Advanced Lecture on Dependency Parsing: Dynamic Oracles and Online Reordering

Miryam de Lhoneux

UPPSALA
UNIVERSITET

14 December 2017
Outline

1. Arc-Hybrid
2. Dynamic Oracles
3. Reordering
4. Dynamic Oracles and Reordering
Outline for section 1

1. Arc-Hybrid
2. Dynamic Oracles
3. Reordering
4. Dynamic Oracles and Reordering
### Configuration:

<table>
<thead>
<tr>
<th>STACK</th>
<th>BUFFER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drive</td>
<td>friend</td>
</tr>
<tr>
<td>your</td>
<td>home</td>
</tr>
<tr>
<td></td>
<td>root</td>
</tr>
</tbody>
</table>

### Transitions:

Kuhlmann et al. (2011)
Transition-Based Parsing with Arc-Hybrid

Configuration:

STACK

Drive    your

BUFFER

friend   home   root

Transitions:

LEFT-ARC

Kuhlmann et al. (2011)
Transition-Based Parsing with Arc-Hybrid

Configuration:

STACK

Drive    your

BUFFER

friend   home   root

Transitions:

LEFT-ARC

Kuhlmann et al. (2011)
Transition-Based Parsing with Arc-Hybrid

Configuration:

STACK

Drive your

BUFFER

friend home root

Transitions:

LEFT–ARC

Kuhlmann et al. (2011)
Transition-Based Parsing with Arc-Hybrid

Configuration:

STACK

Drive your

BUFFER

friend home root

Transitions:

LEFT−ARC

RIGHT−ARC

Kuhlmann et al. (2011)
Transition-Based Parsing with Arc-Hybrid

Configuration:

STACK
Drive     your

BUFFER
friend   home   root

Transitions:

LEFT-ARC

RIGHT-ARC

Kuhlmann et al. (2011)
Transition-Based Parsing with Arc-Hybrid

Configuration:
STACK
Drive your
BUFFER
friend home root

Transitions:
LEFT−ARC
RIGHT−ARC

Kuhlmann et al. (2011)
Transition-Based Parsing with Arc-Hybrid

Configuration:

STACK

BUFFER

Drive your

friend home root

Transitions:

LEFT-ARC

RIGHT-ARC

SHIFT

Kuhlmann et al. (2011)
Transition-Based Parsing with Arc-Hybrid

Configuration:

STACK

BUFFER

Drive your

friend home root

Transitions:

LEFT–ARC

RIGHT–ARC

SHIFT

Kuhlmann et al. (2011)
Algorithm 2 Online training with a static oracle

1: \( w \leftarrow 0 \)
2: \textbf{for} \( I = 1 \rightarrow \text{ITERATIONS} \) \textbf{do}
3: \quad \textbf{for} \ sentence \( x \) with gold tree \( G_{\text{gold}} \) in corpus \textbf{do}
4: \quad \quad \( c \leftarrow c_s(x) \)
5: \quad \quad \textbf{while} \( c \) is not terminal \textbf{do}
6: \quad \quad \quad \( t_p \leftarrow \arg \max_t w \cdot \phi(c, t) \)
7: \quad \quad \quad \( t_o \leftarrow o(c, G_{\text{gold}}) \)
8: \quad \quad \quad \textbf{if} \( t_p \neq t_o \) \textbf{then}
9: \quad \quad \quad \quad \( w \leftarrow w + \phi(c, t_o) - \phi(c, t_p) \)
10: \quad \quad \quad \( c \leftarrow t_o(c) \)
11: \quad \quad \textbf{return} \( w \)

\textbf{Figure} : Figure taken from Goldberg and Nivre (2012)
Algorithm 2 Online training with a static oracle

