Dual Decomposition for Natural Language Processing

Alexander M. Rush and Michael Collins
Decoding complexity

**focus:** decoding problem for natural language tasks

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

**motivation:**

- richer model structure often leads to improved accuracy
- exact decoding for complex models tends to be intractable
Decoding tasks

many common problems are intractable to decode exactly

high complexity

• combined parsing and part-of-speech tagging (Rush et al., 2010)
• “loopy” HMM part-of-speech tagging
• syntactic machine translation (Rush and Collins, 2011)

NP-Hard

• symmetric HMM alignment (DeNero and Macherey, 2011)
• phrase-based translation (Chang and Collins, 2011)
• higher-order non-projective dependency parsing (Koo et al., 2010)

in practice:

• approximate decoding methods (coarse-to-fine, beam search, cube pruning, gibbs sampling, belief propagation)
• approximate models (mean field, variational models)
cannot hope to find exact algorithms (particularly when NP-Hard)

aim: develop decoding algorithms with formal guarantees

method:
  • derive fast algorithms that provide certificates of optimality
  • show that for practical instances, these algorithms often yield exact solutions
  • provide strategies for improving solutions or finding approximate solutions when no certificate is found

dual decomposition helps us develop algorithms of this form
Lagrangian relaxation (Held and Karp, 1971)

Important method from combinatorial optimization

Initially used for traveling salesman problems

Optimal tour - NP-Hard

![Optimal tour graph]

Optimal 1-tree - easy (MST)

![Optimal 1-tree graph]
Dual decomposition (Komodakis et al., 2010; Lemaréchal, 2001)

**goal:** solve complicated optimization problem

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

**method:** decompose into subproblems, solve iteratively

**benefit:** can choose decomposition to provide “easy” subproblems

**aim** for simple and efficient combinatorial algorithms

- dynamic programming
- minimum spanning tree
- shortest path
- min-cut
- bipartite match
- etc.
Related work

there are related methods used NLP with similar motivation

related methods:

- belief propagation (particularly max-product) (Smith and Eisner, 2008)
- factored A* search (Klein and Manning, 2003)
- exact coarse-to-fine (Raphael, 2001)

aim to find exact solutions without exploring the full search space
Tutorial outline

focus:
• developing dual decomposition algorithms for new NLP tasks
• understanding formal guarantees of the algorithms
• extensions to improve exactness and select solutions

outline:
1. worked algorithm for combined parsing and tagging
2. important theorems and formal derivation
3. more examples from parsing, sequence labeling, MT
4. practical considerations for implementing dual decomposition
5. relationship to linear programming relaxations
6. further variations and advanced examples
1. Worked example

**aim:** walk through a dual decomposition algorithm for combined parsing and part-of-speech tagging

- introduce formal notation for parsing and tagging
- give assumptions necessary for decoding
- step through a run of the dual decomposition algorithm
Combined parsing and part-of-speech tagging

**goal:** find parse tree that optimizes

\[
\text{score}(S \rightarrow NP \ VP) + \text{score}(VP \rightarrow V \ NP) + \\
... + \text{score}(N \rightarrow V) + \text{score}(N \rightarrow \text{United}) + ...
\]
Constituency parsing

notation:
- $\mathcal{Y}$ is set of constituency parses for input
- $y \in \mathcal{Y}$ is a valid parse
- $f(y)$ scores a parse tree

goal:

$$\arg \max_{y \in \mathcal{Y}} f(y)$$

eexample: a context-free grammar for constituency parsing

$$S \rightarrow NP \mid VP$$

$$NP \rightarrow N \mid NP$$

$$VP \rightarrow V \mid NP$$

$$N \rightarrow \text{United} \mid \text{some}$$

$$V \rightarrow \text{flies}$$

$$NP \rightarrow \text{D} \mid \text{A} \mid \text{N}$$

$$\text{A} \rightarrow \text{large}$$

$$\text{N} \rightarrow \text{jet}$$
Part-of-speech tagging

notation:
- $\mathcal{Z}$ is set of tag sequences for input
- $z \in \mathcal{Z}$ is a valid tag sequence
- $g(z)$ scores of a tag sequence

goal:
- $\arg \max_{z \in \mathcal{Z}} g(z)$

example: an HMM for part-of speech tagging

```
N → V → D → A → N
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow
United_1 \quad flies_2 \quad some_3 \quad large_4 \quad jet_5
```
Identifying tags

notation: identify the tag labels selected by each model

• \( y(i, t) = 1 \) when parse \( y \) selects tag \( t \) at position \( i \)
• \( z(i, t) = 1 \) when tag sequence \( z \) selects tag \( t \) at position \( i \)

example: a parse and tagging with \( y(4, A) = 1 \) and \( z(4, A) = 1 \)
Combined optimization

goal:

$$\underset{y \in \mathcal{Y}, z \in \mathcal{Z}}{\operatorname{arg\ max}} \ f(y) + g(z)$$

such that for all $i = 1 \ldots n$, $t \in \mathcal{T}$,

$$y(i, t) = z(i, t)$$

i.e. find the best parse and tagging pair that agree on tag labels

equivalent formulation:

$$\underset{y \in \mathcal{Y}}{\operatorname{arg\ max}} f(y) + g(l(y))$$

where $l : \mathcal{Y} \rightarrow \mathcal{Z}$ extracts the tag sequence from a parse tree
Dynamic programming intersection

can solve by solving the product of the two models

example:

- parsing model is a context-free grammar
- tagging model is a first-order HMM
- can solve as CFG and finite-state automata intersection

replace $S \rightarrow NP \ VP$

with

$S_{N,N} \rightarrow NP_{N,N} VP_{V,N}$

```
S
  NP
    N
      United
    V
      flies
  VP
    NP
      D
        some
      A
        large
      N
        jet
```
Parsing assumption

the structure of $\mathcal{Y}$ could be CFG, TAG, etc.

**assumption:** optimization with $u$ can be solved efficiently

$$\arg \max_{y \in \mathcal{Y}} f(y) + \sum_{i,t} u(i, t)y(i, t)$$

generally benign since $u$ can be incorporated into the structure of $f$

**example:** CFG with rule scoring function $h$

$$f(y) = \sum_{X \rightarrow Y \; Z \in y} h(X \rightarrow Y \; Z) + \sum_{(i, X) \in y} h(X \rightarrow w_i)$$

where

$$\arg \max_{y \in \mathcal{Y}} f(y) + \sum_{i,t} u(i, t)y(i, t) =$$

$$\arg \max_{y \in \mathcal{Y}} \sum_{X \rightarrow Y \; Z \in y} h(X \rightarrow Y \; Z) + \sum_{(i, X) \in y} (h(X \rightarrow w_i) + u(i, X))$$
Tagging assumption

we make a similar assumption for the set $\mathcal{Z}$

**assumption:** optimization with $u$ can be solved efficiently

$$
\arg\max_{z \in \mathcal{Z}} g(z) - \sum_{i,t} u(i, t)z(i, t)
$$

**example:** HMM with scores for transitions $T$ and observations $O$

$$
g(z) = \sum_{t \rightarrow t' \in z} T(t \rightarrow t') + \sum_{(i, t) \in z} O(t \rightarrow w_i)
$$

where

$$
\arg\max_{z \in \mathcal{Z}} g(z) - \sum_{i,t} u(i, t)z(i, t) =
$$

$$
\arg\max_{z \in \mathcal{Z}} \sum_{t \rightarrow t' \in z} T(t \rightarrow t') + \sum_{(i, t) \in z} (O(t \rightarrow w_i) - u(i, t))
$$
Dual decomposition algorithm

Set $u^{(1)}(i, t) = 0$ for all $i, t \in T$

For $k = 1$ to $K$

\[ y^{(k)} \leftarrow \arg \max_{y \in Y} f(y) + \sum_{i, t} u^{(k)}(i, t)y(i, t) \quad \text{[Parsing]} \]

\[ z^{(k)} \leftarrow \arg \max_{z \in Z} g(z) - \sum_{i, t} u^{(k)}(i, t)z(i, t) \quad \text{[Tagging]} \]

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

Else $u^{(k+1)}(i, t) \leftarrow u^{(k)}(i, t) - \alpha_k(y^{(k)}(i, t) - z^{(k)}(i, t))$
CKY Parsing

\[ y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t)) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t)) \]

Key

- \( f(y) \) ⇐ CFG
- \( g(z) \) ⇐ HMM
- \( \mathcal{Y} \) ⇐ Parse Trees
- \( \mathcal{Z} \) ⇐ Taggings
- \( y(i, t) = 1 \) if \( y \) contains tag \( t \) at position \( i \)

Penalties

\[ u(i, t) = 0 \text{ for all } i, t \]
CKY Parsing

\[ y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t)) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t)) \]

Key

\[ f(y) \quad \Leftarrow \quad \text{CFG} \]
\[ \mathcal{Y} \quad \Leftarrow \quad \text{Parse Trees} \]
\[ y(i, t) = 1 \text{ if } y \text{ contains tag } t \text{ at position } i \]

Penalties

\[ u(i, t) = 0 \text{ for all } i, t \]
CKY Parsing

\[
y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t))
\]

Viterbi Decoding

\[
z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t))
\]

Key

\[
f(y) \iff \text{CFG} \quad \quad g(z) \iff \text{HMM}
\]
\[
\mathcal{Y} \iff \text{Parse Trees} \quad \quad \mathcal{Z} \iff \text{Taggings}
\]
\[
y(i, t) = 1 \text{ if } y \text{ contains tag } t \text{ at position } i
\]
CKY Parsing

\[
\begin{array}{c}
S \\
\downarrow \\
\text{NP} & \text{VP} \\
\downarrow & \\
A & N & D & A & V \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{United} & \text{flies} & \text{some} & \text{large} & \text{jet} \\
\end{array}
\]

Penalties

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

Viterbi Decoding

\[
y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t))
\]

\[
z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t))
\]

Key

\[
f(y) \Leftarrow \text{CFG} \quad \quad g(z) \Leftarrow \text{HMM}
\]

\[
\mathcal{Y} \Leftarrow \text{Parse Trees} \quad \quad \mathcal{Z} \Leftarrow \text{Taggings}
\]

\[
y(i, t) = 1 \text{ if } y \text{ contains tag } t \text{ at position } i
\]
CKY Parsing

\[ y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t)) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t)) \]

Key

- \( f(y) \) \( \Leftarrow \) CFG
- \( \mathcal{Y} \) \( \Leftarrow \) Parse Trees
- \( y(i, t) = 1 \) if \( y \) contains tag \( t \) at position \( i \)

Penalties

\[ u(i, t) = 0 \] for all \( i, t \)

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u(1, A)</td>
<td>-1</td>
</tr>
<tr>
<td>u(1, N)</td>
<td>1</td>
</tr>
<tr>
<td>u(2, N)</td>
<td>-1</td>
</tr>
<tr>
<td>u(2, V)</td>
<td>1</td>
</tr>
<tr>
<td>u(5, V)</td>
<td>-1</td>
</tr>
<tr>
<td>u(5, N)</td>
<td>1</td>
</tr>
</tbody>
</table>
CKY Parsing

\[ y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t)) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t)) \]

Key

\[ f(y) \iff \text{CFG} \]
\[ g(z) \iff \text{HMM} \]
\[ \mathcal{Y} \iff \text{Parse Trees} \]
\[ \mathcal{Z} \iff \text{Taggings} \]
\[ y(i, t) = 1 \text{ if } y \text{ contains tag } t \text{ at position } i \]

Penalties

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

Iteration 1

\begin{align*}
  &u(1, A) &-1 \\
  &u(1, N) &1 \\
  &u(2, N) &-1 \\
  &u(2, V) &1 \\
  &u(5, V) &-1 \\
  &u(5, N) &1 
\end{align*}

Converged

\[ y^* = \arg \max_{y \in \mathcal{Y}} f(y) + g(y) \]
CKY Parsing

\[ y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t)) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t)) \]

Key

\[
\begin{align*}
    f(y) & \iff \text{CFG} & g(z) & \iff \text{HMM} \\
    \mathcal{Y} & \iff \text{Parse Trees} & \mathcal{Z} & \iff \text{Taggings} \\
    y(i, t) = 1 & \text{ if } y \text{ contains tag } t \text{ at position } i
\end{align*}
\]
CKY Parsing

\[ y^* = \arg \max_{y \in Y} (f(y) + \sum_{i, t} u(i, t)y(i, t)) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in Z} (g(z) - \sum_{i, t} u(i, t)z(i, t)) \]

Penalties

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

Iteration 1

\[
\begin{array}{l}
  u(1, A) \quad -1 \\
  u(1, N) \quad 1 \\
  u(2, N) \quad -1 \\
  u(2, V) \quad 1 \\
  u(5, V) \quad -1 \\
  u(5, N) \quad 1 \\
\end{array}
\]

Key

\[
\begin{align*}
  f(y) & \quad \Leftarrow \quad \text{CFG} \\
  Y & \quad \Leftarrow \quad \text{Parse Trees} \\
  y(i, t) = 1 & \quad \text{if } y \text{ contains tag } t \text{ at position } i \\
  g(z) & \quad \Leftarrow \quad \text{HMM} \\
  Z & \quad \Leftarrow \quad \text{Taggings}
\end{align*}
\]
CKY Parsing

\[ S \]
\[ \text{NP} \quad \text{VP} \]
\[ \text{N} \quad \text{V} \]
\[ \text{United} \quad \text{flies} \quad D \quad A \quad \text{NP} \]
\[ \text{A} \quad \text{some} \quad \text{large} \quad \text{jet} \]

\[ y^* = \arg \max_{y \in Y} (f(y) + \sum_{i,t} u(i, t)y(i, t)) \]

Penalties

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

\begin{tabular}{|l|c|}
\hline
\text{Iteration 1} & \\
\hline
\text{u(1, A)} & -1 \\
\text{u(1, N)} & 1 \\
\text{u(2, N)} & -1 \\
\text{u(2, V)} & 1 \\
\text{u(5, V)} & -1 \\
\text{u(5, N)} & 1 \\
\hline
\end{tabular}

\begin{tabular}{|l|c|}
\hline
\text{Iteration 2} & \\
\hline
\text{u(5, V)} & -1 \\
\text{u(5, N)} & 1 \\
\hline
\end{tabular}

Viterbi Decoding

\[ A \rightarrow N \rightarrow D \rightarrow A \rightarrow N \]
\[ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \]
\[ \text{United}_1 \quad \text{flies}_2 \quad \text{some}_3 \quad \text{large}_4 \quad \text{jet}_5 \]

\[ z^* = \arg \max_{z \in Z} (g(z) - \sum_{i,t} u(i, t)z(i, t)) \]

Key

\[ f(y) \leftarrow \text{CFG} \quad g(z) \leftarrow \text{HMM} \]
\[ Y \leftarrow \text{Parse Trees} \quad Z \leftarrow \text{Taggings} \]
\[ y(i, t) = 1 \text{ if } y \text{ contains tag } t \text{ at position } i \]
CKY Parsing

\[ y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t)) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t)) \]

Key

- \( f(y) \) ⇐ CFG
- \( g(z) \) ⇐ HMM
- \( \mathcal{Y} \) ⇐ Parse Trees
- \( \mathcal{Z} \) ⇐ Taggings
- \( y(i, t) = 1 \) if \( y \) contains tag \( t \) at position \( i \)

Penalties

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

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u(1, A) )</td>
<td>-1</td>
</tr>
<tr>
<td>( u(1, N) )</td>
<td>1</td>
</tr>
<tr>
<td>( u(2, N) )</td>
<td>-1</td>
</tr>
<tr>
<td>( u(2, V) )</td>
<td>1</td>
</tr>
<tr>
<td>( u(5, V) )</td>
<td>-1</td>
</tr>
<tr>
<td>( u(5, N) )</td>
<td>1</td>
</tr>
</tbody>
</table>
CKY Parsing

\[ y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t)) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t)) \]

Key

\[ f(y) \quad \leftarrow \quad \text{CFG} \quad \quad g(z) \quad \leftarrow \quad \text{HMM} \]
\[ \mathcal{Y} \quad \leftarrow \quad \text{Parse Trees} \quad \quad \mathcal{Z} \quad \leftarrow \quad \text{Taggings} \]
\[ y(i, t) = 1 \quad \text{if} \quad y \text{ contains tag } t \text{ at position } i \]

