Dual Decomposition for Parsing with Non-Projective Head Automata

Terry Koo, Alexander M. Rush, Michael Collins, David Sontag, and Tommi Jaakkola
The Cost of Model Complexity

We are always looking for better ways to model natural language.

Tradeoff: Richer models $\Rightarrow$ Harder decoding

Added complexity is both computational and implementational.
The Cost of Model Complexity

We are always looking for better ways to model natural language.

Tradeoff: Richer models $\Rightarrow$ Harder decoding

Added complexity is both computational and implementational.

Tasks with challenging decoding problems:

- Speech Recognition
- Sequence Modeling (e.g. extensions to HMM/CRF)
- Parsing
- Machine Translation
The Cost of Model Complexity

We are always looking for better ways to model natural language.

Tradeoff: Richer models ⇒ Harder decoding

Added complexity is both computational and implementational.

Tasks with challenging decoding problems:

- Speech Recognition
- Sequence Modeling (e.g. extensions to HMM/CRF)
- Parsing
- Machine Translation

\[ y^* = \arg \max_y f(y) \quad \text{Decoding} \]
Non-Projective Dependency Parsing

Important problem in many languages.

Problem is **NP-Hard** for all but the simplest models.
Dual Decomposition

A classical technique for constructing decoding algorithms.

Solve complicated models

\[ y^* = \arg \max_y f(y) \]

by decomposing into smaller problems.

Upshot: Can utilize a toolbox of combinatorial algorithms.

- Dynamic programming
- Minimum spanning tree
- Shortest path
- Min-Cut
- ...
A Dual Decomposition Algorithm
for Non-Projective Dependency Parsing

**Simple** - Uses basic combinatorial algorithms

**Efficient** - Faster than previously proposed algorithms

**Strong Guarantees** - Gives a certificate of optimality when exact

Solves 98% of examples exactly, even though the problem is NP-Hard

**Widely Applicable** - Similar techniques extend to other problems
Non-Projective Dependency Parsing

- Starts at the root symbol *
- Each word has a exactly one parent word
- Produces a tree structure (no cycles)
- Dependencies can cross
Algorithm Outline

Arc-Factored Model

Sibling Model
Algorithm Outline

Arc-Factored Model

Dual Decomposition

Sibling Model
Arc-Factored

\[ f(y) = \]

\[
\begin{align*}
&\text{score}(\text{head} = *, \mod = \text{saw}) + \\
&\text{score}(\text{saw}, \text{John}) + \\
&\text{score}(\text{saw}, \text{movie}) + \\
&\text{score}(\text{saw}, \text{today}) + \\
&\text{score}(\text{movie}, \text{a}) + \\
&... \\
\end{align*}
\]

e.g. \[\text{score}(\ast, \text{saw}) = \log p \left( \text{saw} \mid \ast \right) \] (generative model)

or \[\text{score}(\ast, \text{saw}) = w \cdot \phi \left( \text{saw}, \ast \right) \] (CRF/perceptron model)

\[ y^* = \arg \max_y f(y) \]
Arc-Factored

\[ f(y) = \text{score}(\text{head} = \ast_0, \text{mod} = \text{saw}_2) \]
Arc-Factored

\[
f(y) = score(\text{head} = \ast_0, \text{mod} = \text{saw}_2) + score(\text{saw}_2, \text{John}_1)
\]
Arc-Factored

\[ f(y) = \text{score}(\text{head} = *_0, \text{mod} = \text{saw}_2) + \text{score}(\text{saw}_2, \text{John}_1) \]
\[ + \text{score}(\text{saw}_2, \text{movie}_4) \]
Arc-Factored

\[ f(y) = \text{score}(\text{head} = *_0, \text{mod} = \text{saw}_2) + \text{score}(\text{saw}_2, \text{John}_1) + \text{score}(\text{saw}_2, \text{movie}_4) + \text{score}(\text{saw}_2, \text{today}_5) \]
Arc-Factored

\[ f(y) = \text{score(head = } *_0, \text{ mod = saw}_2) + \text{score(saw}_2, \text{ John}_1) + \text{score(saw}_2, \text{ movie}_4) + \text{score(saw}_2, \text{ today}_5) + \text{score(movie}_4, \text{ a}_3) + ... \]
f(y) = \text{score}(\text{head} = *_0, \text{mod} = \text{saw}_2) + \text{score}(\text{saw}_2, \text{John}_1) \\
+ \text{score}(\text{saw}_2, \text{movie}_4) + \text{score}(\text{saw}_2, \text{today}_5) \\
+ \text{score}(\text{movie}_4, \text{a}_3) + ... \\