1: \( w \leftarrow 0 \)
2: for \( I = 1 \rightarrow \text{ITERATIONS} \) do
3: \hspace{1em} for sentence \( x \) with gold tree \( G_{\text{gold}} \) in corpus do
4: \hspace{2em} \( c \leftarrow c_s(x) \)
5: \hspace{1em} while \( c \) is not terminal do
6: \hspace{2em} \( t_p \leftarrow \arg \max_t w \cdot \phi(c, t) \)
7: \hspace{2em} \( t_o \leftarrow o(c, G_{\text{gold}}) \)
8: \hspace{2em} if \( t_p \neq t_o \) then
9: \hspace{2.5em} \( w \leftarrow w + \phi(c, t_o) - \phi(c, t_p) \)
10: \hspace{2em} \( c \leftarrow t_o(c) \)
11: \hspace{1em} return \( w \)

**Figure**: Figure taken from Goldberg and Nivre (2012)
Static Oracle for Arc-Hybrid

Drive ➔ your ➔ friend ➔ home ➔ **root**
Static Oracle for Arc-Hybrid

[ ] [Drive your friend home **root**]
Static Oracle for Arc-Hybrid

SHIFT

Drive  your  friend  home  **root**

[ Drive ]  [ your friend home **root** ]
Static Oracle for Arc-Hybrid

SHIFT

Drive your friend home **root**

[ Drive your ] [ friend home **root** ]
Drive your friend home **root**
Static Oracle for Arc-Hybrid

SHIFT

Drive → your → friend → home → **root**

[Drive friend] [home **root**]

Miryam de Lhoneux
Advanced Lecture on Dependency Parsing
Static Oracle for Arc-Hybrid

RIGHT-ARC

Drive your friend home **root**

[Drive] [home **root**]
Static Oracle for Arc-Hybrid

SHIFT

Drive → your → friend → home → **root**

[ Drive home] [**root**]
Static Oracle for Arc-Hybrid

RIGHT-ARC

Drive your friend home **root**

[ Drive ] [ **root** ]
Static Oracle for Arc-Hybrid

LEFT-ARC

Drive, your, friend, home, **root**

[] [**root**]
Write the transition sequence for this tree:
Outline for section 2

1. Arc-Hybrid
2. Dynamic Oracles
3. Reordering
4. Dynamic Oracles and Reordering
Algorithm 2 Online training with a static oracle

1: \( w \leftarrow 0 \)
2: \textbf{for} \( I = 1 \rightarrow \text{ ITERATIONS } \) \textbf{do}
3: \quad \textbf{for} sentence \( x \) with gold tree \( G_{\text{gold}} \) in corpus \textbf{do}
4: \quad \quad \( c \leftarrow c_s(x) \)
5: \quad \quad \textbf{while} \( c \) is not terminal \textbf{do}
6: \quad \quad \quad \( t_p \leftarrow \arg\max_t w \cdot \phi(c,t) \)
7: \quad \quad \quad \( t_o \leftarrow o(c,G_{\text{gold}}) \)
8: \quad \quad \quad \textbf{if} \( t_p \neq t_o \) \textbf{then}
9: \quad \quad \quad \quad \( w \leftarrow w + \phi(c,t_o) - \phi(c,t_p) \)
10: \quad \quad \quad \quad \( c \leftarrow t_o(c) \)
11: \quad \textbf{return} \( w \)

\textbf{Figure} : Figure taken from Goldberg and Nivre (2012)
Algorithm 2 Online training with a static oracle

1: \( w \leftarrow 0 \)
2: for \( I = 1 \rightarrow \text{ITERATIONS} \) do
3: for sentence \( x \) with gold tree \( G_{\text{gold}} \) in corpus do
4: \( c \leftarrow c_s(x) \)
5: while \( c \) is not terminal do
6: \( t_p \leftarrow \arg \max_t w \cdot \phi(c, t) \)
7: \( t_o \leftarrow o(c, G_{\text{gold}}) \)
8: if \( t_p \neq t_o \) then
9: \( w \leftarrow w + \phi(c, t_o) - \phi(c, t_p) \)
10: \( c \leftarrow t_o(c) \)

11: return \( w \)