Penalties

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

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th></th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ u(1, A) ]</td>
<td>-1</td>
<td>[ u(5, V) ]</td>
</tr>
<tr>
<td>[ u(1, N) ]</td>
<td>1</td>
<td>[ u(5, N) ]</td>
</tr>
<tr>
<td>[ u(2, N) ]</td>
<td>-1</td>
<td>[ u(5, V) ]</td>
</tr>
<tr>
<td>[ u(2, V) ]</td>
<td>1</td>
<td>[ u(5, N) ]</td>
</tr>
</tbody>
</table>
CKY Parsing

\[ y^* = \arg \max_{y \in \mathcal{Y}} \left( f(y) + \sum_{i,t} u(i, t)y(i, t) \right) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} \left( g(z) - \sum_{i,t} u(i, t)z(i, t) \right) \]

Key

\[
\begin{align*}
f(y) & \iff \text{CFG} \\
\mathcal{Y} & \iff \text{Parse Trees} \\
y(i, t) = 1 \text{ if } y \text{ contains tag } t \text{ at position } i \end{align*}
\]

Penalties

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

\[
\begin{array}{c|c}
\text{Iteration 1} & \\
\hline
u(1, A) & -1 \\
u(1, N) & 1 \\
u(2, N) & -1 \\
u(2, V) & 1 \\
u(5, V) & -1 \\
u(5, N) & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{Iteration 2} & \\
\hline
u(5, V) & -1 \\
u(5, N) & 1 \\
\end{array}
\]
CKY Parsing

\[ y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,t} u(i, t)y(i, t)) \]

Viterbi Decoding

\[ z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,t} u(i, t)z(i, t)) \]

Penalties

\[
\begin{align*}
  u(i, t) = 0 & \text{ for all } i, t \\
  \text{Iteration 1} & \\
  u(1, A) & -1 \\
  u(1, N) & 1 \\
  u(2, N) & -1 \\
  u(2, V) & 1 \\
  u(5, V) & -1 \\
  u(5, N) & 1 \\
  \text{Iteration 2} & \\
  u(5, V) & -1 \\
  u(5, N) & 1 \\
\end{align*}
\]

Converged

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

Key

\[
\begin{align*}
  f(y) & \Leftarrow \text{CFG} \\
  \mathcal{Y} & \Leftarrow \text{Parse Trees} \\
  g(z) & \Leftarrow \text{HMM} \\
  \mathcal{Z} & \Leftarrow \text{Taggings} \\
  y(i, t) = 1 & \text{ if } y \text{ contains tag } t \text{ at position } i
\end{align*}
\]
Main theorem

**theorem:** if at any iteration, for all \( i, t \in \mathcal{T} \)

\[
y^{(k)}(i, t) = z^{(k)}(i, t)
\]

then \((y^{(k)}, z^{(k)})\) is the global optimum

**proof:** focus of the next section
2. Formal properties

**aim:** formal derivation of the algorithm given in the previous section

- derive Lagrangian dual
- prove three properties
  - upper bound
  - convergence
  - optimality
- describe subgradient method
Lagrangian

goal:

\[ \text{arg} \max_{y \in \mathcal{Y}, z \in \mathcal{Z}} f(y) + g(z) \text{ such that } y(i, t) = z(i, t) \]

Lagrangian:

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

redistribute terms

\[ L(u, y, z) = \left( f(y) + \sum_{i, t} u(i, t) y(i, t) \right) + \left( g(z) - \sum_{i, t} u(i, t) z(i, t) \right) \]
Lagrangian dual

Lagrangian:

\[ L(u, y, z) = \left( f(y) + \sum_{i,t} u(i, t)y(i, t) \right) + \left( g(z) - \sum_{i,t} u(i, t)z(i, t) \right) \]

Lagrangian dual:

\[ L(u) = \max_{y \in \mathcal{Y}, z \in \mathcal{Z}} L(u, y, z) \]

\[ = \max_{y \in \mathcal{Y}} \left( f(y) + \sum_{i,t} u(i, t)y(i, t) \right) + \max_{z \in \mathcal{Z}} \left( g(z) - \sum_{i,t} u(i, t)z(i, t) \right) \]
Theorem 1. Upper bound

**define:**

- $y^*, z^*$ is the optimal combined parsing and tagging solution with $y^*(i, t) = z^*(i, t)$ for all $i, t$

**theorem:** for any value of $u$

$$L(u) \geq f(y^*) + g(z^*)$$

$L(u)$ provides an upper bound on the score of the optimal solution

**note:** upper bound may be useful as input to branch and bound or A* search
Theorem 1. Upper bound (proof)

**Theorem:** for any value of $u$, $L(u) \geq f(y^*) + g(z^*)$

**Proof:**

\[
L(u) = \max_{y \in \mathcal{Y}, z \in \mathcal{Z}} L(u, y, z) \quad (1)
\]  
\[
\geq \max_{y \in \mathcal{Y}, z \in \mathcal{Z}: y = z} L(u, y, z) \quad (2)
\]  
\[
= \max_{y \in \mathcal{Y}, z \in \mathcal{Z}: y = z} f(y) + g(z) \quad (3)
\]  
\[
= f(y^*) + g(z^*) \quad (4)
\]
Formal algorithm (reminder)

Set $u^{(1)}(i, t) = 0$ for all $i, t \in T$

For $k = 1$ to $K$

$$y^{(k)} \leftarrow \arg \max_{y \in \mathcal{Y}} f(y) + \sum_{i,t} u^{(k)}(i, t)y(i, t) \text{ [Parsing]}$$

$$z^{(k)} \leftarrow \arg \max_{z \in \mathcal{Z}} g(z) - \sum_{i,t} u^{(k)}(i, t)z(i, t) \text{ [Tagging]}$$

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

Else $u^{(k+1)}(i, t) \leftarrow u^{(k)}(i, t) - \alpha_k (y^{(k)}(i, t) - z^{(k)}(i, t))$
Theorem 2. Convergence

notation:
- $u^{(k+1)}(i, t) \leftarrow u^{(k)}(i, t) + \alpha_k (y^{(k)}(i, t) - z^{(k)}(i, t))$ is update
- $u^{(k)}$ is the penalty vector at iteration $k$
- $\alpha_k$ is the update rate at iteration $k$

theorem: for any sequence $\alpha_1, \alpha_2, \alpha_3, \ldots$ such that

$$\lim_{t \to \infty} \alpha^t = 0 \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha^t = \infty,$$

we have

$$\lim_{t \to \infty} L(u^t) = \min_u L(u)$$

i.e. the algorithm converges to the tightest possible upper bound

proof: by subgradient convergence (next section)
Dual solutions

define:

- for any value of $u$

\[ y_u = \arg \max_{y \in \mathcal{Y}} \left( f(y) + \sum_{i, t} u(i, t)y(i, t) \right) \]

and

\[ z_u = \arg \max_{z \in \mathcal{Z}} \left( g(z) - \sum_{i, t} u(i, t)z(i, t) \right) \]

- $y_u$ and $z_u$ are the dual solutions for a given $u$
Theorem 3. Optimality

**Theorem:** if there exists $u$ such that

$$y_u(i, t) = z_u(i, t)$$

for all $i, t$ then

$$f(y_u) + g(z_u) = f(y^*) + g(z^*)$$

i.e. if the dual solutions agree, we have an optimal solution

$$(y_u, z_u)$$
Theorem 3. Optimality (proof)

**Theorem**: if $u$ such that $y_u(i, t) = z_u(i, t)$ for all $i, t$ then

$$f(y_u) + g(z_u) = f(y^*) + g(z^*)$$

**Proof**: by the definitions of $y_u$ and $z_u$

$$L(u) = f(y_u) + g(z_u) + \sum_{i,t} u(i, t)(y_u(i, t) - z_u(i, t))$$

$$= f(y_u) + g(z_u)$$

since $L(u) \geq f(y^*) + g(z^*)$ for all values of $u$

$$f(y_u) + g(z_u) \geq f(y^*) + g(z^*)$$

but $y^*$ and $z^*$ are optimal

$$f(y_u) + g(z_u) \leq f(y^*) + g(z^*)$$
Dual optimization

Lagrangian dual:

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

goal: dual problem is to find the tightest upper bound

\[
\min_u L(u)
\]
Dual subgradient

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

properties:
- \( L(u) \) is convex in \( u \) (no local minima)
- \( L(u) \) is not differentiable (because of max operator)

handle non-differentiability by using subgradient descent

define: 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 algorithm

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

recall, \( y_u \) and \( z_u \) are the argmax’s of the two terms

**subgradient:**

\[ g_u(i, t) = y_u(i, t) - z_u(i, t) \]

**subgradient descent:** move along the subgradient

\[ u'(i, t) = u(i, t) - \alpha (y_u(i, t) - z_u(i, t)) \]

guaranteed to find a minimum with conditions given earlier for \( \alpha \)
3. More examples

**aim:** demonstrate similar algorithms that can be applied to other decoding applications

- context-free parsing combined with dependency parsing
- corpus-level part-of-speech tagging
- combined translation alignment
Combined constituency and dependency parsing

(Rush et al., 2010)

**setup:** assume separate models trained for constituency and dependency parsing

**problem:** find constituency parse that maximizes the sum of the two models

**example:**
- combine lexicalized CFG with second-order dependency parser
Lexicalized constituency parsing

**notation:**
- \( \mathcal{Y} \) is set of lexicalized constituency parses for input
- \( y \in \mathcal{Y} \) is a valid parse
- \( f(y) \) scores a parse tree

**goal:**

\[
\arg \max_{y \in \mathcal{Y}} f(y)
\]

**example:** a lexicalized context-free grammar

![Diagram](https://via.placeholder.com/150)

- \( S(flies) \)
- \( NP(United) \)
- \( VP(flies) \)
- \( N \)
- \( V \)
- \( NP(jet) \)
- \( United \)
- \( flies \)
- \( D \)
- \( A \)
- \( N \)
- \( some \)
- \( large \)
- \( jet \)
Dependency parsing

define:

• $\mathcal{Z}$ is set of dependency parses for input
• $z \in \mathcal{Z}$ is a valid dependency parse
• $g(z)$ scores a dependency parse

example:

*0 United1 flies2 some3 large4 jet5
Identifying dependencies

notation: identify the dependencies selected by each model

- \( y(i, j) = 1 \) when word \( i \) modifies of word \( j \) in constituency parse \( y \)
- \( z(i, j) = 1 \) when word \( i \) modifies of word \( j \) in dependency parse \( z \)

example: a constituency and dependency parse with \( y(3, 5) = 1 \) and \( z(3, 5) = 1 \)
Combined optimization

**goal:**

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

such that for all \( i = 1 \ldots n, j = 0 \ldots n, \)

\[y(i, j) = z(i, j)\]
CKY Parsing

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

Dependency Parsing

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

Key

\[ f(y) \quad \Leftarrow \quad \text{CFG} \quad \quad \quad g(z) \quad \Leftarrow \quad \text{Dependency Model} \]
\[ \mathcal{Y} \quad \Leftarrow \quad \text{Parse Trees} \quad \quad \quad \mathcal{Z} \quad \Leftarrow \quad \text{Dependency Trees} \]
\[ y(i,j) = 1 \quad \text{if} \quad y \text{ contains dependency } i,j \]

Penalties

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

Penalties

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

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

Dependency Parsing

*0  United1 flies2 some3 large4 jet5

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

Key

\[ f(y) \leftarrow \text{CFG} \]
\[ g(z) \leftarrow \text{Dependency Model} \]
\[ Y \leftarrow \text{Parse Trees} \]
\[ Z \leftarrow \text{Dependency Trees} \]
\[ y(i, j) = 1 \text{ if } y \text{ contains dependency } i, j \]
CKY Parsing

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

Dependency Parsing

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

Key

\[
\begin{align*}
f(y) & \iff \text{CFG} \\
\mathcal{Y} & \iff \text{Parse Trees} \\
y(i,j) = 1 & \iff y \text{ contains dependency } i,j \\
g(z) & \iff \text{Dependency Model} \\
\mathcal{Z} & \iff \text{Dependency Trees}
\end{align*}
\]

Penalties

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

\[
y^* = \operatorname{arg\,max}_{y \in \mathcal{Y}} (f(y) + \sum_{i,j} u(i,j)y(i,j))
\]

Dependency Parsing

\[
z^* = \operatorname{arg\,max}_{z \in \mathcal{Z}} (g(z) - \sum_{i,j} u(i,j)z(i,j))
\]

Key

- \( f(y) \leftarrow \text{CFG} \)
- \( g(z) \leftarrow \text{Dependency Model} \)
- \( \mathcal{Y} \leftarrow \text{Parse Trees} \)
- \( \mathcal{Z} \leftarrow \text{Dependency Trees} \)
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)

Penalties

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

**Dependency Parsing**

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

**Key**

- \( f(y) \) \( \leftarrow \) CFG
- \( g(z) \) \( \leftarrow \) Dependency Model
- \( \mathcal{Y} \) \( \leftarrow \) Parse Trees
- \( \mathcal{Z} \) \( \leftarrow \) Dependency Trees
- \( y(i,j) = 1 \) if \( y \) contains dependency \( i,j \)

**Penalties**

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

**Iteration 1**

\[ u(2,3) = -1 \]
\[ u(5,3) = 1 \]
CKY Parsing

$$y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,j} u(i, j)y(i, j))$$

Dependency Parsing

$$z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,j} u(i, j)z(i, j))$$

Key

- $f(y) \Leftarrow \text{CFG}$
- $g(z) \Leftarrow \text{Dependency Model}$
- $\mathcal{Y} \Leftarrow \text{Parse Trees}$
- $\mathcal{Z} \Leftarrow \text{Dependency Trees}$
- $y(i, j) = 1$ if $y$ contains dependency $i, j$

Penalties

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>$u(2, 3)$</th>
<th>$u(5, 3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

Penalties

$$u(i, j) = 0 \text{ for all } i, j$$
**CKY Parsing**

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

**Dependency Parsing**

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

**Key**

\[
\begin{align*}
f(y) & \iff \text{CFG} \\
Y & \iff \text{Parse Trees} \\
y(i,j) & = 1 \text{ if } y \text{ contains dependency } i,j \\
g(z) & \iff \text{Dependency Model} \\
Z & \iff \text{Dependency Trees}
\end{align*}
\]

**Penalties**

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

\[
\begin{array}{c|c}
\text{Iteration 1} & \\
\hline
u(2,3) & -1 \\
u(5,3) & 1
\end{array}
\]
**CKY Parsing**

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

**Dependency Parsing**

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

**Key**

\[
f(y) \iff \text{CFG} \\
\mathcal{Y} \iff \text{Parse Trees} \\
y(i,j) = 1 \text{ if } y \text{ contains dependency } i,j
\]

**Penalties**

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

<table>
<thead>
<tr>
<th>Iteration 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u(2,3))</td>
</tr>
<tr>
<td>(u(5,3))</td>
</tr>
</tbody>
</table>
CKY Parsing

$y^* = \operatorname{arg\ max}_{y \in \mathcal{Y}} (f(y) + \sum_{i,j} u(i,j)y(i,j))$

Dependency Parsing

$z^* = \operatorname{arg\ max}_{z \in \mathcal{Z}} (g(z) - \sum_{i,j} u(i,j)z(i,j))$

Key

$f(y) \iff \text{CFG} \quad g(z) \iff \text{Dependency Model}$

$\mathcal{Y} \iff \text{Parse Trees} \quad \mathcal{Z} \iff \text{Dependency Trees}$

$y(i,j) = 1$ if $y$ contains dependency $i,j$

Penalties

$u(i,j) = 0$ for all $i,j$

Iteration 1

| $u(2,3)$ | -1 |
| $u(5,3)$ | 1 |

Converged

$y^* = \operatorname{arg\ max}_{y \in \mathcal{Y}} f(y) + g(y)$
Integrated Constituency and Dependency Parsing: Accuracy

F1 Score
- Collins (1997) Model 1
- Fixed, First-best Dependencies from Koo (2008)
- Dual Decomposition
**Corpus-level tagging**

**setup:** given a corpus of sentences and a trained sentence-level tagging model

**problem:** find best tagging for each sentence, while at the same time enforcing inter-sentence soft constraints

**example:**
- test-time decoding with a trigram tagger
- constraint that each word type prefer a single POS tag
Corpus-level tagging

English is my first language

He studies language arts now

Language makes us human beings
Sentence-level decoding

notation:
• $\mathcal{Y}_i$ is set of tag sequences for input sentence $i$
• $\mathcal{Y} = \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_m$ is set of tag sequences for the input corpus
• $Y \in \mathcal{Y}$ is a valid tag sequence for the corpus
• $F(Y) = \sum_i f(Y_i)$ is the score for tagging the whole corpus

goal:
$$\arg\max_{Y \in \mathcal{Y}} F(Y)$$

eexample: decode each sentence with a trigram tagger

English is my first language
He studies language arts now
Inter-sentence constraints

notation:

- $\mathcal{Z}$ is set of possible assignments of tags to word types
- $z \in \mathcal{Z}$ is a valid tag assignment
- $g(z)$ is a scoring function for assignments to word types

example: an MRF model that encourages words of the same type to choose the same tag

\[ g(z_1) > g(z_2) \]
Identifying word tags

**notation:** identify the tag labels selected by each model

- $Y_s(i, t) = 1$ when the tagger for sentence $s$ at position $i$ selects tag $t$
- $z(s, i, t) = 1$ when the constraint assigns at sentence $s$ position $i$ the tag $t$

**example:** a parse and tagging with $Y_1(5, N) = 1$ and $z(1, 5, N) = 1$

```
English is my first language
He studies language arts now
```

\[ V \]

\[ Z \]
Combined optimization

goal:

$$\arg \max_{Y \in \mathcal{Y}, z \in \mathcal{Z}} F(Y) + g(z)$$

such that for all $s = 1 \ldots m$, $i = 1 \ldots n$, $t \in \mathcal{T}$,

$$Y_s(i, t) = z(s, i, t)$$
Tagging

MRF

Key

$F(Y) \iff \text{Tagging model}$

$Y \iff \text{Sentence-level tagging}$

$Y_s(i, t) = 1 \iff \text{sentence } s \text{ has tag } t \text{ at position } i$

$g(z) \iff \text{MRF}$

$Z \iff \text{Inter-sentence constraints}$

Penalties

$u(s, i, t) = 0 \text{ for all } s, i, t$
Tagging

English is my first language
He studies language arts now
Language makes us human beings

Penalties

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

MRF

Key

\[
\begin{align*}
F(Y) & \iff \text{Tagging model} \\
\mathcal{Y} & \iff \text{Sentence-level tagging} \\
Y_s(i, t) = 1 & \text{ if sentence } s \text{ has tag } t \text{ at position } i
\end{align*}
\]

\[ g(z) \iff \text{MRF} \]

\[ Z \iff \text{Inter-sentence constraints} \]
Tagging

\[
\begin{align*}
N & \quad V & \quad P & \quad A & \quad N \\
\text{English} & \quad \text{is} & \quad \text{my} & \quad \text{first} & \quad \text{language} \\
P & \quad V & \quad A & \quad N & \quad R \\
\text{He} & \quad \text{studies} & \quad \text{language} & \quad \text{arts} & \quad \text{now} \\
N & \quad V & \quad P & \quad N & \quad N \\
\text{Language} & \quad \text{makes} & \quad \text{us} & \quad \text{human} & \quad \text{beings}
\end{align*}
\]

MRF

\[
\begin{align*}
A & \quad A & \quad A & \quad A \\
\text{language} & \quad \text{language} & \quad \text{language}
\end{align*}
\]

Key

\[
\begin{align*}
F(Y) & \quad \Leftarrow \quad \text{Tagging model} \\
g(z) & \quad \Leftarrow \quad \text{MRF} \\
\mathcal{Y} & \quad \Leftarrow \quad \text{Sentence-level tagging} \\
\mathcal{Z} & \quad \Leftarrow \quad \text{Inter-sentence constraints} \\
Y_s(i, t) = 1 & \quad \text{if} \quad \text{sentence} \ s \ \text{has tag} \ t \ \text{at position} \ i
\end{align*}
\]
English is my first language.

He studies language arts now.

Language makes us human beings.

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

\[
\begin{align*}
F(Y) & \iff \text{Tagging model} \\
\mathcal{Y} & \iff \text{Sentence-level tagging} \\
Y_s(i, t) = 1 & \iff \text{sentence } s \text{ has tag } t \text{ at position } i
\end{align*}
\]
Tagging

English is my first language
He studies language arts now
Language makes us human beings

Penalties
\[ u(s, i, t) = 0 \text{ for all } s, i, t \]

Iteration 1

\[
\begin{align*}
&u(1, 5, N) &-1 \\
u(1, 5, A) &1 \\
&u(3, 1, N) &-1 \\
u(3, 1, A) &1
\end{align*}
\]

MRF

Key

\[
\begin{align*}
F(Y) &\iff \text{Tagging model} \\
\mathcal{Y} &\iff \text{Sentence-level tagging} \\
Y_s(i, t) = 1 &\text{ if sentence } s \text{ has tag } t \text{ at position } i \\
g(z) &\iff \text{MRF} \\
\mathcal{Z} &\iff \text{Inter-sentence constraints}
\end{align*}
\]
Tagging

Penalties

\[ u(s, i, t) = 0 \] for all \( s, i, t \)

Iteration 1

\[
\begin{align*}
  u(1, 5, N) & = -1 \\
  u(1, 5, A) & = 1 \\
  u(3, 1, N) & = -1 \\
  u(3, 1, A) & = 1 
\end{align*}
\]

MRF

Key

\[ F(Y) \quad \Leftrightarrow \quad \text{Tagging model} \]

\[ g(z) \quad \Leftrightarrow \quad \text{MRF} \]

\[ Y_s(i, t) = 1 \quad \text{if} \quad \text{sentence } s \text{ has tag } t \text{ at position } i \]

\[ Z \quad \Leftrightarrow \quad \text{Inter-sentence constraints} \]
Tagging

English is my first language

He studies language arts now

Language makes us human beings

Penalties

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

Iteration 1

\[ u(1, 5, N) = -1 \]
\[ u(1, 5, A) = 1 \]
\[ u(3, 1, N) = -1 \]
\[ u(3, 1, A) = 1 \]

MRF

Key

\[ F(Y) \iff \text{Tagging model} \]
\[ g(z) \iff \text{MRF} \]
\[ Y \iff \text{Sentence-level tagging} \]
\[ Z \iff \text{Inter-sentence constraints} \]
\[ Y_s(i, t) = 1 \text{ if sentence } s \text{ has tag } t \text{ at position } i \]
Tagging

<table>
<thead>
<tr>
<th>N</th>
<th>V</th>
<th>P</th>
<th>A</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>is</td>
<td>my</td>
<td>first</td>
<td>language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th>V</th>
<th>A</th>
<th>N</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>He</td>
<td>studies</td>
<td>language</td>
<td>arts</td>
<td>now</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>V</th>
<th>P</th>
<th>N</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>makes</td>
<td>us</td>
<td>human</td>
<td>beings</td>
</tr>
</tbody>
</table>

MRF

Key

$F(Y) \iff \text{Tagging model}$

$g(z) \iff \text{MRF}$

$\mathcal{Y} \iff \text{Sentence-level tagging}$

$\mathcal{Z} \iff \text{Inter-sentence constraints}$

$Y_s(i, t) = 1$ if sentence $s$ has tag $t$ at position $i$

Penalties

$u(s, i, t) = 0$ for all $s, i, t$

Iteration 1

$u(1, 5, N) = -1$

$u(1, 5, A) = 1$

$u(3, 1, N) = -1$

$u(3, 1, A) = 1$
**Tagging**

English is my first language

He studies language arts now

Language makes us human beings

**MRF**

**Penalties**

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

Iteration 1

\[
\begin{align*}
&u(1, 5, N) = -1 \\
&u(1, 5, A) = 1 \\
&u(3, 1, N) = -1 \\
&u(3, 1, A) = 1
\end{align*}
\]

**Key**

\[ F(Y) \iff \text{Tagging model} \]

\[ Y \iff \text{Sentence-level tagging} \]

\[ Y_s(i, t) = 1 \text{ if sentence } s \text{ has tag } t \text{ at position } i \]
Tagging

English is my first language
He studies language arts now
Language makes us human beings

MRF

Penalties

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

Iteration 1

\[
\begin{align*}
&u(1, 5, N) = -1 \\
&u(1, 5, A) = 1 \\
&u(3, 1, N) = -1 \\
&u(3, 1, A) = 1
\end{align*}
\]

Iteration 2

\[
\begin{align*}
&u(1, 5, N) = -1 \\
&u(1, 5, A) = 1 \\
&u(3, 1, N) = -1 \\
&u(3, 1, A) = 1 \\
&u(2, 3, N) = 1 \\
&u(2, 3, A) = -1
\end{align*}
\]

Key

\[ F(Y) \iff \text{Tagging model} \]
\[ g(z) \iff \text{MRF} \]
\[ Y \iff \text{Sentence-level tagging} \]
\[ Z \iff \text{Inter-sentence constraints} \]
\[ Y_s(i, t) = 1 \text{ if sentence } s \text{ has tag } t \text{ at position } i \]
Tagging

English is my first language

He studies language arts now

Language makes us human beings

MRF

Penalties

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

Iteration 1

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>V</td>
<td>P</td>
<td>A</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>is</td>
<td>my</td>
<td>first</td>
<td>language</td>
</tr>
<tr>
<td>P</td>
<td>V</td>
<td>N</td>
<td>N</td>
<td>R</td>
</tr>
<tr>
<td>He</td>
<td>studies</td>
<td>language</td>
<td>arts</td>
<td>now</td>
</tr>
<tr>
<td>N</td>
<td>V</td>
<td>P</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Language</td>
<td>makes</td>
<td>us</td>
<td>human</td>
<td>beings</td>
</tr>
</tbody>
</table>

Iteration 2

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>V</td>
<td>P</td>
<td>A</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>makes</td>
<td>us</td>
<td>human</td>
<td>beings</td>
</tr>
</tbody>
</table>

Key

\[ F(Y) \iff \text{Tagging model} \]
\[ Y \iff \text{Sentence-level tagging} \]
\[ Y_s(i, t) = 1 \text{ if sentence } s \text{ has tag } t \text{ at position } i \]
**Tagging**

\[ N \quad V \quad P \quad A \quad N \]

English is my first language

\[ P \quad V \quad N \quad N \quad R \]

He studies language arts now

\[ N \quad V \quad P \quad N \quad N \]

Language makes us human beings

---

**Penalties**

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

**Iteration 1**

\[ u(1, 5, N) = -1 \]
\[ u(1, 5, A) = 1 \]
\[ u(3, 1, N) = -1 \]
\[ u(3, 1, A) = 1 \]

**Iteration 2**

\[ u(1, 5, N) = -1 \]
\[ u(1, 5, A) = 1 \]
\[ u(3, 1, N) = -1 \]
\[ u(3, 1, A) = 1 \]
\[ u(2, 3, N) = 1 \]
\[ u(2, 3, A) = -1 \]

---

**MRF**

![MRF Diagram]

---

**Key**

\[ F(Y) \iff \text{Tagging model} \]
\[ g(z) \iff \text{MRF} \]
\[ Y \iff \text{Sentence-level tagging} \]
\[ Z \iff \text{Inter-sentence constraints} \]

\[ Y_s(i, t) = 1 \text{ if sentence } s \text{ has tag } t \text{ at position } i \]
**Combined alignment** (DeNero and Macherey, 2011)

**setup:** assume separate models trained for English-to-French and French-to-English alignment

**problem:** find an alignment that maximizes the score of both models

**example:**
- HMM models for both directional alignments (assume correct alignment is one-to-one for simplicity)
English-to-French alignment

define:

• $\mathcal{Y}$ is set of all possible English-to-French alignments
• $y \in \mathcal{Y}$ is a valid alignment
• $f(y)$ scores of the alignment

example: HMM alignment

\[
\begin{align*}
\text{Le}_1 \quad \rightarrow & \quad \text{laid}_3 \quad \rightarrow \quad \text{chien}_2 \quad \rightarrow \quad \text{a}_4 \quad \rightarrow \quad \text{rouge}_6 \quad \rightarrow \quad \text{fourrure}_5 \\
\text{The}_1 \quad \rightarrow & \quad \text{ugly}_2 \quad \rightarrow \quad \text{dog}_3 \quad \rightarrow \quad \text{has}_4 \quad \rightarrow \quad \text{red}_5 \quad \rightarrow \quad \text{fur}_6
\end{align*}
\]
French-to-English alignment

**define:**
- $\mathcal{Z}$ is set of all possible French-to-English alignments
- $z \in \mathcal{Z}$ is a valid alignment
- $g(z)$ scores of an alignment

**example:** HMM alignment

Le$_1$ chien$_2$ laid$_3$ a$_4$ fourrure$_5$ rouge$_6$

The$_1$ ugly$_2$ dog$_3$ has$_4$ fur$_6$ red$_5$

![Diagram of HMM alignment]

- le
- chien
- laid
- a
- fourrure
- rouge
Identifying word alignments

**notation:** identify the tag labels selected by each model

- $y(i,j) = 1$ when e-to-f alignment $y$ selects French word $i$ to align with English word $j$
- $z(i,j) = 1$ when f-to-e alignment $z$ selects French word $i$ to align with English word $j$

**example:** two HMM alignment models with $y(6,5) = 1$ and $z(6,5) = 1$
Combined optimization

goal:

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

such that for all \( i = 1 \ldots n, j = 1 \ldots n, \)

\[
y(i, j) = z(i, j)
\]
English-to-French

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

French-to-English

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

Key

\begin{align*}
  f(y) & \iff \text{HMM Alignment} \\
  \mathcal{Y} & \iff \text{English-to-French model} \\
  y(i,j) = 1 & \text{ if French word } i \text{ aligns to English word } j \\
  g(z) & \iff \text{HMM Alignment} \\
  \mathcal{Z} & \iff \text{French-to-English model} \\
\end{align*}
**English-to-French**

\[
y^* = \operatorname{arg\,max}_{y \in Y} (f(y) + \sum_{i,j} u(i,j) y(i,j))
\]

**French-to-English**

\[
z^* = \operatorname{arg\,max}_{z \in Z} (g(z) - \sum_{i,j} u(i,j) z(i,j))
\]

**Key**

\[
\begin{align*}
f(y) & \iff \text{HMM Alignment} \\
y & \iff \text{English-to-French model} \\
y(i,j) = 1 & \text{ if French word } i \text{ aligns to English word } j
\end{align*}
\]

**Penalties**

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

$$y^* = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,j} u(i, j)y(i, j))$$

French-to-English

$$z^* = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,j} u(i, j)z(i, j))$$

Key

- $f(y) \iff$ HMM Alignment
- $\mathcal{Y} \iff$ English-to-French model
- $g(z) \iff$ HMM Alignment
- $\mathcal{Z} \iff$ French-to-English model
- $y(i, j) = 1$ if French word $i$ aligns to English word $j$

Penalties

$$u(i, j) = 0 \text{ for all } i, j$$
English-to-French

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

French-to-English

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

Key

- \(f(y) \leftarrow \text{HMM Alignment}\)
- \(\mathcal{Y} \leftarrow \text{English-to-French model}\)
- \(\mathcal{Z} \leftarrow \text{French-to-English model}\)
- \(y(i,j) = 1 \quad \text{if} \quad \text{French word } i \text{ aligns to English word } j\)

Penalties

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

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

French-to-English

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

Key

- \( f(y) \) \( \iff \) HMM Alignment
- \( \mathcal{Y} \) \( \iff \) English-to-French model
- \( y(i,j) = 1 \) if French word \( i \) aligns to English word \( j \)
- \( g(z) \) \( \iff \) HMM Alignment
- \( \mathcal{Z} \) \( \iff \) French-to-English model

Penalties

- \( u(i,j) = 0 \) for all \( i,j \)

<table>
<thead>
<tr>
<th>Iteration 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u(3,2) )</td>
</tr>
<tr>
<td>( u(2,2) )</td>
</tr>
<tr>
<td>( u(2,3) )</td>
</tr>
<tr>
<td>( u(3,3) )</td>
</tr>
</tbody>
</table>
English-to-French

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

French-to-English

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

Key

\[ f(y) \iff \text{HMM Alignment} \quad g(z) \iff \text{HMM Alignment} \]

\[ Y \iff \text{English-to-French model} \quad Z \iff \text{French-to-English model} \]

\[ y(i,j) = 1 \quad \text{if} \quad \text{French word } i \text{ aligns to English word } j \]

Penalties

\[
\begin{array}{c|c}
\text{Iteration 1} & \\
\hline
u(3, 2) & -1 \\
u(2, 2) & 1 \\
u(2, 3) & -1 \\
u(3, 3) & 1 \\
\end{array}
\]

\[ u(i,j) = 0 \text{ for all } i,j \]
**English-to-French**

\[
y^\ast = \arg \max_{y \in \mathcal{Y}} (f(y) + \sum_{i,j} u(i, j)y(i, j))
\]