\text{e.g. } \text{score}(*_0, \text{saw}_2) = \log p(\text{saw}_2|*_0) \quad \text{(generative model)}
Arc-Factored

\[
f(y) = \text{score}(\text{head} = *_0, \text{mod} = \text{saw}_2) + \text{score}(\text{saw}_2, \text{John}_1) + \text{score}(\text{saw}_2, \text{movie}_4) + \text{score}(\text{saw}_2, \text{today}_5) + \text{score}(\text{movie}_4, \text{a}_3) + \ldots
\]

e.g. \( \text{score}(*_0, \text{saw}_2) = \log p(\text{saw}_2|*_0) \) (generative model)

or \( \text{score}(*_0, \text{saw}_2) = w \cdot \phi(\text{saw}_2, *_0) \) (CRF/perceptron model)
Arc-Factored

$$f(y) = score(head = *_0, mod = saw_2) + score(saw_2, John_1)$$

$$+ score(saw_2, movie_4) + score(saw_2, today_5)$$

$$+ score(movie_4, a_3) + ...$$

e.g. \( score(*_0, saw_2) = \log p(saw_2 | *_0) \) (generative model)

or \( score(*_0, saw_2) = w \cdot \phi(saw_2, *_0) \) (CRF/perceptron model)

\( y^* = \arg \max_y f(y) \) ⇐ Minimum Spanning Tree Algorithm
Sibling Models

\[ f(y) = \]

- \( *_0 \) John\(_1\) saw\(_2\) a\(_3\) movie\(_4\) today\(_5\) that\(_6\) he\(_7\) liked\(_8\)
Sibling Models

\[ f(y) = \text{score}(\text{head} = *_0, \text{prev} = \text{NULL}, \text{mod} = \text{saw}_2) \]
Sibling Models

\[ f(y) = \text{score}(\text{head} = *_0, \text{prev} = \text{NULL}, \text{mod} = \text{saw}_2) \]

\[ + \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) \]
Sibling Models

\[ f(y) = score(head = *, prev = NULL, mod = saw) + score(saw, NULL, John) + score(saw, NULL, movie) + \ldots \]

\[ e.g. \quad score(saw, movie, today) = \log p(today | saw, movie) \]

or

\[ score(saw, movie, today) = w \cdot \phi(saw, movie, today) \]
f(y) = score(head = *₀, prev = NULL, mod = saw₂)

+ score(saw₂, NULL, John₁) + score(saw₂, NULL, movie₄)

+ score(saw₂, movie₄, today₅) + ...
Sibling Models

\[ f(y) = score(head = *_0, \text{prev} = \text{NULL}, \text{mod} = \text{saw}_2) \]

\[ + score(saw_2, \text{NULL}, \text{John}_1) + score(saw_2, \text{NULL}, \text{movie}_4) \]

\[ + score(saw_2, \text{movie}_4, \text{today}_5) + \ldots \]

e.g. \( score(saw_2, \text{movie}_4, \text{today}_5) = \log p(\text{today}_5|saw_2, \text{movie}_4) \)
Sibling Models

\[ f(y) = \text{score}(\text{head} = *_0, \text{prev} = \text{NULL}, \text{mod} = \text{saw}_2) \]
\[ + \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{movie}_4) \]
\[ + \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) + \ldots \]

e.g. \[ \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) = \log p(\text{today}_5|\text{saw}_2, \text{movie}_4) \]
\[ \text{or } \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) = w \cdot \phi(\text{saw}_2, \text{movie}_4, \text{today}_5) \]
Sibling Models