**Figure**: Figure taken from Goldberg and Nivre (2012)
Spurious ambiguity

Send him a flower

[ Send him ]    [ a flower ]
Spurious ambiguity

RIGHT-ARC

Send him a flower

[ Send ] [ a flower ]
Spurious ambiguity

[ Send him ] [ a flower ]
Spurious ambiguity

Send him a flower

[ Send him a ]  [ flower ]
Spurious ambiguity

LEFT-ARC

Send him a flower

[ Send him ] [ flower ]
Spurious ambiguity

RIGHT-ARC

Send him a flower

[ Send ] [ flower ]
Algorithm 2 Online training with a static oracle

1: \( w \leftarrow 0 \)
2: \textbf{for} \( I = 1 \rightarrow \text{ITERATIONS} \) \textbf{do}
3: \textbf{for} sentence \( x \) with gold tree \( G_{\text{gold}} \) in corpus \textbf{do}
4: \( c \leftarrow c_s(x) \)
5: \textbf{while} \( c \) is not terminal \textbf{do}
6: \( t_p \leftarrow \arg \max_t w \cdot \phi(c,t) \)
7: \( t_o \leftarrow o(c,G_{\text{gold}}) \)
8: \textbf{if} \( t_p \neq t_o \) \textbf{then}
9: \( w \leftarrow w + \phi(c,t_o) - \phi(c,t_p) \)
10: \( c \leftarrow t_o(c) \)
11: \textbf{return} \( w \)

**Figure**: Figure taken from Goldberg and Nivre (2012)
Static Oracle: problem 2

Algorithm 2 Online training with a static oracle

1: \( w \leftarrow 0 \)
2: \( \text{for } I = 1 \rightarrow \text{ITERATIONS } \text{do} \)
3: \( \text{for sentence } x \text{ with gold tree } G_{\text{gold}} \text{ in corpus do} \)
4: \( c \leftarrow c_s(x) \)
5: \( \text{while } c \text{ is not terminal do} \)
6: \( t_p \leftarrow \arg\max_t w \cdot \phi(c, t) \)
7: \( t_o \leftarrow o(c, G_{\text{gold}}) \)
8: \( \text{if } t_p \neq t_o \text{ then} \)
9: \( w \leftarrow w + \phi(c, t_o) - \phi(c, t_p) \)
10: \( c \leftarrow t_o(c) \) \hspace{1cm} \text{We always apply the correct transition}
\hspace{1cm} \text{We only see 'gold' configurations in training!}
11: \( \text{return } w \)

Figure : Figure taken from Goldberg and Nivre (2012)
Algorithm 3 Online training with exploration for greedy transition-based parsers ($i$th iteration)

1: for sentence $W$ with gold tree $T$ in corpus do
2:     $c \leftarrow \text{INITIAL}(W)$
3:     while not TERMINAL($c$) do
4:         $\text{CORRECT}(c) \leftarrow \{t | o(t; c, T) = \text{true}\}$
5:         $t_p \leftarrow \arg\max_{t \in \text{LEGAL}(c)} w \cdot \phi(c, t)$
6:         $t_o \leftarrow \arg\max_{t \in \text{CORRECT}(c)} w \cdot \phi(c, t)$
7:         if $t_p \notin \text{CORRECT}(c)$ then
8:             $\text{UPDATE}(w, \phi(c, t_o), \phi(c, t_p))$
9:             $c \leftarrow \text{EXPLORE}_{k, p}(c, t_o, t_p, i)$
10:        else
11:            $c \leftarrow t_p(c)$

1: function $\text{EXPLORE}_{k, p}(c, t_o, t_p, i)$
2:     if $i > k$ and $\text{RAND()} < p$ then
3:         return $t_p(c)$
4:     else
5:         return $\text{NEXT}(c, t_o)$

Figure: Figure taken from Goldberg and Nivre (2013)
Training with a Dynamic Oracle

