *SHIFT* until one of them is on the stack.
2. And we already did this case..

\[ \ldots F G H I J K L M N \ldots \]
Both items are on the buffer:
1. **SHIFT** until one of them is on the stack.
2. And we already did this case..

\[ \ldots \ F \ G \ H \ I \ J \ K \ L \ M \ N \ldots \]
We can add arc (h,m) iff:

- The arc (h,m) already exist.
- \( m \) is not assigned a parent.
- either:
  - both \( m \) and \( h \) are on buffer,
  - one of them is on stack and other is on buffer.
Reachability

Easy question: Arc Reachability

I am at configuration C. Is there a sequence of actions that will result in the addition of arc \((i, j)\)?

Solution

It is very easy to construct such a sequence or prove that it cannot exist.

- For all transitions systems we know of, this can be answered in \(O(1)\), perhaps after a \(O(n)\) preprocessing.
Reachability

Easy question: Arc Reachability

I am at configuration C. Is there a sequence of actions that will result in the addition of arc \((i, j)\)?

Solution

It is very easy to construct such a sequence or prove that it cannot exist.

▶ For all transitions systems we know of, this can be answered in \(O(1)\), perhaps after a \(O(n)\) preprocessing.

The question we care about: Arc Set Reachability

I am at configuration C. Is there a sequence of actions that will result in the addition of all the arcs \((i_1, j_1) \cdots (i_k, j_k)\)?
Reachability

Easy question: Arc Reachability

\[ I \text{ am at configuration } C. \text{ Is there a sequence of actions that will result in the addition of arc } (i, j) \? \]

Solution

It is very easy to construct such a sequence or prove that it cannot exist. For all transition systems we know of, this can be answered in \( O(1) \), perhaps after a \( O(n) \) preprocessing.

The question we care about: Arc Set Reachability

\[ I \text{ am at configuration } C. \text{ Is there a sequence of actions that will result in the addition of all the arcs } (i_1, j_1) \cdot \cdot (i_k, j_k) \? \]
A formal property

Definition
A transition system is **arc decomposable** if it satisfies the following property:

If a set of arcs $\mathcal{A}$ are individually reachable from a configuration $C$, then every subset of them which can be extended to form a projective spanning tree is reachable from configuration $C$. 
The good stuff

Theorem (stated without proof)

*The ARCEAGER system is arc decomposable.*
The good stuff

Theorem (stated without proof)

*The ARCEAGER system is arc decomposable.*

Corollary

*We can work in terms of reachable arcs and reason about reachable structures.*
If we can reach all the gold arcs individually, we can reach the gold tree.

Stay in configurations from which all gold arcs are reachable.
A Dynamic Oracle

If we can reach all the gold arcs individually, we can reach the gold tree.

Stay in configurations from which all gold arcs are reachable.

\[ \text{allowed_actions}(\text{conf}, \mathcal{G}) = \{ \text{action} \mid \text{reachable_arcs}(\text{conf}.\text{apply}(<\text{action}>) \supseteq \mathcal{G} \} \]
Can calculate which arcs are lost by each action

**Example: SHIFT**

K is moved from buffer to stack.

**Before SHIFT**

K on buffer:
- can be modifier of all visible items - can be head of all head-less visible items

**After SHIFT**

K on stack:
- cannot head stack items
- cannot modify stack items
Replacing the static oracle

Algorithm: Training with a dynamic oracle

\[ w \leftarrow 0 \]
\[ \text{for sentence, tree pair in corpus do} \]

\[ \text{conf} \leftarrow \text{initialize(sentence)} \]
\[ \text{while not conf.IsFinal()} \text{ do} \]

\[ \text{predicted} \leftarrow \text{predict}(w, \phi(\text{conf})) \]
\[ \text{action} \leftarrow \text{predicted} \]
\[ \text{if not is_allowed(predicted, conf, tree) then} \]
\[ \quad \text{action} \leftarrow \text{highest scoring allowed action} \]
\[ \quad w.\text{update}(\phi(\text{conf}), \text{action}, \text{predicted}) \]
\[ \text{conf} \leftarrow \text{conf.apply(action)} \]
\[ \text{return } w \]
Some numbers: training with spurious ambiguity