\[ f(y) = \text{score}(\text{head} = *_0, \text{prev} = \text{NULL}, \text{mod} = \text{saw}_2) + \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{movie}_4) + \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) + \ldots \]

e.g. \( \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) = \log p(\text{today}_5 | \text{saw}_2, \text{movie}_4) \)

or \( \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) = w \cdot \phi(\text{saw}_2, \text{movie}_4, \text{today}_5) \)

\[ y^* = \arg \max_y f(y) \leftarrow \text{NP-Hard} \]
Thought Experiment: Individual Decoding

*ₐ₀ John₁ saw₂ a₃ movie₄ today₅ that₆ he₇ liked₈
Thought Experiment: Individual Decoding

\[ \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{movie}_4) + \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) \]
Thought Experiment: Individual Decoding

\[
score(saw_2, NULL, John_1) + score(saw_2, NULL, movie_4) \\
+ score(saw_2, movie_4, today_5) \\

\]

\[
= score(saw_2, NULL, John_1) + score(saw_2, NULL, that_6)
\]
Thought Experiment: Individual Decoding

*0 John1 saw2 a3 movie4 today5 that6 he7 liked8

\[
score(saw_2, \text{NULL}, John_1) + score(saw_2, \text{NULL}, movie_4) \\
+ score(saw_2, movie_4, today_5)
\]

\[
score(saw_2, \text{NULL}, John_1) + score(saw_2, \text{NULL}, that_6)
\]

\[
score(saw_2, \text{NULL}, a_3) + score(saw_2, a_3, he_7)
\]
Thought Experiment: Individual Decoding

*₀ John₁ saw₂ a₃ movie₄ today₅ that₆ he₇ liked₈

\[
2^{n-1} \text{ possibilities}
\]

\[
\begin{align*}
\text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + & \text{score}(\text{saw}_2, \text{NULL}, \text{movie}_4) \\
+ & \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5)
\end{align*}
\]

\[
\begin{align*}
\text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + & \text{score}(\text{saw}_2, \text{NULL}, \text{that}_6) \\
\end{align*}
\]

\[
\begin{align*}
\text{score}(\text{saw}_2, \text{NULL}, \text{a}_3) + & \text{score}(\text{saw}_2, \text{a}_3, \text{he}_7)
\end{align*}
\]
Thought Experiment: Individual Decoding

\[
\begin{align*}
&\text{Under Sibling Model, can solve for each word with Viterbi decoding.}
\end{align*}
\]
Thought Experiment Continued

$*$

John$_1$ saw$_2$ a$_3$ movie$_4$ today$_5$ that$_6$ he$_7$ liked$_8$

Idea: Do individual decoding for each head word using dynamic programming.

If we’re lucky, we’ll end up with a valid final tree.
Thought Experiment Continued

Idea: Do individual decoding for each head word using dynamic programming.

If we’re lucky, we’ll end up with a valid final tree.
Thought Experiment Continued

Idea: Do individual decoding for each head word using dynamic programming.

If we’re lucky, we’ll end up with a valid final tree.
Idea: Do individual decoding for each head word using dynamic programming.

If we’re lucky, we’ll end up with a valid final tree.
Thought Experiment Continued

Idea: Do individual decoding for each head word using dynamic programming.

If we’re lucky, we’ll end up with a valid final tree.
Thought Experiment Continued

Idea: Do individual decoding for each head word using dynamic programming.

If we’re lucky, we’ll end up with a valid final tree.
Thought Experiment Continued

Idea: Do individual decoding for each head word using dynamic programming.

If we’re lucky, we’ll end up with a valid final tree.
Thought Experiment Continued

Idea: Do individual decoding for each head word using dynamic programming.

If we’re lucky, we’ll end up with a valid final tree.
Thought Experiment Continued

Idea: Do individual decoding for each head word using dynamic programming.

If we’re lucky, we’ll end up with a valid final tree.