**French-to-English**

\[
z^\ast = \arg \max_{z \in \mathcal{Z}} (g(z) - \sum_{i,j} u(i, j)z(i, j))
\]

**Key**

- \( f(y) \) $\iff$ HMM Alignment
- \( g(z) \) $\iff$ HMM Alignment
- \( \mathcal{Y} \) $\iff$ English-to-French model
- \( \mathcal{Z} \) $\iff$ French-to-English model
- \( y(i, j) = 1 \) if French word \( i \) aligns to English word \( j \)

**Penalties**

\[
\begin{align*}
u(i, j) &= 0 \text{ for all } i, j \\
\text{Iteration 1} \\
u(3, 2) &= -1 \\
u(2, 2) &= 1 \\
u(2, 3) &= -1 \\
u(3, 3) &= 1
\end{align*}
\]
English-to-French

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

French-to-English

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

Key

- \( f(y) \) \( \iff \) HMM Alignment
- \( \mathcal{Y} \) \( \iff \) English-to-French model
- \( y(i,j) = 1 \) if French word \( i \) aligns to English word \( j \)
- \( g(z) \) \( \iff \) HMM Alignment
- \( \mathcal{Z} \) \( \iff \) French-to-English model

Penalties

\[
\begin{array}{c|c}
  u(i,j) = 0 & \text{for all } i,j \\
  u(3,2) & -1 \\
  u(2,2) & 1 \\
  u(2,3) & -1 \\
  u(3,3) & 1 \\
\end{array}
\]
English-to-French

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

French-to-English

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

Key

\[ f(y) \iff \text{HMM Alignment} \]
\[ \mathcal{Y} \iff \text{English-to-French model} \]
\[ y(i, j) = 1 \text{ if French word } i \text{ aligns to English word } j \]

Penalties

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

Iteration 1

\[ \begin{array}{c|c}
\hline
 & \text{u(3, 2)} & \text{u(2, 2)} & \text{u(2, 3)} & \text{u(3, 3)} \\
\hline
\text{u(3, 2)} & -1 & & & \\
\text{u(2, 2)} & 1 & & & \\
\text{u(2, 3)} & & -1 & & \\
\text{u(3, 3)} & & & 1 & \\
\hline
\end{array} \]
English-to-French

\[ y^* = \arg \max_{y \in Y} \left( f(y) + \sum_{i,j} u(i, j)y(i, j) \right) \]

French-to-English

\[ z^* = \arg \max_{z \in Z} \left( g(z) - \sum_{i,j} u(i, j)z(i, j) \right) \]

Key

- \( f(y) \leftarrow \text{HMM Alignment} \)
- \( Y \leftarrow \text{English-to-French model} \)
- \( y(i,j) = 1 \) if French word \( i \) aligns to English word \( j \)
- \( g(z) \leftarrow \text{HMM Alignment} \)
- \( Z \leftarrow \text{French-to-English model} \)
4. Practical issues

**aim:** overview of practical dual decomposition techniques

- tracking the progress of the algorithm
- choice of update rate $\alpha_k$
- lazy update of dual solutions
- extracting solutions if algorithm does not converge
Optimization tracking

at each stage of the algorithm there are several useful values

track:

- $y^{(k)}$, $z^{(k)}$ are current dual solutions
- $L(u^{(k)})$ is the current dual value
- $y^{(k)}$, $l(y^{(k)})$ is a potential primal feasible solution
- $f(y^{(k)}) + g(l(y^{(k)}))$ is the potential primal value
example run from syntactic machine translation (later in talk)

- current primal
  \[ f(y^{(k)}) + g(l(y^{(k)})) \]

- current dual
  \[ L(u^{(k)}) \]
useful signals:

- $L(u^{(k)}) - L(u^{(k-1)})$ is the dual change (may be positive)
- $\min_k L(u^{(k)})$ is the best dual value (tightest upper bound)
- $\max_k f(y^{(k)}) + g(l(y^{(k)}))$ is the best primal value

the optimal value must be between the best dual and primal values
best primal

\[ \max_k f(y^{(k)}) + g(l(y^{(k)})) \]

best dual

\[ \min_k L(u^{(k)}) \]

gap

\[ \min_k L(u^k) - \max_k f(y^{(k)}) + g(l(y^{(k)})) \]
Update rate

choice of $\alpha_k$ has important practical consequences

- $\alpha_k$ too high causes dual value to fluctuate
- $\alpha_k$ too low means slow progress
practical: find a rate that is robust to varying inputs

- $\alpha_k = c$ (constant rate) can be very fast, but hard to find constant that works for all problems
- $\alpha_k = \frac{c}{k}$ (decreasing rate) often cuts rate too aggressively, lowers value even when making progress
- rate based on dual progress
  - $\alpha_k = \frac{c}{t + 1}$ where $t < k$ is number of iterations where dual value increased
  - robust in practice, reduces rate when dual value is fluctuating
Lazy decoding

**idea:** don’t recompute $y^{(k)}$ or $z^{(k)}$ from scratch each iteration

**lazy decoding:** if subgradient $u^{(k)}$ is sparse, then $y^{(k)}$ may be very easy to compute from $y^{(k-1)}$

**use:**
- helpful if $y$ or $z$ factor naturally into independent components
- can be important for fast decompositions
Lazy decoding example

recall corpus-level tagging example

at this iteration, only sentence 2 receives a weight update

with lazy decoding

\[ Y_1^{(k)} \leftarrow Y_1^{(k-1)} \]
\[ Y_3^{(k)} \leftarrow Y_3^{(k-1)} \]
Lazy decoding results

lazy decoding is critical for the efficiency of some applications

recomputation statistics for non-projective dependency parsing
Approximate solution

upon agreement the solution is exact, but this may not occur otherwise, there is an easy way to find an approximate solution

choose: the structure $y^{(k')}$ where

$$k' = \arg \max_k f(y^{(k)}) + g(l(y^{(k)}))$$

is the iteration with the best primal score

guarantee: the solution $y^{k'}$ is non-optimal by at most

$$\left( \min_k L(u^k) \right) - \left( f(y^{(k')}) + g(l(y^{(k')}) \right)$$

there are other methods to estimate solutions, for instance by averaging solutions (see Nedić and Ozdaglar (2009))
Choosing best solution

non-exact example from syntactic translation

best approximate primal solution occurs at iteration 63
Early stopping results

early stopping results for constituency and dependency parsing

![Graph showing early stopping results for constituency and dependency parsing. The graph plots the percentage of f score, % certificates, and % match K=50 against the maximum number of dual decomposition iterations.](image-url)
Early stopping results

early stopping results for non-projective dependency parsing
Tightening

instead of using approximate solution, can tighten the algorithm
may help find an exact solution at the cost of added complexity
this technique is the focus of the next section
5. Linear programming

**aim:** explore the connections between dual decomposition and linear programming

- basic optimization over the simplex
- formal properties of linear programming
- full example with fractional optimal solutions
- tightening linear program relaxations
Simplex

define:

- $\Delta_y \subset \mathcal{R}^{\mid \mathcal{Y} \mid}$ is the simplex over $\mathcal{Y}$ where $\alpha \in \Delta_y$ implies $\alpha_y \geq 0$ and $\sum_y \alpha_y = 1$

- $\alpha$ is distribution over $\mathcal{Y}$
- $\Delta_z$ is the simplex over $\mathcal{Z}$
- $\delta_y : \mathcal{Y} \rightarrow \Delta_y$ maps elements to the simplex

example:

$\mathcal{Y} = \{y_1, y_2, y_3\}$

vertices

- $\delta_y(y_1) = (1, 0, 0)$
- $\delta_y(y_2) = (0, 1, 0)$
- $\delta_y(y_3) = (0, 0, 1)$
Theorem 1. Simplex linear program
optimize over the simplex $\Delta_y$ instead of the discrete sets $\mathcal{Y}$

goal: optimize linear program

$$\max_{\alpha \in \Delta_y} \sum_y \alpha_y f(y)$$

theorem:

$$\max_{y \in \mathcal{Y}} f(y) = \max_{\alpha \in \Delta_y} \sum_y \alpha_y f(y)$$

proof: points in $\mathcal{Y}$ correspond to the extreme points of simplex

$$\{\delta_y(y) : y \in \mathcal{Y}\}$$

linear program has optimum at extreme point

note: finding the highest scoring distribution $\alpha$ over $\mathcal{Y}$

proof shows that best distribution chooses a single parse
Combined linear program

optimize over the simplices \( \Delta_y \) and \( \Delta_z \) instead of the discrete sets \( Y \) and \( Z \)

**goal:** optimize linear program

\[
\max_{\alpha \in \Delta_y, \beta \in \Delta_z} \sum_y \alpha_y f(y) + \sum_z \beta_z g(z)
\]

such that for all \( i, t \)

\[
\sum_y \alpha_y y(i, t) = \sum_z \beta_z z(i, t)
\]

**note:** the two distributions must match in expectation of POS tags
the best distributions \( \alpha^*, \beta^* \) are possibly no longer a single parse tree or tag sequence
Lagrangian

Lagrangian:

\[ M(u, \alpha, \beta) = \sum_y \alpha_y f(y) + \sum_z \beta_z g(z) + \sum_{i,t} u(i, t) \left( \sum_y \alpha_y y(i, t) - \sum_z \beta_z z(i, t) \right) \]

\[ = \left( \sum_y \alpha_y f(y) + \sum_{i,t} u(i, t) \sum_y \alpha_y y(i, t) \right) + \right. \left( \sum_z \beta_z g(z) - \sum_{i,t} u(i, t) \sum_z \beta_z z(i, t) \right) \]

Lagrangian dual:

\[ M(u) = \max_{\alpha \in \Delta_y, \beta \in \Delta_z} M(u, \alpha, \beta) \]
Theorem 2. Strong duality

define:
• \( \alpha^*, \beta^* \) is the optimal assignment to \( \alpha, \beta \) in the linear program

theorem:
\[
\min_u M(u) = \sum_y \alpha_y^* f(y) + \sum_z \beta_z^* g(z)
\]

proof: by linear programming duality
Theorem 3. Dual relationship

**Theorem:** for any value of $u$,

$$M(u) = L(u)$$

**Note:** solving the original Lagrangian dual also solves dual of the linear program
Theorem 3. Dual relationship (proof sketch)

focus on $\mathcal{Y}$ term in Lagrangian

$$L(u) = \max_{y \in \mathcal{Y}} \left( f(y) + \sum_{i,t} u(i, t) y(i, t) \right) + \ldots$$

$$M(u) = \max_{\alpha \in \Delta_y} \left( \sum_y \alpha_y f(y) + \sum_{i,t} u(i, t) \sum_y \alpha_y y(i, t) \right) + \ldots$$

by theorem 1. optimization over $\mathcal{Y}$ and $\Delta_y$ have the same max

similar argument for $\mathcal{Z}$ gives $L(u) = M(u)$
Summary

\[ f(y) + g(z) \quad \text{original primal objective} \]
\[ L(u) \quad \text{original dual} \]
\[ \sum_y \alpha_y f(y) + \sum_z \beta_z g(z) \quad \text{LP primal objective} \]
\[ M(u) \quad \text{LP dual} \]

relationship between LP dual, original dual, and LP primal objective

\[ \min_u M(u) = \min_u L(u) = \sum_y \alpha^*_y f(y) + \sum_z \beta^*_z g(z) \]
Primal relationship

**define:**

- $Q \subseteq \Delta_y \times \Delta_z$ corresponds to feasible solutions of the original problem
  \[
  Q = \{ (\delta_y(y), \delta_z(z)) : y \in \mathcal{Y}, z \in \mathcal{Z}, \]
  \[
  y(i, t) = z(i, t) \text{ for all } (i, t) \}\]

- $Q' \subseteq \Delta_y \times \Delta_z$ is the set of feasible solutions to the LP
  \[
  Q' = \{ (\alpha, \beta) : \alpha \in \Delta_y, \beta \in \Delta_z, \]
  \[
  \sum_y \alpha_y y(i, t) = \sum_z \beta_z z(i, t) \text{ for all } (i, t) \}\]

- $Q \subseteq Q'$

**solutions:**

$$\max_{q \in Q} h(q) \leq \max_{q \in Q'} h(q) \text{ for any } h$$
Concrete example

- \( \mathcal{Y} = \{y_1, y_2, y_3\} \)
- \( \mathcal{Z} = \{z_1, z_2, z_3\} \)
- \( \Delta_y \subset \mathbb{R}^3, \Delta_z \subset \mathbb{R}^3 \)
choose:
• $\alpha^{(1)} = (0, 0, 1) \in \Delta_y$ is representation of $y_3$
• $\beta^{(1)} = (0, 0, 1) \in \Delta_z$ is representation of $z_3$
confirm:
$$\sum_y \alpha^{(1)}_y y(i, t) = \sum_z \beta^{(1)}_z z(i, t)$$
$\alpha^{(1)}$ and $\beta^{(1)}$ satisfy agreement constraint
Fractional solution

\[ y_1 \quad y_2 \quad y_3 \]

\[
\begin{array}{c}
Y \\
\begin{array}{c}
\begin{array}{c}
 x \\
 a \\
 \text{He is}
\end{array} \\
\begin{array}{c}
 b \\
 \text{He is}
\end{array} \\
\begin{array}{c}
 c \\
 \text{He is}
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
Z \\
\begin{array}{c}
 a \rightarrow b \\
 \text{He is}
\end{array} \\
\begin{array}{c}
 b \rightarrow a \\
 \text{He is}
\end{array} \\
\begin{array}{c}
 c \rightarrow c \\
 \text{He is}
\end{array}
\end{array}
\]

choose:
- \( \alpha^{(2)} = (0.5, 0.5, 0) \in \Delta_y \) is combination of \( y_1 \) and \( y_2 \)
- \( \beta^{(2)} = (0.5, 0.5, 0) \in \Delta_z \) is combination of \( z_1 \) and \( z_2 \)

confirm:
\[
\sum_y \alpha_y^{(2)} y(i, t) = \sum_z \beta_z^{(2)} z(i, t)
\]

\( \alpha^{(2)} \) and \( \beta^{(2)} \) satisfy agreement constraint, but not integral
Optimal solution

weights:

- the choice of \( f \) and \( g \) determines the optimal solution
- if \((f, g)\) favors \((\alpha^{(2)}, \beta^{(2)})\), the optimal solution is fractional

example: \( f = [1 \ 1 \ 2] \) and \( g = [1 \ 1 \ -2] \)

- \( f \cdot \alpha^{(1)} + g \cdot \beta^{(1)} = 0 \) vs \( f \cdot \alpha^{(2)} + g \cdot \beta^{(2)} = 2 \)
- \( \alpha^{(2)}, \beta^{(2)} \) is optimal, even though it is fractional

summary: dual and LP primal optimal:

\[
\min_u M(u) = \min_u L(u) = \sum_y \alpha_y^{(2)} f(y) + \sum_z \beta_z^{(2)} g(z) = 2
\]

original primal optimal:

\[
f(y^*) + g(z^*) = 0
\]
round 1

dual solutions:

\[
\begin{align*}
y_3 & \quad \text{He} \\
 x & \quad \text{c} \\
 c & \quad \text{c} \\
\text{He} & \quad \text{is}
\end{align*}
\]

dual values:

\[
\begin{align*}
y^{(1)} & \quad 2.00 \\
z^{(1)} & \quad 1.00 \\
L(u^{(1)}) & \quad 3.00
\end{align*}
\]
round 2

dual solutions:

\[
\begin{align*}
&x & y_2 \quad & z_1 \quad & b \quad & b \\
& & & & & \\
& & & & He & is
\end{align*}
\]

dual values:

\[
\begin{align*}
y^{(2)} & = 2.00 \\
z^{(2)} & = 1.00 \\
L(u^{(2)}) & = 3.00
\end{align*}
\]

previous solutions:

\[
\begin{align*}
y_3 & \quad z_2 \\
y_2 & \quad z_1 \\
y_1 & \quad z_1 \\
y_2 & \quad z_2 \\
y_1 & \quad z_1
\end{align*}
\]
round 3

**dual solutions:**

\[ y_1 \]

\[ x \]

\[ a \quad a \]

\[ \text{He} \quad \text{is} \]

\[ z_1 \]

\[ a \rightarrow b \]

\[ y^{(3)} = 2.50 \]

\[ z^{(3)} = 0.50 \]

\[ L(u^{(3)}) = 3.00 \]