Parsing Accuracies on various English datasets

- WSJ22
- WSJ23
- Brown
- Questions
- BNC
- Answers / web
- Emails / web
- Groups / web
- Reviews / web
- Blogs / web
- Football

<table>
<thead>
<tr>
<th>Parsing Accuracy</th>
<th>Greedy</th>
<th>Greedy + Ambiguity</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSJ22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSJ23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Questions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Answers / web</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Emails / web</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Groups / web</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reviews / web</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blogs / web</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Football</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

avg: 0.25        max: 0.5
Some numbers: training with spurious ambiguity

 Parsing Accuracies on the CoNLL data

<table>
<thead>
<tr>
<th>Language</th>
<th>Greedy</th>
<th>Greedy + Ambiguity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basque</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Catalan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chinese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Czech</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greek</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hungarian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Italian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turkish</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

avg: 0.03    max: 0.8
Some numbers: training with spurious ambiguity mostly beneficial.
Some numbers: training with spurious ambiguity

but we are not done yet!
Another look at training, take 2

Algorithm: Training with a dynamic oracle

\[w \leftarrow 0\]
\[\text{for sentence,tree pair in corpus do}\]

\[\text{conf } \leftarrow \text{initialize(sentence)}\]
\[\text{while not conf.IsFinal() do}\]

\[\text{predicted } \leftarrow \text{predict}(w, \phi(\text{conf}))\]
\[\text{action } \leftarrow \text{predicted}\]
\[\text{if not is_allowed(predicted, conf, tree) then}\]

\[\text{action } \leftarrow \text{highest scoring allowed action}\]
\[w.\text{update}(\phi(\text{conf}), \text{action}, \text{predicted})\]

\[\text{conf } \leftarrow \text{conf.apply(action)}\]

\[\text{return } w\]
Algorithm: Training with a dynamic oracle

\[ w \leftarrow 0 \]

for sentence,tree pair in corpus do

conf \leftarrow \text{initialize}(\text{sentence})

while not conf.IsFinal() do

predicted \leftarrow \text{predict}(w, \phi(\text{conf}))

action \leftarrow \text{predicted}

if not \text{is\_allowed}(\text{predicted}, \text{conf}, \text{tree}) then

action \leftarrow \text{highest scoring allowed action}

w.update(\phi(\text{conf}), \text{action}, \text{predicted})

conf \leftarrow \text{conf.apply}(\text{action})

return w
Another look at training, take 2

Algorithm: Training with a dynamic oracle

\[ w \leftarrow 0 \]

for sentence, tree pair in corpus

conf \leftarrow initialize(sentence)

while not conf.IsFinal()

predicted \leftarrow predict(w, \phi(conf))

action \leftarrow predicted

if not is_allowed(predicted, conf, tree) then

action \leftarrow highest scoring allowed action

w.update(\phi(conf), action, predicted)

conf \leftarrow conf.apply(action)

return w

Always following a correct action.

At training, parser never sees the result of incorrect actions.
So what?

Always taking an allowed action is bad!

Training is attempting to learn the optimal action for a configuration

▶ We may reach a configuration we haven't seen before.
▶ We may need to react differently to configurations we have seen before.
So what?

Always taking an allowed action is bad!

Training is attempting to learn the optimal action for a configuration, assuming that all previous actions were correct!
So what?

Always taking an allowed action is bad!

Training is attempting to learn the optimal action for a configuration, **assuming that all previous actions were correct!**

At parsing time, we may make a mistake

Once we err once in parsing time, the training assumptions no longer hold:

- We may reach a configuration we haven’t seen before.
- We may need to react differently to configurations we *have* seen before.
always try to make the training conditions as similar as possible to the testing conditions!
always try to make the training conditions as similar as possible to the testing conditions!

We need to expose the parser to configurations that result from following incorrect actions, and to the **optimal actions** to take in these configurations.
always try to make the training conditions as similar as possible to the testing conditions!