But we might violate some constraints.
Dual Decomposition Idea

<table>
<thead>
<tr>
<th></th>
<th>No Constraints</th>
<th>Tree Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sibling Model</td>
<td>Individual Decoding</td>
<td>Minimum Spanning Tree</td>
</tr>
<tr>
<td>Arc-Factored</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dual Decomposition Idea

- No Constraints
- Tree Constraints

Arc-Factored
- Minimum Spanning Tree

Sibling Model
- Individual Decoding
- Dual Decomposition
Dual Decomposition Structure

Goal $y^* = \arg \max_{y \in \mathcal{Y}} f(y)$

Valid Trees

- All Possible
- Sibling
- Arc-Factored
- Constraint
Goal \( y^* = \arg \max_{y \in \mathcal{Y}} f(y) \)

Rewrite as \( \arg \max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f(z) + g(y) \) such that \( z = y \)
Dual Decomposition Structure

Goal $y^* = \arg \max_{y \in Y} f(y)$

Rewrite as $\arg \max_{z \in Z, y \in Y} f(z) + g(y)$

such that $z = y$
Dual Decomposition Structure

Goal \( y^* = \arg \max_{y \in \mathcal{Y}} f(y) \)

Rewrite as \( \arg \max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f(z) + g(y) \)

such that \( z = y \)

Valid Trees

All Possible
Goal $y^* = \arg \max_{y \in \mathcal{Y}} f(y)$

Rewrite as $\arg \max_{z \in \mathcal{Z}, \ y \in \mathcal{Y}} f(z) + g(y)$

such that $z = y$
Dual Decomposition Structure

Goal $y^* = \arg\max_{y \in \mathcal{Y}} f(y)$

Rewrite as $\arg\max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f(z) + g(y)$ such that $z = y$

Valid Trees

All Possible

Sibling

Arc-Factored

Valid Trees
Dual Decomposition Structure

Goal $y^* = \arg \max_{y \in \mathcal{Y}} f(y)$

Rewrite as $\arg \max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f(z) + g(y)$

such that $z = y$

Constraint

Sibling
Arc-Factored

All Possible
Valid Trees
Algorithm Sketch

Set penalty weights equal to 0 for all edges.

For $k = 1$ to $K$

...
Set penalty weights equal to 0 for all edges.

For \( k = 1 \) to \( K \)

\[ z^{(k)} \leftarrow \text{Decode} \ (f(z) + \text{penalty}) \text{ by Individual Decoding} \]
Algorithm Sketch

Set penalty weights equal to 0 for all edges.

For $k = 1$ to $K$

$z^{(k)} \leftarrow \text{Decode} \ (f(z) + \text{penalty}) \text{ by Individual Decoding}$

$y^{(k)} \leftarrow \text{Decode} \ (g(y) - \text{penalty}) \text{ by Minimum Spanning Tree}$
Algorithm Sketch

Set penalty weights equal to 0 for all edges.

For $k = 1$ to $K$

$z^{(k)} \leftarrow$ Decode $(f(z) + \text{penalty})$ by Individual Decoding

$y^{(k)} \leftarrow$ Decode $(g(y) - \text{penalty})$ by Minimum Spanning Tree

If $y^{(k)}(i, j) = z^{(k)}(i, j)$ for all $i, j$ Return $(y^{(k)}, z^{(k)})$
Algorithm Sketch

Set penalty weights equal to 0 for all edges.

For $k = 1$ to $K$

$z^{(k)} \leftarrow$ Decode $(f(z) + \text{penalty})$ by Individual Decoding

$y^{(k)} \leftarrow$ Decode $(g(y) - \text{penalty})$ by Minimum Spanning Tree

If $y^{(k)}(i, j) = z^{(k)}(i, j)$ for all $i, j$ Return $(y^{(k)}, z^{(k)})$

Else Update penalty weights based on $y^{(k)}(i, j) - z^{(k)}(i, j)$
Individual Decoding

\[ z^* = \underset{z \in Z}{\text{arg max}} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \underset{y \in Y}{\text{arg max}} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Key