**previous solutions:**

\[ y_3 \quad z_2 \]

\[ y_2 \quad z_1 \]

\[ y_1 \quad z_1 \]
round 4

dual solutions:

\[ y_1 \]

\[ x \]

\[ a \rightarrow b \]

\[ z_1 \]

He is

\[ y_2 \]

\[ y_1 \]

\[ z_2 \]

\[ z_1 \]

Round

dual values:

\[ y^{(4)} = 2.17 \]

\[ z^{(4)} = 0.17 \]

\[ L(u^{(4)}) = 2.33 \]

previous solutions:

\[ y_3 \quad z_2 \]

\[ y_2 \quad z_1 \]

\[ y_1 \quad z_1 \]

\[ y_1 \quad z_1 \]
round 5

dual solutions:

\[ y_2 \]

\[ x \]

\[ b \quad b \]

\[ \text{He} \quad \text{is} \]

\[ z_2 \]

\[ b \rightarrow a \]

\[ \text{He} \quad \text{is} \]

dual values:

\[ y^{(5)} \quad 2.08 \]

\[ z^{(5)} \quad 0.08 \]

\[ L(u^{(5)}) \quad 2.17 \]

previous solutions:

\[ y_3 \quad z_2 \]

\[ y_2 \quad z_1 \]

\[ y_1 \quad z_1 \]

\[ y_1 \quad z_1 \]

\[ y_2 \quad z_2 \]
round 6

dual solutions:

\[ y_1 \quad \text{z}_1 \]
\[ x \quad \text{a} \rightarrow \text{b} \]
\[ \text{He} \quad \text{is} \]

previous solutions:

\[ y_3 \quad \text{z}_2 \]
\[ y_2 \quad \text{z}_1 \]
\[ y_1 \quad \text{z}_1 \]
\[ y_1 \quad \text{z}_1 \]
\[ y_2 \quad \text{z}_2 \]
\[ y_1 \quad \text{z}_1 \]

Round

dual values:

\[ y^{(6)} \quad 2.12 \]
\[ z^{(6)} \quad 0.12 \]
\[ L(u^{(6)}) \quad 2.23 \]
round 7

dual solutions:

\begin{align*}
  y_2 & \\
  x & \\
  b & b \\
  \text{He} & \text{is}
\end{align*}

\begin{align*}
  z_2 & \\
  b \rightarrow a
\end{align*}

dual values:

\begin{align*}
  y^{(7)} & 2.05 \\
  z^{(7)} & 0.05 \\
  L(u^{(7)}) & 2.10
\end{align*}

previous solutions:

\begin{align*}
  y_3 & z_2 \\
  y_2 & z_1 \\
  y_1 & z_1 \\
  y_1 & z_1 \\
  y_2 & z_2 \\
  y_1 & z_1 \\
  y_2 & z_2
\end{align*}
round 8

dual solutions:

\[ x \]
\[ \begin{array}{c}
 y_1 \\
 z_1 \\
 a \\
 \text{He is}
\end{array} \]

\[ \begin{array}{c}
 a \\
 \rightarrow \\
 b
\end{array} \]

\[ \text{Round 8 dual values:} \]

\[ y^{(8)} = 2.09 \]
\[ z^{(8)} = 0.09 \]
\[ L(u^{(8)}) = 2.19 \]

previous solutions:

\[ y_3 \quad z_2 \]
\[ y_2 \quad z_1 \]
\[ y_1 \quad z_1 \]
\[ y_1 \quad z_1 \]
\[ y_2 \quad z_2 \]
\[ y_1 \quad z_1 \]
\[ y_2 \quad z_2 \]
\[ y_1 \quad z_1 \]
**round 9**

**dual solutions:**

\[
\begin{align*}
Y_2 & \\
X & \\
b & b \\
\text{He} & \text{is}
\end{align*}
\]

\[
\begin{align*}
z_2 & \\
b & \rightarrow a \\
\text{He} & \text{is}
\end{align*}
\]

**dual values:**

\[
\begin{align*}
y^{(9)} & 2.03 \\
z^{(9)} & 0.03 \\
L(u^{(9)}) & 2.06
\end{align*}
\]

**previous solutions:**

\[
\begin{align*}
y_3 & z_2 \\
y_2 & z_1 \\
y_1 & z_1 \\
y_1 & z_1 \\
y_2 & z_2 \\
y_1 & z_1 \\
y_2 & z_2 \\
y_1 & z_1 \\
y_2 & z_2 \\
y_1 & z_1
\end{align*}
\]
modify:

- extend $\mathcal{Y}$, $\mathcal{Z}$ to identify bigrams of part-of-speech tags
- $y(i, t_1, t_2) = 1 \iff y(i, t_1) = 1$ and $y(i + 1, t_2) = 1$
- $z(i, t_1, t_2) = 1 \iff z(i, t_1) = 1$ and $z(i + 1, t_2) = 1$

all bigram constraints: valid to add for all $i, t_1, t_2 \in \mathcal{T}$

$$
\sum_y \alpha_y y(i, t_1, t_2) = \sum_z \beta_z z(i, t_1, t_2)
$$

however this would make decoding expensive
Iterative tightening

single bigram constraint: cheaper to implement

\[ \sum_y \alpha_y y(1, a, b) = \sum_z \beta_z z(1, a, b) \]

the solution \( \alpha^{(1)}, \beta^{(1)} \) trivially passes this constraint, while \( \alpha^{(2)}, \beta^{(2)} \) violates it
Dual decomposition with tightening

tightened decomposition includes an additional Lagrange multiplier

\[ y_{u,v} = \arg \max_{y \in \mathcal{Y}} f(y) + \sum_{i,t} u(i, t)y(i, t) + v(1, a, b)y(1, a, b) \]

\[ z_{u,v} = \arg \max_{z \in \mathcal{Z}} g(z) - \sum_{i,t} u(i, t)z(i, t) - v(1, a, b)z(1, a, b) \]

in general, this term can make the decoding problem more difficult

example:

- for small examples, these penalties are easy to compute
- for CFG parsing, need to include extra states that maintain tag bigrams (still faster than full intersection)
round 7

dual solutions:

\[
\begin{array}{c}
y_3 \\
x \\
c \\
C \\
\text{He} \\
is \\
\end{array}
\quad
\begin{array}{c}
z_2 \\
\text{He} \\
is \\
\end{array}
\]

dual values:

\[
\begin{array}{c}
y^{(7)} \quad 2.00 \\
z^{(7)} \quad 1.00 \\
L(u^{(7)}) \quad 3.00 \\
\end{array}
\]

previous solutions:

\[
\begin{array}{c}
y_3 \quad z_2 \\
y_2 \quad z_3 \\
y_1 \quad z_2 \\
y_3 \quad z_1 \\
y_2 \quad z_3 \\
y_1 \quad z_2 \\
y_3 \quad z_3 \\
\end{array}
\]
round 8
dual solutions:

\[ x \]
\[ b \]
\[ \text{He is} \]
\[ y_2 \]
\[ z_3 \]

\[ c \rightarrow c \]

dual values:
\[ y^{(8)} = 3.00 \]
\[ z^{(8)} = 2.00 \]
\[ L(u^{(8)}) = 5.00 \]

previous solutions:
\[ y_3 \quad z_2 \]
\[ y_2 \quad z_3 \]
round 9

dual solutions:

\[
\begin{align*}
\text{dual values:} \\
y^{(9)} & \quad 3.00 \\
z^{(9)} & \quad -1.00 \\
L(u^{(9)}) & \quad 2.00 \\
\end{align*}
\]

previous solutions:

\[
\begin{align*}
y_3 & \quad z_2 \\
y_2 & \quad z_3 \\
y_1 & \quad z_2 \\
\end{align*}
\]
round 10

dual solutions:

\[ y_3 \]
\[ x \]
\[ c \quad c \]
\[ \text{He} \quad \text{is} \]
\[ z_1 \]

\[ a \rightarrow b \]

previous solutions:

\[ y_3 \quad z_2 \]
\[ y_2 \quad z_3 \]
\[ y_1 \quad z_2 \]
\[ y_3 \quad z_1 \]

dual values:

\[ y^{(10)} = 2.00 \]
\[ z^{(10)} = 1.00 \]
\[ L(u^{(10)}) = 3.00 \]

Round

0 1 2 3 4 5

7 8 9 10 11 12 13 14 15 16

Round
round 11

dual solutions:

\[ y_2 \]
\[ x \]
\[ b \]
\[ b \]
\[ He \]
\[ is \]

\[ z_3 \]
\[ c \rightarrow c \]
\[ He \]
\[ is \]

dual values:

\[ y^{(11)} \]: 3.00
\[ z^{(11)} \]: 2.00
\[ L(u^{(11)}) \]: 5.00

previous solutions:

\[ y_3 \] \[ z_2 \]
\[ y_2 \] \[ z_3 \]
\[ y_1 \] \[ z_2 \]
\[ y_3 \] \[ z_1 \]
\[ y_2 \] \[ z_3 \]
round 12

dual solutions:

\[ y_1 \]
\[ x \]
\[ a \]
\[ \text{He is} \]
\[ z_2 \]
\[ b \rightarrow a \]

\[ L(u^{(12)}) \]
\[ y^{(12)} \]
\[ z^{(12)} \]

\[ 2.00 \]
\[ 3.00 \]
\[ -1.00 \]

previous solutions:

\[ y_3 \quad z_2 \]
\[ y_2 \quad z_3 \]
\[ y_1 \quad z_2 \]
\[ y_3 \quad z_1 \]
\[ y_2 \quad z_3 \]
\[ y_1 \quad z_2 \]
round 13

dual solutions:

\[
\begin{align*}
  y_3 & \quad \text{He is} \\
  x & \quad \text{He is} \\
  c & \quad c \\
  a & \quad b \\
  z_1 & \quad \text{He is}
\end{align*}
\]

dual values:

\[
\begin{align*}
  y^{(13)} & = 2.00 \\
  z^{(13)} & = -1.00 \\
  L(u^{(13)}) & = 1.00
\end{align*}
\]

previous solutions:

\[
\begin{align*}
  y_3 & \quad z_2 \\
  y_2 & \quad z_3 \\
  y_1 & \quad z_2 \\
  y_3 & \quad z_1 \\
  y_2 & \quad z_3 \\
  y_1 & \quad z_2 \\
  y_3 & \quad z_1
\end{align*}
\]
round 14

dual solutions:

\[ \begin{align*}
    y_2 & \quad \text{He} & \quad z_3 \\
    x & \quad b & \quad b \\
    & \quad \text{He} & \quad \text{is}
\end{align*} \]

dual values:

\[ \begin{align*}
    y^{(14)} & \quad 3.00 \\
    z^{(14)} & \quad 2.00 \\
    L(u^{(14)}) & \quad 5.00
\end{align*} \]

previous solutions:

\[ \begin{align*}
    y_3 & \quad z_2 \\
    y_2 & \quad z_3 \\
    y_1 & \quad z_2 \\
    y_3 & \quad z_1 \\
    y_2 & \quad z_3 \\
    y_1 & \quad z_2 \\
    y_3 & \quad z_1 \\
    y_2 & \quad z_3 \\
\end{align*} \]
round 15

dual solutions:

\[ \begin{align*}
    &y_1 \\
    &x \\
    &\text{He} \\
    &a \\
    &\text{is} \\
    &y_3 \\
    &b \\
    &\text{He} \\
    &a \\
    &\text{is} \\
    &z_2 \\
\end{align*} \]

\[ \begin{align*}
\text{dual values:} \\
y^{(15)} &\quad 3.00 \\
z^{(15)} &\quad -1.00 \\
L(u^{(15)}) &\quad 2.00 \\
\end{align*} \]

previous solutions:

\[ \begin{align*}
    &y_3 \\
    &z_2 \\
    &y_2 \\
    &z_3 \\
    &y_1 \\
    &z_2 \\
    &y_3 \\
    &z_1 \\
    &y_2 \\
    &z_3 \\
    &y_1 \\
    &z_2 \\
\end{align*} \]
round 16

dual solutions:

\[
\begin{align*}
&\text{Table}\quad \begin{array}{cc}
\text{Round} & \text{dual values:} \\
& y^{(16)} & 2.00 \\
& z^{(16)} & -2.00 \\
& L(u^{(16)}) & 0.00
\end{array} \\
&\text{previous solutions:} \\
& y_3 & z_2 \\
& y_2 & z_3 \\
& y_1 & z_2 \\
& y_3 & z_1 \\
& y_2 & z_3 \\
& y_1 & z_2 \\
& y_3 & z_1 \\
& y_2 & z_3 \\
& y_1 & z_2 \\
& y_3 & z_3
\end{align*}
\]

\[
\begin{align*}
&\text{Round} \\
&7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16
\end{align*}
\]
6. Advanced examples

**aim:** demonstrate some different relaxation techniques

- higher-order non-projective dependency parsing
- syntactic machine translation
Higher-order non-projective dependency parsing

setup: given a model for higher-order non-projective dependency parsing (sibling features)

problem: find non-projective dependency parse that maximizes the score of this model

difficulty:
• model is NP-hard to decode
• complexity of the model comes from enforcing combinatorial constraints

strategy: design a decomposition that separates combinatorial constraints from direct implementation of the scoring function
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
- ...
Non-Projective Dependency Parsing

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

\[
f(y) = \text{score}(\text{head} = 0, \text{mod} = 2) + \text{score}(2, 1) + \text{score}(2, 4) + \text{score}(2, 5) + \text{score}(4, 3) + \ldots
\]

\[
e.g. \quad \text{score}(0, 2) = \log p(2|0) \quad \text{(generative model)}
\]

\[
\text{or} \quad \text{score}(0, 2) = w \cdot \phi(2, 0) \quad \text{(CRF/perceptron model)}
\]

\[
y^* = \arg \max_y f(y) \quad \text{\text{Minimum Spanning Tree Algorithm}}
\]
Arc-Factored

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

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

\[ f(y) = \text{score}(\text{head} = \ast_0, \text{mod} = \text{saw}_2) + \text{score}(\text{saw}_2, \text{John}_1) + \text{score}(\text{saw}_2, \text{movie}_4) + \text{score}(\text{saw}_2, \text{today}_5) \]
Arc-Factored

\[
f(y) = score(head = \ast_0, \text{mod} = \text{saw}_2) + score(\text{saw}_2, \text{John}_1) + score(\text{saw}_2, \text{movie}_4) + score(\text{saw}_2, \text{today}_5) + score(\text{movie}_4, \text{a}_3) + \ldots
\]
Arc-Factored

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

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

\[ + \text{score}(\text{movie}_4, \text{a}_3) + ... \]

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

\[ f(y) = \text{score}(\text{head} = \ast_0, \text{mod} = \text{saw}_2) + \text{score}(\text{saw}_2, \text{John}_1) \]
\[ + \text{score}(\text{saw}_2, \text{movie}_4) + \text{score}(\text{saw}_2, \text{today}_5) \]
\[ + \text{score}(\text{movie}_4, \text{a}_3) + \ldots \]

e.g. \( \text{score}(\ast_0, \text{saw}_2) = \log p(\text{saw}_2|\ast_0) \) (generative model)

or \( \text{score}(\ast_0, \text{saw}_2) = w \cdot \phi(\text{saw}_2, \ast_0) \) (CRF/perceptron 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)

\[ y^* = \arg \max_y f(y) \Leftarrow \text{Minimum Spanning Tree Algorithm} \]
Sibling Models

\[ f(y) = \]

\[ \text{score}(\text{head} = \ast_0, \text{prev} = \text{NULL}, \text{mod} = \ldots) = \log p(\text{today} | \text{saw}, \text{movie}) \]

or

\[ \text{score}(\text{saw}, \text{movie}, \text{today}) = w \cdot \phi(\text{saw}, \text{movie}, \text{today}) \]

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

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

\[ f(y) = \text{score}(\text{head} = *_0, \text{prev} = \text{NULL}, \text{mod} = \text{saw}_2) + \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) \]
Sibling Models

\[ f(y) = score(head = *_0, prev = NULL, mod = saw_2) + score(saw_2, NULL, John_1) + score(saw_2, NULL, movie_4) \]
Sibling Models

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

\[ + \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{movie}_4) \]

\[ + \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) + \ldots \]
Sibling Models

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

\[ + \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{movie}_4) \]

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

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

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

\[ + \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{movie}_4) \]

\[ + \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) + ... \]

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

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

*0 John1 saw2 a3 movie4 today5 that6 he7 liked8

\[ \text{score(saw2, NULL, John1)} + \ldots + \text{score(saw2, a3, he7)} \]

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

Under Sibling Model, can solve for each word with Viterbi decoding.
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

\[ \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{movie}_4) \]
\[ + \text{score}(\text{saw}_2, \text{movie}_4, \text{today}_5) \]
\[ + \text{score}(\text{saw}_2, \text{movie}_4, \text{that}_6) \]

\[ \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{that}_6) \]
Thought Experiment: Individual Decoding

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

\[ \text{score}(\text{saw}_2, \text{NULL}, \text{John}_1) + \text{score}(\text{saw}_2, \text{NULL}, \text{that}_6) \]

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

* \( 0 \) John_1 \( 1 \) saw \( 2 \) a \( 3 \) movie \( 4 \) today \( 5 \) that \( 6 \) he \( 7 \) liked \( 8 \)

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

\[ + \text{score}(\text{saw}_2, \text{NULL}, \text{that}_6) \]

\[ + \text{score}(\text{saw}_2, \text{a}_3, \text{he}_7) \]

\( 2^{n-1} \) possibilities

Under Sibling Model, can solve for each word with Viterbi decoding.
Thought Experiment: Individual Decoding

*0 John 1 saw 2 a 3 movie 4 today 5 that 6 he 7 liked 8

\[
\text{score}(\text{saw}, \text{NULL}, \text{John}) + \text{score}(\text{saw}, \text{NULL}, \text{movie}) + \text{score}(\text{saw}, \text{movie}, \text{today})
\]

\[
\text{score}(\text{saw}, \text{NULL}, \text{John}) + \text{score}(\text{saw}, \text{NULL}, \text{that}) + \text{score}(\text{saw}, \text{a}, \text{he})
\]

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

Under Sibling Model, can solve for each word with Viterbi decoding.
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.
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.
<table>
<thead>
<tr>
<th></th>
<th>No Constraints</th>
<th>Tree Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc-Factored</td>
<td></td>
<td>Minimum Spanning Tree</td>
</tr>
<tr>
<td>Sibling Model</td>
<td>Individual Decoding</td>
<td></td>
</tr>
</tbody>
</table>
## Dual Decomposition Idea

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

- Minimum Spanning Tree
Goal $y^* = \arg \max_{y \in Y} f(y)$
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 $z = y$.
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

Constraint
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

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

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

Constraint
Algorithm Sketch

Set penalty weights equal to 0 for all edges.