Algorithm 3 Online training with exploration for greedy transition-based parsers (ith iteration)

1: for sentence $W$ with gold tree $T$ in corpus do
2: \hspace{1em} $c \leftarrow \text{INITIAL}(W)$
3: \hspace{1em} while not TERMINAL($c$) do
4: \hspace{2.5em} \text{CORRECT}($c$) $\leftarrow \{t \mid o(t; c, T) = \text{true}\}$
5: \hspace{2.5em} $t_p \leftarrow \arg \max_{t \in \text{LEGAL}(c)} w \cdot \phi(c, t)$
6: \hspace{2.5em} $t_o \leftarrow \arg \max_{t \in \text{CORRECT}(c)} w \cdot \phi(c, t)$
7: \hspace{2.5em} if $t_p \not\in \text{CORRECT}(c)$ then
8: \hspace{4em} \text{UPDATE}(w, \phi(c, t_o), \phi(c, t_p))
9: \hspace{4em} $c \leftarrow \text{EXPLORE}_{k, p}(c, t_o, t_p, i)$
10: \hspace{2.5em} else
11: \hspace{3em} $c \leftarrow t_p(c)$

function \text{EXPLORE}_{k, p}(c, t_o, t_p, i)
1: \hspace{1em} if $i > k$ and \text{RAND}() < $p$ then
2: \hspace{2em} return $t_p(c)$
3: \hspace{2em} else
4: \hspace{3em} return $\text{NEXT}(c, t_o)$

Figure: Figure taken from Goldberg and Nivre (2013)
Algorithm 3 Online training with exploration for greedy transition-based parsers ($i$th iteration)

1: for sentence $W$ with gold tree $T$ in corpus do
2:     $c \leftarrow \text{INITIAL}(W)$
3:     while not $\text{TERMINAL}(c)$ do
4:         $\text{CORRECT}(c) \leftarrow \{ t | o(t; c, T) = \text{true} \}$
5:         $t_p \leftarrow \arg \max_{t \in \text{LEGAL}(c)} w \cdot \phi(c, t)$
6:         $t_o \leftarrow \arg \max_{t \in \text{CORRECT}(c)} w \cdot \phi(c, t)$
7:         if $t_p \notin \text{CORRECT}(c)$ then
8:             $\text{UPDATE}(w, \phi(c, t_o), \phi(c, t_p))$
9:             $c \leftarrow \text{EXPLORE}_{k, p}(c, t_o, t_p, i)$
10:        else
11:            $c \leftarrow t_p(c)$
12:        end

1: function $\text{EXPLORE}_{k, p}(c, t_o, t_p, i)$
2:     if $i > k$ and $\text{RAND()} < p$ then
3:         return $t_p(c)$
4:     else
5:         return $\text{NEXT}(c, t_o)$

Figure: Figure taken from Goldberg and Nivre (2013)
Dynamic Oracle for Arc-Hybrid

Goldberg and Nivre (2013)
Dynamic Oracle for Arc-Hybrid

RIGHT-ARC

Drive → your → friend → home → **root**

[ Drive your ] [ friend home **root** ]

Goldberg and Nivre (2013)
Dynamic Oracle for Arc-Hybrid

RIGHT-ARC

[ Drive ]
[ friend home **root** ]

Goldberg and Nivre (2013)
Dynamic Oracle for Arc-Hybrid

Goldberg and Nivre (2013)
Dynamic Oracle for Arc-Hybrid

SHIFT

Drive your friend home **root**

[ Drive your ] [ friend home **root** ]

Goldberg and Nivre (2013)
Dynamic Oracle for Arc-Hybrid

SHIFT

Drive your friend home **root**

[ Drive your friend ] [ home **root** ]

Goldberg and Nivre (2013)
Dynamic Oracle for Arc-Hybrid

SHIFT

[ Drive your friend ] -- cost+=1 -- [ home **root** ]

Goldberg and Nivre (2013)
Dynamic Oracle for Arc-Hybrid

- $C(\text{LEFT}; c, T)$: Adding the arc $(b, s_0)$ and popping $s_0$ from the stack means that $s_0$ will not be able to acquire heads from $H = \{s_1\} \cup \beta$ and will not be able to acquire dependents from $D = \{b\} \cup \beta$. The cost is therefore the number of arcs in $T$ of the form $(s_0, d)$ and $(h, s_0)$ for $h \in H$ and $d \in D$.

