The Cost of Model Complexity

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

Tradeoff: Richer models ⇒ Harder decoding

Added complexity is both computational and implementational.

Tasks with challenging decoding problems:
- Speech Recognition
- Sequence Modeling (e.g. extensions to HMM/CRF)
- Parsing
- Machine Translation

\[ y^* = \arg \max_y f(y) \quad \text{Decoding} \]

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

\[ *_0 \xrightarrow{\text{John}} \xrightarrow{\text{saw}} \xrightarrow{\text{a}} \xrightarrow{\text{movie}} \xrightarrow{\text{today}} \xrightarrow{\text{that}} \xrightarrow{\text{he}} \xrightarrow{\text{liked}} _8 \]

Important problem in many languages.

Problem is NP-Hard for all but the simplest models.
A Dual Decomposition Algorithm for Non-Projective Dependency Parsing

Simple - Uses basic combinatorial algorithms

Efficient - Faster than previously proposed algorithms

Strong Guarantees - Gives a certificate of optimality when exact

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

Widely Applicable - Similar techniques extend to other problems

Roadmap

Algorithm

Experiments

Derivation

Non-Projective Dependency Parsing

*0 John1 saw2 a3 movie4 today5 that6 he7 liked8

▶ Starts at the root symbol *

▶ Each word has exactly one parent word

▶ Produces a tree structure (no cycles)

▶ Dependencies can cross

Algorithm Outline

Arc-Factored Model

Dual Decomposition

Sibling 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)

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

Thought Experiment: Individual Decoding

2

\[ \text{possibilities} \]

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

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

Sibling Models

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

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

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

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

Thought Experiment Continued

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

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

But we might violate some constraints.
### Dual Decomposition Idea

<table>
<thead>
<tr>
<th>Dual Decomposition Structure</th>
<th>Algorithm Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong> $y^* = \arg \max_{y \in \mathcal{Y}} f(y)$</td>
<td>Set penalty weights equal to 0 for all edges.</td>
</tr>
<tr>
<td><strong>Rewrite as</strong> $\max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f(z) + g(y)$</td>
<td><strong>For</strong> $k = 1$ to $K$</td>
</tr>
<tr>
<td>such that $z = y$</td>
<td>$z^{(k)} \leftarrow$ Decode $(f(z) + \text{penalty})$ by Individual Decoding</td>
</tr>
<tr>
<td></td>
<td>$y^{(k)} \leftarrow$ Decode $(g(y) - \text{penalty})$ by Minimum Spanning Tree</td>
</tr>
<tr>
<td><strong>Set penalty weights equal to $V$ for all edges $T$</strong></td>
<td><strong>If</strong> $y^{(k)}(i,j) = z^{(k)}(i,j)$ for all $i,j$ <strong>Return</strong> $(y^{(k)}, z^{(k)})$</td>
</tr>
<tr>
<td><strong>Arc-Factored</strong></td>
<td><strong>Else</strong> Update penalty weights based on $y^{(k)}(i,j) - z^{(k)}(i,j)$</td>
</tr>
<tr>
<td><strong>Minimum Spanning Tree</strong></td>
<td><strong>Converged</strong> $y^* = \arg \max_{y \in \mathcal{Y}} f(y) + g(y)$</td>
</tr>
<tr>
<td><strong>Sibling Model</strong></td>
<td><strong>Key</strong></td>
</tr>
<tr>
<td><strong>Individual Decoding</strong></td>
<td>$f(z) \iff$ Sibling Model $g(y) \iff$ Arc-Factored Model</td>
</tr>
<tr>
<td>$z^* = \arg \max_{z \in \mathcal{Z}} (f(z) + \sum_{i,j} u(i,j)z(i,j))$</td>
<td>$\mathcal{Z} \iff$ No Constraints $\mathcal{Y} \iff$ Tree Constraints</td>
</tr>
<tr>
<td><strong>Minimum Spanning Tree</strong></td>
<td>$y(i,j) = 1$ if $y$ contains dependency $i,j$</td>
</tr>
<tr>
<td>$y^* = \arg \max_{y \in \mathcal{Y}} (g(y) - \sum_{i,j} u(i,j)y(i,j))$</td>
<td><strong>Penalties</strong></td>
</tr>
</tbody>
</table>

| $u(i,j) = 0$ for all $i,j$ | Iteration 1 |
| $u(8,1)$ | -1 |
| $u(4,6)$ | -1 |
| $u(2,6)$ | 1 |
| $u(8,7)$ | 1 |
| **Iteration 2** | |
| $u(8,1)$ | -1 |
| $u(4,6)$ | -2 |
| $u(2,6)$ | 2 |
| $u(8,7)$ | 1 |
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).

---

Roadmap

**Algorithm**

**Experiments**

**Derivation**

---

Extensions

- **Grandparent Models**

![Diagram of Grandparent Models]

$$f(y) = \ldots + \text{score}(gp=s_0, \ head = \text{soc}_2, \ prev = \text{movie}_4, \ mod = \text{today}_5)$$

- **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.

Parsing Speed

- Number of sentences parsed per second
- Comparable to dynamic programming for projective parsing

Accuracy

<table>
<thead>
<tr>
<th>Language</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

F₁ 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

Roadmap

Algorithm
Experiments
Derivation
Deriving the Algorithm

Goal:

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

Rewrite:

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

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

Lagrangian:

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

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

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

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

A Subgradient Algorithm for Minimizing \( L(u) \)

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

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

\[ L(v) \geq L(u) + g_u \cdot (v - u) \]

Subgradient methods use updates \( u' = u - \alpha g_u \)

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

Related Work

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

Summary

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

Arc-Factored Model

Dual Decomposition

Sibling Model
Other Applications

- Dual decomposition can be applied to other decoding problems.
- Rush et al. (2010) focuses on integrated dynamic programming algorithms.
  - Integrated Parsing and Tagging
  - Integrated Constituency and Dependency Parsing

Dependency and Constituency

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

Future Directions

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

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

- Possible NLP Applications
  - Machine Translation
  - Speech Recognition
  - "Loopy" Sequence Models

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

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

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

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

- (More details in the paper.)