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

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

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

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 $K$

\[ z^{(k)} \leftarrow \text{Decode} \left( f(z) + \text{penalty} \right) \text{ by Individual Decoding} \]
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 $K$

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

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

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 \( K \)

\[
\begin{align*}
    z^{(k)} &\leftarrow \text{Decode } (f(z) + \text{penalty}) \text{ by Individual Decoding} \\
    y^{(k)} &\leftarrow \text{Decode } (g(y) - \text{penalty}) \text{ by Minimum Spanning Tree} \\
    \text{If } y^{(k)}(i,j) = z^{(k)}(i,j) \text{ for all } i,j \quad \text{Return } (y^{(k)}, z^{(k)}) \\
    \text{Else } \quad \text{Update penalty weights based on } y^{(k)}(i,j) - z^{(k)}(i,j)
\end{align*}
\]
**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 \( i,j \)

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

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

**Penalties**

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

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

**Minimum Spanning Tree**

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

**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

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

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 \( i,j \)

Penalties

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

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

Key

- \( f(z) \) \( \iff \) Sibling Model
- \( g(y) \) \( \iff \) Arc-Factored Model
- \( \mathcal{Z} \) \( \iff \) No Constraints
- \( \mathcal{Y} \) \( \iff \) 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

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

Key

\[
\begin{align*}
  f(z) & \iff \text{Sibling Model} \\
  Z & \iff \text{No Constraints} \\
  y(i,j) = 1 & \text{if } y \text{ contains dependency } i,j \\
  g(y) & \iff \text{Arc-Factored Model} \\
  Y & \iff \text{Tree Constraints}
\end{align*}
\]
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

- \( f(z) \) \( \Leftarrow \) Sibling Model
- \( g(y) \) \( \Leftarrow \) Arc-Factored Model
- \( \mathcal{Z} \) \( \Leftarrow \) No Constraints
- \( \mathcal{Y} \) \( \Leftarrow \) 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

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

Key

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

Penalties

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

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

Penalties

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

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

\[ f(z) \quad \Leftarrow \text{Sibling Model} \quad g(y) \quad \Leftarrow \text{Arc-Factored Model} \]

\[ Z \quad \Leftarrow \text{No Constraints} \quad \mathcal{Y} \quad \Leftarrow \text{Tree Constraints} \]

\[ y(i,j) = 1 \quad \text{if} \quad y \text{ contains dependency } i,j \]
**Individual Decoding**

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

**Minimum Spanning Tree**

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

**Key**

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

**Penalties**

<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>

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

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

Key

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

Penalties

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

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

Converged

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

\[ f(y) = \ldots + \text{score}(gp = \ast_0, head = \text{saw}_2, prev = \text{movie}_4, mod = \text{today}_5) \]

- Head Automata (Eisner, 2000)

Generalization of Sibling models

Allow arbitrary automata as local scoring function.
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.

- Cze
- Eng
- Dan
- Dut
- Por
- Slo
- Swe
- Tur

Percentage of examples where the dual decomposition finds an exact solution.
Parsing Speed

- 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

\[ \text{Individual} \quad \text{MST} \quad \text{Dual} \]

\[ F_1 \text{ for dependency accuracy} \]
**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**
  - LP(1)
  - LP(2)
  - ILP
  - Dual

- **Sentences per second**
  - LP(1)
  - LP(2)
  - ILP
  - Dual
Syntactic translation decoding

**setup:** assume a trained model for syntactic machine translation

**problem:** find best derivation that maximizes the score of this model

**difficulty:**
- need to incorporate language model in decoding
- empirically, relaxation is often not tight, so dual decomposition does not always converge

**strategy:**
- use a different relaxation to handle language model
- incrementally add constraints to find exact solution
Syntactic Translation

Problem:
Decoding synchronous grammar for machine translation

Example:

<s> abarks le dug </s>
<s> the dog barks loudly </s>

Goal:

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

where \( y \) is a parse derivation in a synchronous grammar
Hiero Example

Consider the input sentence

<s> abarks le dug </s>

And the synchronous grammar

S → <s> X </s>, <s> X </s>
X → abarks X, X barks loudly
X → abarks X, barks X
X → abarks X, barks X loudly
X → le dug, the dog
X → le dug, a cat
Hiero Example

Apply synchronous rules to map this sentence

Many possible mappings:

<s> the dog barks loudly </s>
<s> a cat barks loudly </s>
<s> barks the dog </s>
<s> barks a cat </s>
<s> barks the dog loudly </s>
<s> barks a cat loudly </s>
## Translation Forest

<table>
<thead>
<tr>
<th>Rule</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \rightarrow &lt;s&gt; \ 4 \ &lt;/s&gt;$</td>
<td>-1</td>
</tr>
<tr>
<td>$4 \rightarrow 5 \ \text{barks loudly}$</td>
<td>2</td>
</tr>
<tr>
<td>$4 \rightarrow \text{barks 5}$</td>
<td>0.5</td>
</tr>
<tr>
<td>$4 \rightarrow \text{barks 5 loudly}$</td>
<td>3</td>
</tr>
<tr>
<td>$5 \rightarrow \text{the dog}$</td>
<td>-4</td>
</tr>
<tr>
<td>$5 \rightarrow \text{a cat}$</td>
<td>2.5</td>
</tr>
</tbody>
</table>

**Example:** a derivation in the translation forest

$$<s> 4 5 \ a \ \text{cat} \ \text{barks loudly} \ </s>$$
Scoring function

**Score**: sum of hypergraph derivation and language model

\[ f(y) = \text{score}(5 \rightarrow \text{a cat}) \]
Scoring function

**Score**: sum of hypergraph derivation and language model

\[ f(y) = score(5 \rightarrow \text{a cat}) + score(4 \rightarrow 5 \text{ barks loudly}) \]
Scoring function

Score: sum of hypergraph derivation and language model

\[
f(y) = \text{score}(5 \rightarrow \text{a cat}) + \text{score}(4 \rightarrow 5 \text{ barks loudly}) + \ldots \\
+ \text{score}(\langle s \rangle, \text{the})
\]
**Scoring function**

**Score**: sum of hypergraph derivation and language model

\[ f(y) = score(5 \rightarrow a \text{ cat}) + score(4 \rightarrow 5 \text{ barks loudly}) + \ldots \]

\[ + score(<s>, a) + score(a, \text{ cat}) \]
Exact Dynamic Programming

To maximize combined model, need to ensure that bigrams are consistent with parse tree.

```
<s> 4
5
a cat
barks loudly
</s>
```
Exact Dynamic Programming

To maximize combined model, need to ensure that bigrams are consistent with parse tree.

Original Rules

| 5 → the dog |
| 5 → a cat |

New Rules

| <s>5cat → <s>thethe thedogdog |
| barks5cat → barks thethe thedogdog |
| <s>5cat → <s>a a catcat |
| barks5cat → barks a a catcat |
Lagrangian Relaxation Algorithm for Syntactic Translation

Outline:

• Algorithm for simplified version of translation
• Full algorithm with certificate of exactness
• Experimental results
Thought experiment: Greedy language model

Choose best bigram for a given word

- score(\langle s\rangle, \text{barks})
Thought experiment: Greedy language model

Choose best bigram for a given word

- $\text{score}(\langle s \rangle, \text{barks})$
- $\text{score}(\text{dog}, \text{barks})$
Thought experiment: Greedy language model

Choose best bigram for a given word

- $\text{score}(<s>, \text{barks})$
- $\text{score}(\text{dog}, \text{barks})$
- $\text{score}(\text{cat}, \text{barks})$
Thought experiment: Greedy language model

Choose best bigram for a given word

\[ \text{barks} \]

- \( \text{score}(<s>, \text{barks}) \)
- \( \text{score}(\text{dog}, \text{barks}) \)
- \( \text{score}(\text{cat}, \text{barks}) \)

Can compute with a simple maximization

\[ \arg \max_{w: (w, \text{barks}) \in B} \text{score}(w, \text{barks}) \]
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word

\[
\text{barks} \quad \text{loudly} \quad \text{the} \quad \text{dog} \quad \text{a} \quad \text{cat}
\]
Thought experiment: Full decoding

**Step 1.** Greedily choose best bigram for each word
Thought experiment: Full decoding

**Step 1.** Greedily choose best bigram for each word
Thought experiment: Full decoding

**Step 1.** Greedily choose best bigram for each word

\[
\begin{align*}
\text{barks} & \quad \text{loudly} & \quad \text{the} & \quad \text{dog} & \quad \text{a} & \quad \text{cat} \\
\text{barks} & \quad \text{dog} & \quad \text{barks} & \quad \langle s \rangle
\end{align*}
\]
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word

"barks loudly the dog a cat"
Thought experiment: Full decoding

**Step 1.** Greedily choose best bigram for each word

```plaintext
</s>  barks  loudly  the  dog  a  cat
barks  dog  barks  <s>  the  <s>
```
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word

\[
\langle s \rangle \quad \text{barks} \quad \text{loudly} \quad \text{the} \quad \text{dog} \quad \text{a} \quad \text{cat}
\]
Thought experiment: Full decoding

**Step 1.** Greedily choose best bigram for each word

```
</s>  
 barks  
  
 dog  
  
 loudly  
  
 the  
  
 dog  
  
 a  
  
 cat
```

**Step 2.** Find the best derivation with fixed bigrams
Thought experiment: Full decoding

**Step 1.** Greedily choose best bigram for each word

```
<s> barks
   |
   v
barks
dog |
    |
    v
barks
<s> the
   |
   v
the
dog |
    |
    v
the
<s> a
   |
   v
a
<>/s>
cat
```

**Step 2.** Find the best derivation with fixed bigrams

```
<s> 1
   |
   v
<s> 4
   |
   v
<s> 5
   |
   v
a
   |
   v
<s> barks
dog |
    |
    v
barks
   |
   v
loudly
   |
   v
barks
   |
   v
<s> a
   |
   v
<s> <>/s>
cat
```

Thought Experiment Problem

May produce invalid parse and bigram relationship

Greedy bigram selection may conflict with the parse derivation
Thought Experiment Problem

May produce invalid parse and bigram relationship

Greedy bigram selection may conflict with the parse derivation
Formal objective

Notation: \( y(w, v) = 1 \) if the bigram \( \langle w, v \rangle \in B \) is in \( y \)

Goal:

\[
\arg \max_{y \in Y} f(y)
\]

such that for all words nodes \( y_v \)

(1)
Formal objective

Notation: \( y(w, v) = 1 \) if the bigram \( \langle w, v \rangle \in B \) is in \( y \)

Goal:

\[
\arg \max_{y \in \mathcal{Y}} f(y)
\]

such that for all words nodes \( y_v \)

\[
y_v = \sum_{w: \langle w, v \rangle \in B} y(w, v) \quad (1)
\]
Formal objective

Notation: \( y(w, v) = 1 \) if the bigram \( \langle w, v \rangle \in B \) is in \( y \)

Goal:

\[
\arg \max_{y \in Y} f(y)
\]

such that for all words nodes \( v \)

\[
y_v = \sum_{w: \langle w, v \rangle \in B} y(w, v) \quad (1)
\]
\[
y_v = \sum_{w: \langle v, w \rangle \in B} y(v, w) \quad (2)
\]
Formal objective

Notation: \( y(w, v) = 1 \) if the bigram \( \langle w, v \rangle \in \mathcal{B} \) is in \( y \)

Goal:
\[
\arg \max_{y \in \mathcal{Y}} f(y)
\]
such that for all words nodes
\[
y_v = \sum_{w: \langle w, v \rangle \in \mathcal{B}} y(w, v) \quad (1)
\]
\[
y_v = \sum_{w: \langle v, w \rangle \in \mathcal{B}} y(v, w) \quad (2)
\]
Formal objective

Notation: \( y(w, v) = 1 \) if the bigram \( \langle w, v \rangle \in B \) is in \( y \)

Goal:

\[
\arg \max_{y \in Y} f(y)
\]

such that for all words nodes \( y_v \)

\[
y_v = \sum_{w: \langle w, v \rangle \in B} y(w, v) \tag{1}
\]

\[
y_v = \sum_{w: \langle v, w \rangle \in B} y(v, w) \tag{2}
\]

Lagrangian: Relax constraint (2), leave constraint (1)

\[
L(u, y) = \max_{y \in Y} f(y) + \sum_{v} u(v) \left( y_v - \sum_{w: \langle v, w \rangle \in B} y(v, w) \right)
\]

For a given \( u \), \( L(u, y) \) can be solved by our greedy LM algorithm
Algorithm

Set \( u^{(1)}(v) = 0 \) for all \( v \in V_L \)

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

\[
y^{(k)} \leftarrow \arg \max_{y \in Y} L^{(k)}(u, y)
\]

If \( y^{(k)}_v = \sum_{w : \langle v, w \rangle \in B} y^{(k)}(v, w) \) for all \( v \) Return \( y^{(k)} \)

Else

\[
u^{(k+1)}(v) \leftarrow u^{(k)}(v) - \alpha_k \left( y^{(k)}_v - \sum_{w : \langle v, w \rangle \in B} y^{(k)}(v, w) \right)
\]
Thought experiment: Greedy with penalties
Choose best bigram with penalty for a given word

\[ \text{score}(<s>, \text{barks}) - u(<s>) + u(\text{barks}) \]
Thought experiment: Greedy with penalties

Choose best bigram with penalty for a given word

\[ \text{score}(\langle s \rangle, \text{barks}) - u(\langle s \rangle) + u(\text{barks}) \]

\[ \text{score}(\text{cat}, \text{barks}) - u(\text{cat}) + u(\text{barks}) \]
Thought experiment: Greedy with penalties
Choose best bigram with penalty for a given word

- $\text{score}(\text{<s>}, \text{barks}) - u(\text{<s>}) + u(\text{barks})$
- $\text{score}((\text{cat}, \text{barks}) - u(\text{cat}) + u(\text{barks})$
- $\text{score}(\text{dog}, \text{barks}) - u(\text{dog}) + u(\text{barks})$
Thought experiment: Greedy with penalties