- \( f(z) \) ⇐ Sibling Model  
- \( g(y) \) ⇐ Arc-Factored Model  
- \( Z \) ⇐ No Constraints  
- \( Y \) ⇐ Tree Constraints  
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)  

\[ u(i,j) = 0 \text{ for all } i,j \]
Individual Decoding

\[ z^* = \arg \max_{z \in Z} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in Y} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Key

- \( f(z) \) ⇐ Sibling Model
- \( g(y) \) ⇐ Arc-Factored Model
- \( Z \) ⇐ No Constraints
- \( Y \) ⇐ Tree Constraints
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)
**Individual Decoding**

\[ z^* = \arg \max_{z \in \mathcal{Z}} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

**Minimum Spanning Tree**

\[ y^* = \arg \max_{y \in \mathcal{Y}} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

**Penalties**

\[ u(i,j) = 0 \text{ for all } i,j \]

**Key**

- \( f(z) \) ⇐ Sibling Model
- \( g(y) \) ⇐ Arc-Factored Model
- \( \mathcal{Z} \) ⇐ No Constraints
- \( \mathcal{Y} \) ⇐ Tree Constraints
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)
Individual Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in \mathcal{Y}} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Key

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(z) )</td>
<td>Sibling Model</td>
</tr>
<tr>
<td>( \mathcal{Z} )</td>
<td>No Constraints</td>
</tr>
<tr>
<td>( y(i,j) = 1 )</td>
<td>if ( y ) contains dependency ( i,j )</td>
</tr>
<tr>
<td>( g(y) )</td>
<td>Arc-Factored Model</td>
</tr>
<tr>
<td>( \mathcal{Y} )</td>
<td>Tree Constraints</td>
</tr>
</tbody>
</table>

Penalties

\[ u(i,j) = 0 \text{ for all } i,j \]
Individual Decoding

\[ z^* = \arg \max_{z \in Z} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in \mathcal{Y}} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Key

- \( f(z) \) \( \leftarrow \) Sibling Model
- \( \mathcal{Z} \) \( \leftarrow \) No Constraints
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)
- \( g(y) \) \( \leftarrow \) Arc-Factored Model
- \( \mathcal{Y} \) \( \leftarrow \) Tree Constraints

Penalties

\[ u(i,j) = 0 \text{ for all } i,j \]

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( u(8,1) )</td>
<td>-1</td>
</tr>
<tr>
<td>( u(4,6) )</td>
<td>-1</td>
</tr>
<tr>
<td>( u(2,6) )</td>
<td>1</td>
</tr>
<tr>
<td>( u(8,7) )</td>
<td>1</td>
</tr>
</tbody>
</table>
Individual Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in \mathcal{Y}} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Penalties

\[
\begin{array}{c|c}
\text{Iteration 1} & \\
\hline
u(8, 1) & -1 \\
u(4, 6) & -1 \\
u(2, 6) & 1 \\
u(8, 7) & 1 \\
\end{array}
\]

Key

\[ f(z) \iff \text{Sibling Model} \quad g(y) \iff \text{Arc-Factored Model} \]
\[ \mathcal{Z} \iff \text{No Constraints} \quad \mathcal{Y} \iff \text{Tree Constraints} \]
\[ y(i,j) = 1 \quad \text{if} \quad y \text{ contains dependency } i,j \]
Individual Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in \mathcal{Y}} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Key

- \( f(z) \) ⇐ Sibling Model
- \( g(y) \) ⇐ Arc-Factored Model
- \( \mathcal{Z} \) ⇐ No Constraints
- \( \mathcal{Y} \) ⇐ Tree Constraints
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)

Penalties

\[
\begin{array}{c|c}
\text{Iteration 1} & \\
\hline
u(8, 1) & -1 \\
u(4, 6) & -1 \\
u(2, 6) & 1 \\
u(8, 7) & 1 \\
\end{array}
\]