We need to expose the parser to configurations that result from following incorrect actions, and to the **optimal actions** to take in these configurations.

What’s the optimal action?
The oracles so far are well defined **only** for configurations which can lead to the gold tree.

**What we really want**

an oracle which is **well defined** and **optimal** for **every** possible configuration.
What does it mean to behave optimally after a mistake?

A reasonable definition of optimality:

1. If the gold tree is reachable: allow actions leading to the gold tree.
2. If the gold tree is not reachable: allow actions leading to the best reachable tree.

"Best reachable tree" is a reachable tree with minimum hamming loss to the gold tree.

Thanks to arc-decomposition:

A small tweak to our oracle will make it behave optimally for any configuration and return a cost for each action/configuration pair.
What does it mean to behave optimally after a mistake?

A reasonable definition of optimality

- If gold-tree is reachable: allow actions leading to gold tree.
- If gold-tree is **not** reachable: allow actions leading to the **best reachable tree**.
What does it mean to behave optimally after a mistake?

A reasonable definition of optimality

- If gold-tree is reachable: allow actions leading to gold tree.
- If gold-tree is not reachable: allow actions leading to the best reachable tree.

“Best reachable tree”

- a reachable tree with minimum hamming loss to gold tree.
  ⇒ under arc decomposition: a tree containing all the reachable arcs from the gold tree
What does it mean to behave optimally after a mistake?

A reasonable definition of optimality

- If gold-tree is reachable: allow actions leading to gold tree.
- If gold-tree is not reachable: allow actions leading to the best reachable tree.

“Best reachable tree”

- a reachable tree with minimum hamming loss to gold tree.

⇒ under arc decomposition: a tree containing all the reachable arcs from the gold tree

Thanks to arc-decomposition

- A small tweak to our oracle will make it behave optimally for any configuration
- and return a cost for each action/configuration pair.
Actions’ costs and optimal actions

The cost of an action $a$ at a configuration $C$

The number of gold arcs that are reachable from $C$ but cannot be reached after taking action $a$. 

The number of gold arcs that are reachable from $C$ but cannot be reached after taking action $a$. 

$$\text{lost}_\text{arcs}(a, \text{conf}) = \text{reachable}_\text{arcs}(\text{conf}) \setminus \text{reachable}_\text{arcs}(\text{conf}. \text{apply}(a)) \quad (1)$$

The action cost $a$ at a configuration $C$ is:

$$\text{action}_\text{cost}(a, \text{conf}, \text{gold}) = |\text{lost}_\text{arcs}(a, \text{conf}) \cap \text{gold}| \quad (2)$$

Optimal actions have a cost of 0.

There is always an optimal action.
Actions’ costs and optimal actions

The cost of an action $a$ at a configuration $C$

The number of gold arcs that are reachable from $C$ but cannot be reached after taking action $a$.

\[
\text{lost_arcs}(\text{action}, \text{conf}) = \\left( \text{reachable_arcs}(\text{conf}) \ \backslash \ \text{reachable_arcs}(\text{conf}.\text{apply}(\text{action})) \right) \quad (1)
\]

\[
\text{action_cost}(\text{action}, \text{conf}, \text{gold}) = |\text{lost_arcs}(\text{action}, \text{conf}) \cap \text{gold}| \quad (2)
\]
Actions’ costs and optimal actions

The cost of an action $a$ at a configuration $C$

The number of gold arcs that are reachable from $C$ but cannot be reached after taking action $a$.

\[
\text{lost}\_\text{arcs}(\text{action}, \text{conf}) = \text{reachable}\_\text{arcs}(\text{conf}) \cap \text{reachable}\_\text{arcs}(\text{conf}.\text{apply}(\text{action}))
\]

\[\text{action}\_\text{cost}(\text{action}, \text{conf}, \text{gold}) = |\text{lost}\_\text{arcs}(\text{action}, \text{conf}) \cap \text{gold}| \]