- $C(\text{RIGHT}; c, T)$: Adding the arc $(s_1, s_0)$ and popping $s_0$ from the stack means that $s_0$ will not be able to acquire heads or dependents from $B = \{b\} \cup \beta$. The cost is therefore the number of arcs in $T$ of the form $(s_0, d)$ and $(h, s_0)$ for $h, d \in B$.

- $C(\text{SHIFT}; c, T)$: Pushing $b$ onto the stack means that $b$ will not be able to acquire heads from $H = \{s_1\} \cup \sigma$, and will not be able to acquire dependents from $D = \{s_0, s_1\} \cup \sigma$. The cost is therefore the number of arcs in $T$ of the form $(b, d)$ and $(h, b)$ for $h \in H$ and $d \in D$.

**Figure**: Cost function
Exercise!

For the 7 first transitions in the correct transition sequence we defined in the previous exercise, compute the cost of the other legal transitions.
Outline for section 3

1. Arc-Hybrid
2. Dynamic Oracles
3. Reordering
4. Dynamic Oracles and Reordering
Arc-Hybrid Parsing with Reordering

Configuration:

STACK

BUFFER

Drive your friend home root

Transitions:

LEFT−ARC

RIGHT−ARC

SHIFT

Nivre (2009); de Lhoneux et al. (2017)
Arc-Hybrid Parsing with Reordering

Configuration:

STACK

BUFFER

Drive your

friend home root

Transitions:

LEFT-ARC

RIGHT-ARC

SHIFT

SWAP

Nivre (2009); de Lhonneux et al. (2017)
Arc-Hybrid Parsing with Reordering

Configuration:

STACK          BUFFER

Drive your friend home root

Transitions:

LEFT−ARC
RIGHT−ARC
SHIFT
SWAP

Nivre (2009); de Lhoneux et al. (2017)
Arc-Hybrid Parsing with Reordering

found best example ever

(Thanks Carlos Gomez-Rodriguez for the example!)
Arc-Hybrid Parsing with Reordering

found ➔ best ➔ example ➔ ever

Miryam de Lhoneux
Advanced Lecture on Dependency Parsing
Arc-Hyrid Parsing with Reordering

found  best  example  ever
1      2      4      3
Arc-Hybrid Parsing with Reordering

SHIFT

[ ] [ found₁ best₂ example₄ ever₃ ]
Arc-Hybrid Parsing with Reordering

SHIFT

[ found_1 ] \[ best_2 \text{ example}_4 \text{ ever}_3 \]
Arc-Hybrid Parsing with Reordering

SHIFT

[ found\textsubscript{1} best\textsubscript{2} ] \quad [ example\textsubscript{4} ever\textsubscript{3} ]
Arc-Hyrid Parsing with Reordering

SHIFT

[ found₁ best₂ example₄ ] [ ever₃ ]
Arc-Hybrid Parsing with Reordering

SHIFT

[ found$_1$ best$_2$ example$_4$ ]  [ ever$_3$ ]
Arc-Hybrid Parsing with Reordering

SWAP

[ found₁ best₂ ] [ ever₃ example₄ ]
Arc-Hyrid Parsing with Reordering

SHIFT

[ found\textsubscript{1} best\textsubscript{2} ever\textsubscript{3} ] [ example\textsubscript{4} ]
Arc-Hyrid Parsing with Reordering

RIGHT-ARC

[ found₁ best₂ ] [ example₄ ]
Arc-Hybrid Parsing with Reordering

**LEFT-ARC**

```
found  best  example  ever
1      2      4      3
```

[ found₁ ]  [ example₄ ]
Arc-Hyrid Parsing with Reordering

SHIFT

[ found\textsubscript{1} example\textsubscript{4} ] [ ]
Arc-Hybrid Parsing with Reordering

RIGHT-ARC

[ found\textsubscript{1} ] [ ]

found  best  example  ever
1      2      4      3
Exercise!