Choose best bigram with penalty for a given word

- $\text{score}(\langle s\rangle, \text{barks}) - u(\langle s\rangle) + u(\text{barks})$
- $\text{score}(\text{cat}, \text{barks}) - u(\text{cat}) + u(\text{barks})$
- $\text{score}(\text{dog}, \text{barks}) - u(\text{dog}) + u(\text{barks})$

Can still compute with a simple maximization over

$$\arg \max_{w : (w, \text{barks}) \in B} \text{score}(w, \text{barks}) - u(w) + u(\text{barks})$$
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>v</th>
<th>&lt;/s&gt;</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(v)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Greedy decoding
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>v</th>
<th>&lt;/s&gt;</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(v)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Greedy decoding
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>( v )</th>
<th>(&lt;s&gt;) barks loudly the dog a cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u(v) )</td>
<td>0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Greedy decoding

[Diagram showing a tree structure with nodes labeled with words and penalties represented as a table.]
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>v</th>
<th>&lt;s&gt;</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(v)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Greedy decoding

```
<s> barks loudly the dog a cat
a
dog barks

1
5
barks
loudly
barks

cat
dog
<s>
<s>
a
```
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>v</th>
<th>(&lt;s&gt;)</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(v)</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Greedy decoding
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>$v$</th>
<th>$u(v)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;$s$&gt;$</td>
<td>0</td>
</tr>
<tr>
<td>barks</td>
<td>-1</td>
</tr>
<tr>
<td>loudly</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>0</td>
</tr>
<tr>
<td>dog</td>
<td>-1</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
</tr>
</tbody>
</table>

Greedy decoding
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>v</th>
<th>&lt;/s&gt;</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(v)</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Greedy decoding

```text
<loudly>
  /s/     barks
  cat     loudly
  the     barks
  <s>     the
  <s>     a
  cat
```

```text
barks loudly the dog a cat
```
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>v</th>
<th>barks loudly the dog a cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(v)</td>
<td>0</td>
</tr>
</tbody>
</table>

Greedy decoding
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\langle /s\rangle$</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u(v)$</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Greedy decoding

1

5

<s> the
dog
cat

<s> the

4

<s> barks

loudly

<s> the
dog

<s> a

</s> cat

barks loudly the dog a cat
loudly cat barks <s> the <s> a

The highlighted path represents the greedy decoding path.
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>v</th>
<th>&lt;/s&gt;</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(v)</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-0.5</td>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Greedy decoding
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>v</th>
<th>&lt;/s&gt;</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(v)</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-0.5</td>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Greedy decoding
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>v</th>
<th>&lt;/s&gt;</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>u(v)</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-0.5</td>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Greedy decoding
Algorithm example

Penalties

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\langle/s\rangle$</th>
<th>barks</th>
<th>loudly</th>
<th>the</th>
<th>dog</th>
<th>a</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u(v)$</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-0.5</td>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Greedy decoding
**Constraint Issue**

Constraints do not capture all possible reorderings

**Example:** Add rule $\langle 5 \rightarrow \text{cat a} \rangle$ to forest. New derivation

\[
<s>4
5
\text{cat a}
\text{barks loudly}
</s>

Satisfies both constraints (1) and (2), but is not self-consistent.
Constraint Issue

Constraints do not capture all possible reorderings

**Example:** Add rule \( \langle 5 \rightarrow \text{cat a} \rangle \) to forest. New derivation

Satisfies both constraints (1) and (2), but is not self-consistent.
New Constraints: Paths

Fix: In addition to bigrams, consider paths between terminal nodes

Example: Path marker \langle 5 \downarrow, 10 \downarrow \rangle implies that between two word nodes, we move down from node 5 to node 10
New Constraints: Paths

Fix: In addition to bigrams, consider paths between terminal nodes.

Example: Path marker ⟨5 ↓, 10 ↓⟩ implies that between two word nodes, we move down from node 5 to node 10.
Fix: In addition to bigrams, consider paths between terminal nodes

Example: Path marker \(\langle 5 \downarrow, 10 \downarrow \rangle\) implies that between two word nodes, we move down from node 5 to node 10.
Fix: In addition to bigrams, consider paths between terminal nodes.

Example: Path marker \(\langle 5 \downarrow, 10 \downarrow \rangle\) implies that between two word nodes, we move down from node 5 to node 10.

New Constraints: Paths
Fix: In addition to bigrams, consider paths between terminal nodes.

Example: Path marker \(\langle 5 \downarrow, 10 \downarrow \rangle\) implies that between two word nodes, we move down from node 5 to node 10.
Greedy Language Model with Paths

Step 1. Greedily choose best path each word

```plaintext
<html><body>
< /s >
< barks >
< loudly >
< the >
< dog >
< a >
< cat >
</body></html>
```
Greedy Language Model with Paths

Step 1. Greedily choose best path each word

- `<s>`
- `barks`
- `loudly`
- `the`
- `dog`
- `a`
- `cat`
Greedy Language Model with Paths

Step 1. Greedily choose best path each word
Greedy Language Model with Paths

Step 1. Greedily choose best path each word

< /s>
< 4 ↑, </s> ↓>
< loudly ↑, 4 ↓>
< loudly ↑>

< barks ↓>
< 5 ↑, barks ↓>
< loudly ↓, barks ↑>
< loudly ↓>

< loudly ↓>
< barks ↓>
< cat ↑, 5 ↑>
< cat ↑>

< loudly ↓, barks ↑>
< barks ↑>
< 4 ↓, 5 ↓>
< <s> ↑, 4 ↓>
< <s> ↑>

< the ↓>
< the ↓>
< the ↓>
< the ↓>

< the ↓>
< 5 ↓, the ↓>
< 4 ↓, 5 ↓>
< <s> ↑, 4 ↓>
< <s> ↑>

< cat ↓>
< a ↑, cat ↓>
< a ↑>

< /s> barks loudly the dog a cat
< </s> ↓>
< loudly ↑, 4 ↓>
< loudly ↓, barks ↑>
< barks ↑>
< <s> ↑, 4 ↓>
< <s> ↑>

< cat ↓>
< a ↑, cat ↓>
< a ↑>
Greedy Language Model with Paths

Step 1. Greedily choose best path each word

<bark> barks loudly the dog a cat
< </bark>
Greedy Language Model with Paths

Step 1. Greedily choose best path each word

- `<s>`
  - `</s>`
    - `<>/s>`
    - `<4↑, </s>↓>`
    - `<loudly↑, 4↓>`
    - `<loudly↑>`

- `barks`
  - `<barks ↓>`
    - `<5↑, barks ↓>`
    - `<loudly ↓, barks ↑>`
    - `<cat ↑, 5↑>`
    - `<barks ↑>`
    - `<cat ↑>`

- `loudly`
  - `<loudly ↓>`
    - `<5↓, the ↓>`
    - `<4↓, 5↓>`
    - `<<s>↑, 4↓>`
    - `<<s>↑>`

- `the`
  - `<the ↓>`
    - `<the ↑>`
    - `<5↓, the ↓>`
    - `<the ↑, dog ↓>`
    - `<the ↑>`
    - `<4↓, 5↓>`

- `dog`
  - `<dog ↓>`
    - `<the ↑, dog ↓>`
    - `<the ↑>`
    - `<5↓, a ↓>`
    - `<4↓, 5↓>`

- `a`
  - `<a ↓>`
    - `<5↓, a ↓>`
    - `<4↓, 5↓>`
    - `<<s>↑, 4↓>`
    - `<<s>↑>`

- `cat`
  - `<cat ↓>`
    - `<cat ↓, loudly ↓>`
    - `<loudly ↓>`
    - `<loudly ↓>`
    - `<loudly ↓>`
    - `<loudly ↓>`
Greedy Language Model with Paths

Step 1. Greedily choose best path each word
Greedy Language Model with Paths (continued)

Step 2. Find the best derivation over these elements
Greedy Language Model with Paths (continued)

Step 2. Find the best derivation over these elements
Efficiently Calculating Best Paths

There are too many paths to compute argmax directly, but we can compactly represent all paths as a graph.

Graph is linear in the size of the grammar

- Green nodes represent leaving a word
- Red nodes represent entering a word
- Black nodes are intermediate paths
**Goal:** Find the best path between all word nodes (green and red)

**Method:** Run all-pairs shortest path to find best paths
Full Algorithm

Algorithm is very similar to simple bigram case. Penalty weights are associated with nodes in the graph instead of just bigram words.

**Theorem**

If at any iteration the greedy paths agree with the derivation, then \( y^{(k)} \) is the global optimum.

But what if it does not find the global optimum?
Convergence

The algorithm is not guaranteed to converge
May get stuck between solutions.
Convergence

The algorithm is not guaranteed to converge
May get stuck between solutions.
Convergence

The algorithm is not guaranteed to converge

May get stuck between solutions.
Convergence

The algorithm is not guaranteed to converge

May get stuck between solutions.
Convergence

The algorithm is not guaranteed to converge

May get stuck between solutions.
Convergence

The algorithm is not guaranteed to converge

May get stuck between solutions.
Convergence

The algorithm is not guaranteed to converge

May get stuck between solutions.
Convergence

The algorithm is not guaranteed to converge
May get stuck between solutions.
Convergence

The algorithm is not guaranteed to converge

May get stuck between solutions.

Can fix this by incrementally adding constraints to the problem
Tightening

Main idea: Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm)

Example:

Partitions

A = \{2, 6, 7, 8, 9, 10, 11\}
B = {}
**Tightening**

**Main idea:** Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict.
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm).

**Example:**

Partitions

A = \{2, 6, 7, 8, 9, 10, 11\}

B = {}
Tightening

Main idea: Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict.
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm).

Example:

Partitions

\[
A = \{2,6,7,8,9,10,11\}
\]

\[
B = \{\}
\]
Tightening

**Main idea:** Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict.
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm).

**Example:**

Partitions

\[ A = \{2, 6, 7, 8, 9, 10, 11\} \]
\[ B = \{\} \]
Tightening

**Main idea:** Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict.
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm).

**Example:**

```
<s> 4
5
the
dog
barks loudly
</s>
<s> the
cat barks
loudly
```

**Partitions**

A = \{2,6,7,8,9,10\}
B = \{11\}
**Tightening**

**Main idea:** Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict.
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm).

**Example:**

```
<s> the dog barks loudly</s>
<s> the cat barks loudly </s>
```

Partitions

A = \{2, 6, 7, 8, 9, 10\}

B = \{11\}
Tightening

**Main idea:** Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict.
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm).

**Example:**

![Diagram showing partitioning of words]

**Partitions**

A = \{2,6,7,8,9,10\}

B = \{11\}
Experiments

Properties:
- Exactness
- Translation Speed
- Comparison to Cube Pruning

Model:
- Tree-to-String translation model (Huang and Mi, 2010)
- Trained with MERT

Experiments:
- NIST MT Evaluation Set (2008)
Exactness

Percent Exact

LR
ILP
DP
LP

LR  Lagrangian Relaxation
ILP  Integer Linear Programming
DP  Exact Dynamic Programming
LP  Linear Programming
Median Speed

Sentences Per Second

- LR: Lagrangian Relaxation
- ILP: Integer Linear Programming
- DP: Exact Dynamic Programming
- LP: Linear Programming
Comparison to Cube Pruning: Exactness

<table>
<thead>
<tr>
<th>Percent Exact</th>
<th>LR</th>
<th>Cube(50)</th>
<th>Cube(500)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LR: Lagrangian Relaxation
Cube(50): Cube Pruning (Beam=50)
Cube(500): Cube Pruning (Beam=500)
Comparison to Cube Pruning: Median Speed

<table>
<thead>
<tr>
<th>Method</th>
<th>Sentences Per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td></td>
</tr>
<tr>
<td>Cube (50)</td>
<td></td>
</tr>
<tr>
<td>Cube (500)</td>
<td></td>
</tr>
</tbody>
</table>

LR: Lagrangian Relaxation
Cube (50): Cube Pruning (Beam=50)
Cube (500): Cube Pruning (Beam=500)
The Phrase-Based Decoding Problem

- We have a source-language sentence \( x_1, x_2, \ldots, x_N \) (\( x_i \) is the \( i \)'th word in the sentence)

- A phrase \( p \) is a tuple \( (s, t, e) \) signifying that words \( x_s \ldots x_t \) have a target-language translation as \( e \)

- E.g., \( p = (2, 5, \text{the dog}) \) specifies that words \( x_2 \ldots x_5 \) have a translation as \( \text{the dog} \)

- Output from a phrase-based model is a derivation

\[
y = p_1 p_2 \ldots p_L
\]

where \( p_j \) for \( j = 1 \ldots L \) are phrases. A derivation defines a translation \( e(y) \) formed by concatenating the strings

\[
e(p_1)e(p_2)\ldots e(p_L)
\]
Scoring Derivations

- Each phrase $p$ has a score $g(p)$.

- For two consecutive phrases $p_k = (s, t, e)$ and $p_{k+1} = (s', t', e')$, the distortion distance is
  \[ \delta(t, s') = |t + 1 - s'| \]

- The score for a derivation is
  \[
  f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times \delta(t(p_k), s(p_{k+1}))
  \]
  where $\eta \in \mathbb{R}$ is the distortion penalty, and $h(e(y))$ is the language model score.
The Decoding Problem

- $\mathcal{Y}$ is the set of all valid derivations
- For a derivation $y$, $y(i)$ is the number of times word $i$ is translated
- A derivation $y = p_1, p_2, \ldots, p_L$ is valid if:
  - $y(i) = 1$ for $i = 1 \ldots N$
  - For each pair of consecutive phrases $p_k, p_{k+1}$ for $k = 1 \ldots L - 1$, we have $\delta(t(p_k), s(p_{k+1})) \leq d$, where $d$ is the distortion limit.
- Decoding problem is to find
$$\arg\max_{y \in \mathcal{Y}} f(y)$$
Exact Dynamic Programming

- We can find \( \arg \max_{y \in Y} f(y) \) using dynamic programming

- **But**, the runtime (and number of states) is exponential in \( N \).

- Dynamic programming states are of the form \((w_1, w_2, b, r)\)

  where
  
  - \( w_1, w_2 \) are last two words of a hypothesis
  - \( b \) is a bit-string of length \( N \), recording which words have been translated (\( 2^N \) possibilities)
  - \( r \) is the end-point of the last phrase in the hypothesis
Define $\mathcal{Y}'$ to be the set of derivations such that:

- $\sum_{i=1}^{N} y(i) = N$
- For each pair of consecutive phrases $p_k, p_{k+1}$ for $k = 1 \ldots L - 1$, we have $\delta(t(p_k), s(p_{k+1})) \leq d$, where $d$ is the distortion limit.

Notes:

- We have dropped the $y(i) = 1$ constraints.
- We have $\mathcal{Y} \subset \mathcal{Y}'$
Dynamic Programming over $\mathcal{Y}'$

- We can find $\arg \max_{y \in \mathcal{Y}'} f(y)$ efficiently, using dynamic programming.