\( u(i,j) = 0 \) for all \( i,j \)
Individual Decoding

\[ z^* = \arg \max_{z \in Z} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in Y} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Penalties

\[
\begin{array}{l|l}
\hline
\text{Iteration 1} & \\
\hline
u(8, 1) & -1 \\
u(4, 6) & -1 \\
u(2, 6) & 1 \\
u(8, 7) & 1 \\
\hline
\end{array}
\]

\[
\begin{array}{l|l}
\hline
\text{Iteration 2} & \\
\hline
u(8, 1) & -1 \\
u(4, 6) & -2 \\
u(2, 6) & 2 \\
u(8, 7) & 1 \\
\hline
\end{array}
\]

Key

- \( f(z) \) \iff Sibling Model
- \( g(y) \) \iff Arc-Factored Model
- \( Z \) \iff No Constraints
- \( Y \) \iff Tree Constraints
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)
Individual Decoding

\[ z^* = \arg \max_{z \in Z} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in Y} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Penalties

\[
\begin{array}{lcr}
\text{Iteration 1} \\
\hline
u(8, 1) & -1 \\
u(4, 6) & -1 \\
u(2, 6) & 1 \\
u(8, 7) & 1 \\
\hline
\text{Iteration 2} \\
\hline
u(8, 1) & -1 \\
u(4, 6) & -2 \\
u(2, 6) & 2 \\
u(8, 7) & 1 \\
\hline
\end{array}
\]

Key

- \( f(z) \) \( \Leftarrow \) Sibling Model
- \( g(y) \) \( \Leftarrow \) Arc-Factored Model
- \( Z \) \( \Leftarrow \) No Constraints
- \( Y \) \( \Leftarrow \) Tree Constraints
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)
Individual Decoding

\[ z^* = \arg \max_{z \in Z} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in Y} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Penalties

\[
\begin{array}{c|c}
\text{Iteration 1} & \\
\hline
u(8, 1) & -1 \\
u(4, 6) & -1 \\
u(2, 6) & 1 \\
u(8, 7) & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{Iteration 2} & \\
\hline
u(8, 1) & -1 \\
u(4, 6) & -2 \\
u(2, 6) & 2 \\
u(8, 7) & 1 \\
\end{array}
\]

Key

\[
\begin{array}{c|c|c}
\text{f(z)} & \leftarrow \text{Sibling Model} & \text{g(y)} & \leftarrow \text{Arc-Factored Model} \\
\mathcal{Z} & \leftarrow \text{No Constraints} & \mathcal{Y} & \leftarrow \text{Tree Constraints} \\
y(i,j) = 1 \text{ if } y \text{ contains dependency } i,j
\end{array}
\]
Individual Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in \mathcal{Y}} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Key

- \( f(z) \) ⇐ Sibling Model
- \( \mathcal{Z} \) ⇐ No Constraints
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)
- \( g(y) \) ⇐ Arc-Factored Model
- \( \mathcal{Y} \) ⇐ Tree Constraints

Penalties

\[
\begin{array}{c|c}
\text{Iteration 1} & \\
\hline
u(8,1) & -1 \\
u(4,6) & -1 \\
u(2,6) & 1 \\
u(8,7) & 1 \\
\hline
\text{Iteration 2} & \\
\hline
u(8,1) & -1 \\
u(4,6) & -2 \\
u(2,6) & 2 \\
u(8,7) & 1 \\
\end{array}
\]
Individual Decoding

\[ z^* = \arg \max_{z \in Z} (f(z) + \sum_{i,j} u(i,j)z(i,j)) \]

Minimum Spanning Tree

\[ y^* = \arg \max_{y \in Y} (g(y) - \sum_{i,j} u(i,j)y(i,j)) \]

Key

- \( f(z) \) ⇐ Sibling Model
- \( Z \) ⇐ No Constraints
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)

Penalties

\[
\begin{align*}
&\text{Iteration 1} \\
&u(8, 1) \quad -1 \\
&u(4, 6) \quad -1 \\
&u(2, 6) \quad 1 \\
&u(8, 7) \quad 1 \\
&\text{Iteration 2} \\
&u(8, 1) \quad -1 \\
&u(4, 6) \quad -2 \\
&u(2, 6) \quad 2 \\
&u(8, 7) \quad 1 \\
&\text{Converged} \\
&y^* = \arg \max_{y \in Y} f(y) + g(y)
\end{align*}
\]
Guarantees

Theorem

If at any iteration $y^{(k)} = z^{(k)}$, then $(y^{(k)}, z^{(k)})$ is the global optimum.