Write the transition sequence for this tree:

Stay away as$_1$ far as$_2$ possible . **root**

1 2 3 6 4 5 7 8

(Example taken from the UD English treebank.)
Outline for section 4

1. Arc-Hybrid
2. Dynamic Oracles
3. Reordering
4. Dynamic Oracles and Reordering
Dynamic oracle for parsing with reordering:
Open research question!

Partial solution:
A static-dynamic oracle (de Lhoneux et al., 2017)
Dynamic oracle for parsing with reordering:
Open research question!

Partial solution:
A static-dynamic oracle (de Lhoneux et al., 2017)
Dynamic oracle for parsing with reordering:
Open research question!

Partial solution:
A static-dynamic oracle (de Lhoneux et al., 2017)
Dynamic oracle for parsing with reordering: Open research question!

Partial solution: A static-dynamic oracle (de Lhoneux et al., 2017)
Dynamic oracle for parsing with reordering:
Open research question!

Partial solution:
A static-dynamic oracle (de Lhoneux et al., 2017)
Why is it hard?

- $C(\text{LEFT}; c, T)$: Adding the arc $(b, s_0)$ and popping $s_0$ from the stack means that $s_0$ will not be able to acquire heads from $H = \{s_1\} \cup \beta$ and will not be able to acquire dependents from $D = \{b\} \cup \beta$. The cost is therefore the number of arcs in $T$ of the form $(s_0, d)$ and $(h, s_0)$ for $h \in H$ and $d \in D$.

- $C(\text{RIGHT}; c, T)$: Adding the arc $(s_1, s_0)$ and popping $s_0$ from the stack means that $s_0$ will not be able to acquire heads or dependents from $B = \{b\} \cup \beta$. The cost is therefore the number of arcs in $T$ of the form $(s_0, d)$ and $(h, s_0)$ for $h, d \in B$.

- $C(\text{SHIFT}; c, T)$: Pushing $b$ onto the stack means that $b$ will not be able to acquire heads from $H = \{s_1\} \cup \sigma$, and will not be able to acquire dependents from $D = \{s_0, s_1\} \cup \sigma$. The cost is therefore the number of arcs in $T$ of the form $(b, d)$ and $(h, b)$ for $h \in H$ and $d \in D$. 
Why is it hard?

- \( C(\text{LEFT}; c, T) \): Adding the arc \((b, s_0)\) and popping \(s_0\) from the stack means that \(s_0\) will not be able to acquire heads from \(H = \{s_1\} \cup \beta\) and will not be able to acquire dependents from \(D = \{b\} \cup \beta\). The cost is therefore the number of arcs in \(T\) of the form \((s_0, d)\) and \((h, s_0)\) for \(h \in H\) and \(d \in D\).

- \( C(\text{RIGHT}; c, T) \): Adding the arc \((s_1, s_0)\) and popping \(s_0\) from the stack means that \(s_0\) will not be able to acquire heads or dependents from \(B = \{b\} \cup \beta\). The cost is therefore the number of arcs in \(T\) of the form \((s_0, d)\) and \((h, s_0)\) for \(h, d \in B\).

- \( C(\text{SHIFT}; c, T) \): Pushing \(b\) onto the stack means that \(b\) will not be able to acquire heads from \(H = \{s_1\} \cup \sigma\), and will not be able to acquire dependents from \(D = \{s_0, s_1\} \cup \sigma\). The cost is therefore the number of arcs in \(T\) of the form \((b, d)\) and \((h, b)\) for \(h \in H\) and \(d \in D\).
Why is it hard?