- Dynamic programming states are of the form $(w_1, w_2, n, r)$ where
  - $w_1, w_2$ are last two words of a hypothesis
  - $n$ is the length of the partial hypothesis
  - $r$ is the end-point of the last phrase in the hypothesis
The original decoding problem is
\[ \arg \max_{y \in \mathcal{Y}} f(y) \]

We can rewrite this as
\[ \arg \max_{y \in \mathcal{Y}'} f(y) \text{ such that } \forall i, \ y(i) = 1 \]

We deal with the \( y(i) = 1 \) constraints using Lagrangian relaxation.
The Lagrangian is

\[ L(u, y) = f(y) + \sum_{i} u(i)(y(i) - 1) \]

The dual objective is then

\[ L(u) = \max_{y \in Y'} L(u, y). \]

and the dual problem is to solve

\[ \min_{u} L(u). \]
The Algorithm

Initialization: $u^0(i) \leftarrow 0$ for $i = 1 \ldots N$

for $t = 1 \ldots T$

$y^t = \arg\max_{y \in Y'} L(u^{t-1}, y)$

if $y^t(i) = 1$ for $i = 1 \ldots N$

return $y^t$

else

for $i = 1 \ldots N$

$u^t(i) = u^{t-1}(i) - \alpha^t (y^t(i) - 1)$

Figure: The decoding algorithm. $\alpha^t > 0$ is the step size at the $t$'th iteration.
An Example Run of the Algorithm

<table>
<thead>
<tr>
<th>$t$</th>
<th>$L(u^{t-1})$</th>
<th>$y^t(i)$</th>
<th>derivation $y^t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-10.0988</td>
<td>0 0 2 3 3 0 0 2 0 0 0 1</td>
<td>3, 6 the quality and 9, 9 also the regular</td>
</tr>
<tr>
<td>2</td>
<td>-11.1597</td>
<td>0 0 1 0 0 0 1 0 0 4 1 5 1</td>
<td>3, 7 7 12, 12 will continue to 12, 12</td>
</tr>
<tr>
<td>3</td>
<td>-12.3742</td>
<td>3 3 1 2 2 0 0 0 1 0 0 0 1</td>
<td>1, 2 in that way , 5, 5 and 2, 2 can thus 4, 4 quality in that way , 1, 2 the quality and 4, 4</td>
</tr>
<tr>
<td>4</td>
<td>-11.8623</td>
<td>0 1 0 0 0 1 1 3 3 0 3 0 1</td>
<td>2, 2 can the regular 8, 8 distribution should 9, 9 also ensure 8, 8 distribution should 9, 9 also ensure 8, 8 distribution should 9, 9 also ensure 13, 13</td>
</tr>
<tr>
<td>5</td>
<td>-13.9916</td>
<td>0 0 1 1 3 2 4 0 0 0 1 0 1</td>
<td>3, 7 7 5, 5 and 7, 7 and 4, 4 the quality and the regular 9, 9 also ensure</td>
</tr>
<tr>
<td>6</td>
<td>-15.6558</td>
<td>1 1 2 0 2 0 1 1 1 1 1 1 1</td>
<td>1, 2 in that way , 3, 4 the quality of 6, 6 the quality of 6, 6 the distribution should 8, 8 continue to be guaranteed</td>
</tr>
<tr>
<td>7</td>
<td>-16.1022</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
<td>1, 2 in that way , 3, 4 and the regular 8, 8 distribution should 9, 10 continue to be guaranteed</td>
</tr>
</tbody>
</table>

**Input German:** dadurch können die Qualität und die regelmäßige Postzustellung auch weiterhin sichergestellt werden.

The second theorem states that under an appropriate choice of the step sizes $\alpha_t$, the method converges.
Tightening the Relaxation

- In some cases, the relaxation is not tight, and the algorithm will not converge to \( y(i) = 1 \) for \( i = 1 \ldots N \).
- Our solution: incrementally add *hard constraints* until the relaxation is tight.
- Definition: for any set \( C \subseteq \{1, 2, \ldots, N\} \),
  \[
  \mathcal{Y}_C' = \{ y : y \in \mathcal{Y}', \text{ and } \forall i \in C, y(i) = 1 \}
  \]
- We can find \( \arg \max_{y \in \mathcal{Y}_C'} f(y) \) using dynamic programming, with a \( 2^{|C|} \) increase in the number of states.
- Goal: find a small set \( C \) such that Lagrangian relaxation with \( \mathcal{Y}_C' \) returns an exact solution.
An Example Run of the Algorithm

<table>
<thead>
<tr>
<th>( t )</th>
<th>( L(u^{t-1}) )</th>
<th>( y^t(i) )</th>
<th>( \text{derivation } y^t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-11.8658 0000130334110001</td>
<td>5, 6 10, 10 ( \text{that is a country that} )</td>
<td>8, 9 6, 6 ( \text{that} ) 10, 10 ( \text{is a country that} ) 8, 9 6, 6 ( \text{that} ) 10, 10 ( \text{is} ) 8, 8 9, 12 ( \text{a country that} ) 17, 17</td>
</tr>
<tr>
<td>2</td>
<td>-5.46647 2240201000101111</td>
<td>3, 3 1, 1 ( \text{however, it is} ) 2, 3 5, 5 ( \text{however,} ) 3, 3 1, 1 ( \text{however, it is, however,} ) 2, 3 5, 5 ( \text{7, 7} ) 11, 11 ( \text{must be closely monitored} )</td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>32</td>
<td>-17.0203 1111011112111111</td>
<td>1, 5 ( \text{nonetheless,} ) 7, 7 ( \text{colombia} ) 10, 10 8, 8 9, 12 ( \text{a country that} ) 16, 16 ( \text{must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>-17.1727 111121101111111</td>
<td>1, 5 ( \text{nonetheless,} ) 6, 6 8, 9 6, 6 7, 7 11, 12 ( \text{which must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>-17.0203 1111011112111111</td>
<td>1, 5 ( \text{nonetheless,} ) 7, 7 ( \text{colombia} ) 10, 10 8, 8 9, 12 ( \text{a country that} ) 16, 16 ( \text{must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>-17.1631 1111011121111111</td>
<td>1, 5 ( \text{nonetheless,} ) 7, 7 ( \text{colombia} ) 10, 10 8, 8 9, 12 ( \text{must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>-17.0408 111121101111111</td>
<td>1, 5 ( \text{nonetheless,} ) 6, 6 8, 9 6, 6 7, 7 11, 12 ( \text{which must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>-17.1727 1111011111211111</td>
<td>1, 5 ( \text{nonetheless,} ) 7, 7 ( \text{colombia} ) 10, 10 8, 8 9, 12 ( \text{must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>-17.0408 111121110111111</td>
<td>1, 5 ( \text{nonetheless,} ) 6, 6 8, 9 6, 6 7, 7 11, 12 ( \text{which must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>-17.1658 111121101111111</td>
<td>1, 5 ( \text{nonetheless,} ) 6, 6 8, 9 6, 6 7, 7 11, 12 ( \text{which must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>-17.056 1111101112111111</td>
<td>1, 5 ( \text{nonetheless,} ) 7, 7 ( \text{colombia} ) 10, 10 8, 8 9, 12 ( \text{must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>-17.1732 111121110111111</td>
<td>1, 5 ( \text{nonetheless,} ) 6, 6 8, 9 6, 6 7, 7 11, 12 ( \text{must be closely monitored} ) 17, 17</td>
<td></td>
</tr>
</tbody>
</table>

Input German: es bleibt jedoch dabei, dass kolumbien ein land ist, das aufmerksam beobachtet werden muss.

\( \alpha_t = \frac{1}{1 + \lambda_t} \), where \( \lambda_t \) is the number of times that \( L(u(t')) > L(u(t' - 1)) \) for all \( t' \leq t \).

\( \beta_s(u) = \emptyset \) at iteration \( K_t(i) = 1 \).

Adding constraints: 6 10

\( \text{count}(6) = 10; \text{count}(10) = 10; \text{count}(i) = 0 \) for all other \( i \)

Running times presented in Table 2.
The Algorithm with Constraint Generation

\textit{Optimize}(\mathcal{C}, u)

\textbf{while} (dual value still improving)

\[ y^* = \arg\max_{y \in \mathcal{Y}_C} L(u, y) \]

\textbf{if} \( y^*(i) = 1 \) for \( i = 1 \ldots N \) \textbf{return} \( y^* \)

\textbf{else for} \( i = 1 \ldots N \)

\[ u(i) = u(i) - \alpha (y^*(i) - 1) \]

\[ \text{count}(i) = 0 \text{ for } i = 1 \ldots N \]

\textbf{for} \( k = 1 \ldots K \)

\[ y^* = \arg\max_{y \in \mathcal{Y}_C} L(u, y) \]

\textbf{if} \( y^*(i) = 1 \) for \( i = 1 \ldots N \) \textbf{return} \( y^* \)

\textbf{else for} \( i = 1 \ldots N \)

\[ u(i) = u(i) - \alpha (y^*(i) - 1) \]

\[ \text{count}(i) = \text{count}(i) + \left\lfloor y^*(i) \neq 1 \right\rfloor \]

Let \( \mathcal{C}' = \text{set of G i's that have the largest value for count(i) and that are not in C} \)

\textbf{return} \textit{Optimize}(\mathcal{C} \cup \mathcal{C}', u)
Number of Constraints Required

Table 2: Table showing the number of constraints added before convergence of the algorithm in Figure 3, broken down by sentence length. Note that a maximum of 3 constraints are added at each recursive call, but that fewer than 3 constraints are added in cases where fewer than 3 constraints have $count(i) > 0$. x indicates the sentences that fail to converge after 250 iterations. 78.7% of the examples converge without adding any constraints.

<table>
<thead>
<tr>
<th># cons.</th>
<th>1-10 words</th>
<th>11-20 words</th>
<th>21-30 words</th>
<th>31-40 words</th>
<th>41-50 words</th>
<th>All sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-0</td>
<td>183 (98.9%)</td>
<td>511 (91.6%)</td>
<td>438 (77.4%)</td>
<td>222 (64.0%)</td>
<td>82 (48.8%)</td>
<td>1,436 (78.7%)</td>
</tr>
<tr>
<td>1-3</td>
<td>2 (1.1%)</td>
<td>45 (8.1%)</td>
<td>94 (16.6%)</td>
<td>87 (25.1%)</td>
<td>50 (29.8%)</td>
<td>278 (15.2%)</td>
</tr>
<tr>
<td>4-6</td>
<td>0 (0.0%)</td>
<td>2 (0.4%)</td>
<td>27 (4.8%)</td>
<td>24 (6.9%)</td>
<td>19 (11.3%)</td>
<td>72 (3.9%)</td>
</tr>
<tr>
<td>7-9</td>
<td>0 (0.0%)</td>
<td>0 (0.0%)</td>
<td>7 (1.2%)</td>
<td>13 (3.7%)</td>
<td>12 (7.1%)</td>
<td>32 (1.8%)</td>
</tr>
<tr>
<td>x</td>
<td>0 (0.0%)</td>
<td>0 (0.0%)</td>
<td>0 (0.0%)</td>
<td>1 (0.3%)</td>
<td>5 (3.0%)</td>
<td>6 (0.3%)</td>
</tr>
</tbody>
</table>

Table 3: The average time (in seconds) for decoding using the algorithm in Figure 3, with and without A* algorithm, broken down by sentence length and the number of constraints that are added. A* indicates speeding up using A* search; w/o denotes without using A*.

6.1 Comparison to an LP/ILP solver

a general purpose integer linear programming (ILP) solver, which solves the problem exactly.

The experiments focus on translation from German to English, using the Europarl data (Koehn, 2005). We tested on 1,824 sentences of length at most 50 words. The experiments use the algorithm shown in Figure 3. We limit the algorithm to a maximum of 250 iterations and a maximum of 9 hard constraints. The distortion limit $d$ is set to be four, English phrases per German phrase.

To compare to a linear programming (LP) or integer linear programming (ILP) solver, we can implement the search over the set $Y$ through linear relaxation. For sentences with 1-10 words, the vast majority (183 out of 185 examples) converge with 0 constraints added. As sentences get longer, more constraints are often required. However most examples converge without adding hard constraints to tighten the grammar smaller. Table 4 includes results of the time required to find a solution on sentences of length 1-15.

The algorithm makes use of states $(w, n, r)$—i.e., the span $l, m$ is dropped, making the dynamic programming algorithm smaller. Table 4 includes results of the time required to find a solution on sentences of length 1-15. 0% constraints required, increase. The average run time across all sentences is 120.9 seconds.

The experiments use the algorithm shown in Figure 3. We limit the algorithm to a maximum of 250 iterations and a maximum of 9 hard constraints. The distortion limit $d$ is set to be four, German phrases per English phrase.

Table 4 shows the run time of the method without the A* algorithm for decoding. The A* algorithm gives significant reductions in runtime. The method finds exact solutions on 1,818 out of 1,824 sentences (99.67%). (6 examples do not converge within 250 iterations.) Table 1 shows the number of constraints required, increase. The average run time across all sentences is 120.9 seconds.

In 1,436/1,818 (78.7%) sentences, the method converges without adding hard constraints to tighten the grammar smaller. Table 4 includes results of the time required to find a solution on sentences of length 1-15. 0% constraints required, increase. The average run time across all sentences is 120.9 seconds.

Our method finds exact solutions on 1,818 out of 1,824 sentences (99.67%). (6 examples do not converge within 250 iterations.) Table 1 shows the number of constraints required, increase. The average run time across all sentences is 120.9 seconds.

In 1,436/1,818 (78.7%) sentences, the method converges without adding hard constraints to tighten the grammar smaller. Table 4 includes results of the time required to find a solution on sentences of length 1-15. 0% constraints required, increase. The average run time across all sentences is 120.9 seconds.
Table 2: Table showing the number of constraints added before convergence of the algorithm in Figure 3, broken down by sentence length. Note that a maximum of 3 constraints are added at each recursive call, but that fewer than 3 constraints are added in cases where fewer than 3 constraints have count \((i) > 0\). x indicates the sentences that fail to converge after 250 iterations.

Table 3: The average time (in seconds) for decoding using the algorithm in Figure 3, with and without A* algorithm, broken down by sentence length and the number of constraints that are added. A* indicates speeding up using A* search; w/o denotes without using A*.
Comparison to LP/ILP Decoding

<table>
<thead>
<tr>
<th>method</th>
<th>ILP</th>
<th>LP</th>
<th>% frac.</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>length</td>
<td>mean</td>
<td>median</td>
</tr>
<tr>
<td>$\mathcal{Y}'$</td>
<td>1-10</td>
<td>275.2</td>
<td>132.9</td>
</tr>
<tr>
<td></td>
<td>11-15</td>
<td>2,707.8</td>
<td>1,138.5</td>
</tr>
<tr>
<td></td>
<td>16-20</td>
<td>20,583.1</td>
<td>3,692.6</td>
</tr>
<tr>
<td>$\mathcal{Y}''$</td>
<td>1-10</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 4: Average and median time of the LP/ILP solver (in seconds). % frac. indicates how often the LP gives a fractional answer. $\mathcal{Y}'$ indicates the dynamic program using set $\mathcal{Y}'$ as defined in Section 4.1, and $\mathcal{Y}''$ indicates the dynamic program using states $(w_1, w_2, n, r)$. The statistics for ILP for length 16-20 is based on 50 sentences.
Number of Iterations Required

<table>
<thead>
<tr>
<th># iter.</th>
<th>1-10 words</th>
<th>11-20 words</th>
<th>21-30 words</th>
<th>31-40 words</th>
<th>41-50 words</th>
<th>All sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>166 (89.7%)</td>
<td>219 (39.2%)</td>
<td>34 (6.0%)</td>
<td>2 (0.6%)</td>
<td>0 (0.0%)</td>
<td>421 (23.1%)</td>
</tr>
<tr>
<td>8-15</td>
<td>17 (9.2%)</td>
<td>187 (33.5%)</td>
<td>161 (28.4%)</td>
<td>30 (8.6%)</td>
<td>3 (1.8%)</td>
<td>398 (21.8%)</td>
</tr>
<tr>
<td>16-30</td>
<td>1 (0.5%)</td>
<td>93 (16.7%)</td>
<td>208 (36.7%)</td>
<td>112 (32.3%)</td>
<td>22 (13.1%)</td>
<td>436 (23.9%)</td>
</tr>
<tr>
<td>31-60</td>
<td>1 (0.5%)</td>
<td>52 (9.3%)</td>
<td>105 (18.6%)</td>
<td>99 (28.5%)</td>
<td>62 (36.9%)</td>
<td>319 (17.5%)</td>
</tr>
<tr>
<td>61-120</td>
<td>0 (0.0%)</td>
<td>7 (1.3%)</td>
<td>54 (9.5%)</td>
<td>89 (25.6%)</td>
<td>45 (26.8%)</td>
<td>195 (10.7%)</td>
</tr>
<tr>
<td>121-250</td>
<td>0 (0.0%)</td>
<td>0 (0.0%)</td>
<td>4 (0.7%)</td>
<td>14 (4.0%)</td>
<td>31 (18.5%)</td>
<td>49 (2.7%)</td>
</tr>
<tr>
<td>x</td>
<td>0 (0.0%)</td>
<td>0 (0.0%)</td>
<td>0 (0.0%)</td>
<td>1 (0.3%)</td>
<td>5 (3.0%)</td>
<td>6 (0.3%)</td>
</tr>
</tbody>
</table>

Table 1: Table showing the number of iterations taken for the algorithm to converge. x indicates sentences that fail to converge after 250 iterations. Of the examples converge within 120 iterations.
presented dual decomposition as a method for decoding in NLP

**formal guarantees**
- gives certificate or approximate solution
- can improve approximate solutions by tightening relaxation

**efficient algorithms**
- uses fast combinatorial algorithms
- can improve speed with lazy decoding

**widely applicable**
- demonstrated algorithms for a wide range of NLP tasks (parsing, tagging, alignment, mt decoding)
References I


References II


References III