Optimal actions have a cost of 0

There is always an optimal action.
Algorithm: Training with a dynamic oracle and exploration

\[ w \leftarrow 0 \]

\[ \text{for sentence,tree pair in corpus do} \]
\[ \quad \text{conf} \leftarrow \text{initialize(sentence)} \]
\[ \quad \text{while not conf.IsFinal()} \text{ do} \]
\[ \quad \quad \text{predicted} \leftarrow \text{predict}(w, \phi(\text{conf})) \]
\[ \quad \quad \text{action} \leftarrow \text{highest scoring allowed action} \]
\[ \quad \quad \text{if action\_cost(predicted, conf, tree)!= 0 then} \]
\[ \quad \quad \quad w.\text{update}(\phi(\text{conf}), \text{action}, \text{predicted}) \]
\[ \quad \quad \text{conf} \leftarrow \text{conf.apply(action)} \]
\[ \text{return } w \]
Algorithm: Training with a dynamic oracle and exploration

\[ w \leftarrow 0 \]

\textbf{for} sentence, tree pair in corpus \textbf{do}

\hspace{1em} conf \leftarrow initialize(sentence)

\textbf{while} not conf.IsFinal() \textbf{do}

\hspace{2em} predicted \leftarrow predict(w, \phi(conf))

\hspace{2em} action \leftarrow \text{highest scoring allowed action}

\hspace{2em} \textbf{if} \text{ action\_cost(predicted, conf, tree) \neq 0 \textbf{then}}

\hspace{3em} w.update(\phi(conf), action, predicted)

\hspace{2em} conf \leftarrow conf.apply(action)

\textbf{return} \ w
Algorithm: Training with a dynamic oracle and exploration

\[ w \leftarrow 0 \]
for sentence, tree pair in corpus do
    conf \leftarrow initialize(sentence)
    while not conf.IsFinal() do
        predicted \leftarrow predict(\( w, \phi(\text{conf}) \))
        action \leftarrow \text{highest scoring allowed action}
        if action\_cost(predicted, conf, tree)!\(= 0 \) then
            \( w \).update(\( \phi(\text{conf}), \text{action}, \text{predicted} \))
        sometimes
            conf \leftarrow conf.apply(action)
        other times
            conf \leftarrow conf.apply(predicted)
    return \( w \)
Some numbers: training with exploration

- Avg: 1.1
- Max: 1.4

English Results

- WSJ22
- WSJ23
- Brown
- Questions
- BNC
- Answers / web
- Emails / web
- Groups / web
- Reviews / web
- Blogs / web
- Football

Parsing Accuracy

Greedy
Greedy + Ambiguity
Greedy + Exploration
Some numbers: training with exploration

avg: 1.1    max: 2.6
Some numbers: training with exploration

Very meaningful improvements on (almost) all datasets.
Reachability + arc-decomposition: a strong tool

Concrete things we now have

- Able to calculate $lost_{\text{arcs}}(action, conf)$.
- “Dynamic” oracle which is optimal for every configuration (and can work with partial evidence)
- Parsing algorithm for training with the dynamic oracle.
- More accurate deterministic parsing.
Take home:

Overall
- Better understanding $\Rightarrow$ better algorithms $\Rightarrow$ better results

Parser Users:
- Greedy parsers which are as-fast but more accurate than before.
- Ability to relate parsing mistakes to specific actions.

Parsing researchers:
- Stop using static-oracles. Our dynamic one is better.
- Training with spurious ambiguity can help.
- Do exploration when training whenever possible.
- A new way to reason about transition systems.
Extra
Some things we did not know (all related):

- How does an action affect the set of reachable structures?
  - It removes all structures containing the set of arcs which are “lost” by the action.

- What is the set of projective structures reachable from an arbitrary configuration?
  - All the projective structures which can be built from reachable arcs.

- What does a “SHIFT” mean?
  - Arc-Eager: “token on the buffer collected all its left modifiers, and is expecting a head to its right.”
For ML people

Does this remind you of *searn*? it should!

▶ Our “optimal everywhere” dynamic oracle is a good *policy* in the searn sense.
▶ Earlier attempts to use searn for parsing didn’t work.
   ⇒ They were missing the policy component.
▶ Now it's available.