In experiments, we find the global optimum on 98% of examples.
Guarantees

Theorem

If at any iteration $y^{(k)} = z^{(k)}$, then $(y^{(k)}, z^{(k)})$ is the global optimum.

In experiments, we find the global optimum on 98% of examples.

If we do not converge to a match, we can still return an approximate solution (more in the paper).
Extensions

- Grandparent Models

\[ f(y) = \ldots + \text{score}(gp = *_0, \text{head} = \text{saw}_2, \text{prev} = \text{movie}_4, \text{mod} = \text{today}_5) \]

- Head Automata (Eisner, 2000)

Generalization of Sibling models

Allow arbitrary automata as local scoring function.
Roadmap

Algorithm

Experiments

Derivation
Experiments

Properties:

- Exactness
- Parsing Speed
- Parsing Accuracy
- Comparison to Individual Decoding
- Comparison to LP/ILP

Training:

- Averaged Perceptron (more details in paper)

Experiments on:

- CoNLL Datasets
- English Penn Treebank
- Czech Dependency Treebank
How often do we exactly solve the problem?

- Percentage of examples where the dual decomposition finds an exact solution.
Number of sentences parsed per second

Comparable to dynamic programming for projective parsing
### Accuracy

<table>
<thead>
<tr>
<th></th>
<th>Arc-Factored</th>
<th>Prev Best</th>
<th>Grandparent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>89.7</td>
<td>91.5</td>
<td>91.8</td>
</tr>
<tr>
<td>Dut</td>
<td>82.3</td>
<td>85.6</td>
<td>85.8</td>
</tr>
<tr>
<td>Por</td>
<td>90.7</td>
<td>92.1</td>
<td>93.0</td>
</tr>
<tr>
<td>Slo</td>
<td>82.4</td>
<td>85.6</td>
<td>86.2</td>
</tr>
<tr>
<td>Swe</td>
<td>88.9</td>
<td>90.6</td>
<td>91.4</td>
</tr>
<tr>
<td>Tur</td>
<td>75.7</td>
<td>76.4</td>
<td>77.6</td>
</tr>
<tr>
<td>Eng</td>
<td>90.1</td>
<td>—</td>
<td>92.5</td>
</tr>
<tr>
<td>Cze</td>
<td>84.4</td>
<td>—</td>
<td>87.3</td>
</tr>
</tbody>
</table>

Prev Best - Best reported results for CoNLL-X data set, includes:

- Approximate search (McDonald and Pereira, 2006)
- Loop belief propagation (Smith and Eisner, 2008)
- (Integer) Linear Programming (Martins et al., 2009)
Comparison to Subproblems

F$_1$ for dependency accuracy
Comparison to LP/ILP

Martins et al. (2009): Proposes two representations of non-projective dependency parsing as a linear programming relaxation as well as an exact ILP.

- LP (1)
- LP (2)
- ILP

Use an LP/ILP Solver for decoding

We compare:
- Accuracy
- Exactness
- Speed

Both LP and dual decomposition methods use the same model, features, and weights $w$. 
Comparison to LP/ILP: Accuracy

- All decoding methods have comparable accuracy
Comparison to LP/ILP: Exactness and Speed

Percentage with exact solution

Sentences per second
Roadmap

Algorithm

Experiments

Derivation
Deriving the Algorithm

**Goal:**
\[ y^* = \arg \max_{y \in \mathcal{Y}} f(y) \]

**Rewrite:**
\[ \arg \max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f(z) + g(y) \]
\[ \text{s.t. } z(i,j) = y(i,j) \text{ for all } i,j \]

**Lagrangian:**
\[ L(u, y, z) = f(z) + g(y) + \sum_{i,j} u(i,j) (z(i,j) - y(i,j)) \]
Deriving the Algorithm

Goal:
\[ y^* = \arg \max_{y \in Y} f(y) \]