Using information about the position of words in stack and buffer
Why is it hard?

Using information about the position of words in stack and buffer
But now words can move!
Algorithm 3 Online training with exploration for greedy transition-based parsers (ith iteration)

1: for sentence $W$ with gold tree $T$ in corpus do
2:     $c \leftarrow$ INITIAL($W$)
3:     while not TERMINAL($c$) do
4:         $\text{CORRECT}(c) \leftarrow \{t \mid o(t; c, T) = \text{true}\}$
5:         $t_p \leftarrow \arg \max_{t \in \text{LEGAL}(c)} w \cdot \phi(c, t)$
6:         $t_o \leftarrow \arg \max_{t \in \text{CORRECT}(c)} w \cdot \phi(c, t)$
7:         if $t_p \notin \text{CORRECT}(c)$ then
8:             $\text{UPDATE}(w, \phi(c, t_o), \phi(c, t_p))$
9:             $c \leftarrow \text{EXPLORE}_{k,p}(c, t_o, t_p, i)$
10:        else
11:            $c \leftarrow t_p(c)$

1: function EXPLORE$_{k,p}(c, t_o, t_p, i)$
2:     if $i > k$ and RAND() < $p$ then
3:         return $t_p(c)$
4:     else
5:         return NEXT$(c, t_o)$

Figure: Figure taken from Goldberg and Nivre (2013)
### Algorithm 3: Online training with exploration for greedy transition-based parsers ($i$th iteration)

1. for sentence $W$ with gold tree $T$ in corpus do
2. \hspace{1em} $c \leftarrow \text{INITIAL}(W)$
3. \hspace{1em} while not $\text{TERMINAL}(c)$ do
4. \hspace{2em} $\text{CORRECT}(c) \leftarrow \{t \mid o(t; c, T) = \text{true}\}$
5. \hspace{2em} $t_p \leftarrow \arg \max_{t \in \text{LEGAL}(c)} w \cdot \phi(c, t)$
6. \hspace{2em} $t_o \leftarrow \arg \max_{t \in \text{CORRECT}(c)} w \cdot \phi(c, t)$
7. \hspace{2em} if $t_p \not\in \text{CORRECT}(c)$ then
8. \hspace{3em} $\text{UPDATE}(w, \phi(c, t_o), \phi(c, t_p))$
9. \hspace{3em} $c \leftarrow \text{EXPLORE}_{k,p}(c, t_o, t_p, i)$
10. else \hspace{1em} We disallow this if the correct transition is swap
11. \hspace{1em} $c \leftarrow t_p(c)$

1. function $\text{EXPLORE}_{k,p}(c, t_o, t_p, i)$
2. \hspace{1em} if $i > k$ and $\text{RAND}() < p$ then
3. \hspace{2em} return $t_p(c)$
4. \hspace{1em} else
5. \hspace{2em} return $\text{NEXT}(c, t_o)$

---

**Figure**: Figure taken from Goldberg and Nivre (2013)
A Static-Dynamic Oracle

found
1

best
2

dynamic
4

example

ever
3

[ found$_1$ best$_2$ ]  [ example$_4$ ever$_3$ ]
A Static-Dynamic Oracle

RIGHT-ARC

[found₁] [example₄ ever₃]
A Static-Dynamic Oracle

RIGHT-ARC

\[
\begin{align*}
\text{found} & \quad \text{best} & \quad \text{example} & \quad \text{ever} \\
1 & \quad 2 & \quad 4 & \quad 3 \\
\end{align*}
\]

[ found\textsubscript{1} ] [ example\textsubscript{4} ever\textsubscript{3} ]
A Static-Dynamic Oracle

LEFT-ARC

[ found₁ ] [ example₄ ever₃ ]
A Static-Dynamic Oracle

LEFT-ARC

\[ \text{found}_1 \quad \text{best} \quad \text{example}_4 \quad \text{ever}_3 \]

\[ \text{cost} += 1 \]