Rewrite:
\[ \arg \max_{z \in Z, y \in Y} f(z) + g(y) \]

s.t. \( z(i,j) = y(i,j) \) for all \( i,j \)

Lagrangian:
\[ L(u, y, z) = f(z) + g(y) + \sum_{i,j} u(i,j) (z(i,j) - y(i,j)) \]

The dual problem is to find \( \min_u L(u) \) where

\[ L(u) = \max_{y \in Y, z \in Z} L(u, y, z) = \max_{z \in Z} \left( f(z) + \sum_{i,j} u(i,j)z(i,j) \right) \]

\[ + \max_{y \in Y} \left( g(y) - \sum_{i,j} u(i,j)y(i,j) \right) \]

Dual is an upper bound: \( L(u) \geq f(z^*) + g(y^*) \) for any \( u \)
A Subgradient Algorithm for Minimizing $L(u)$

$$L(u) = \max_{z \in Z} \left( f(z) + \sum_{i,j} u(i,j)z(i,j) \right) + \max_{y \in Y} \left( g(y) - \sum_{i,j} u(i,j)y(i,j) \right)$$

$L(u)$ is convex, but not differentiable. A subgradient of $L(u)$ at $u$ is a vector $g_u$ such that for all $v$,

$$L(v) \geq L(u) + g_u \cdot (v - u)$$

Subgradient methods use updates $u' = u - \alpha g_u$

In fact, for our $L(u)$, $g_u(i,j) = z^*(i,j) - y^*(i,j)$
Related Work

- Methods that use general purpose linear programming or integer linear programming solvers (Martins et al. 2009; Riedel and Clarke 2006; Roth and Yih 2005)
- Dual decomposition for inference in MRFs (Komodakis et al., 2007; Wainwright et al., 2005)
- Methods that incorporate combinatorial solvers within loopy belief propagation (Duchi et al. 2007; Smith and Eisner 2008)
Summary

\[ y^* = \arg \max_y f(y) \iff \text{NP-Hard} \]

Arc-Factored Model

Sibling Model
Summary

\[ y^* = \arg \max_y f(y) \iff \text{NP-Hard} \]

Arc-Factored Model

Dual Decomposition

Sibling Model
Other Applications

- Dual decomposition can be applied to other decoding problems.
- Rush et al. (2010) focuses on integrated dynamic programming algorithms.
  - Integrated Parsing and Tagging
  - Integrated Constituency and Dependency Parsing
y^* = \arg \max_y f(y) \iff \text{Slow}

HMM Model

CFG Model
Parsing and Tagging

\[ y^* = \arg \max_y f(y) \Leftarrow \text{Slow} \]

HMM Model

Dual Decomposition

CFG Model
Dependency and Constituency

\[ y^* = \arg \max_y f(y) \leftarrow \text{Slow} \]

Dependency Model

Lexicalized CFG
Dependency and Constituency

\[ y^* = \arg \max_y f(y) \iff \text{Slow} \]

Dependency Model

Dual Decomposition

Lexicalized CFG
Future Directions

There is much more to explore around dual decomposition in NLP.

▶ Known Techniques
  ▶ Generalization to more than two models
  ▶ K-best decoding
  ▶ Approximate subgradient
  ▶ Heuristic for branch-and-bound type search

▶ Possible NLP Applications
  ▶ Machine Translation
  ▶ Speech Recognition
  ▶ “Loopy” Sequence Models

▶ Open Questions
  ▶ Can we speed up subalgorithms when running repeatedly?
  ▶ What are the trade-offs of different decompositions?
  ▶ Are there better methods for optimizing the dual?
Appendix
Training the Model

\[ f(y) = \ldots + \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) + \ldots \]

- \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) = w \cdot \phi(\text{saw}_2, \text{movie}_4, \text{today}_5)

- Weight vector \( w \) trained using Averaged perceptron.

- (More details in the paper.)
Early Stopping
Caching speed