Outline

• Introduction

• Dual Decomposition

• Incremental ILP

• Amortizing Inference
What is Inference

- NLP problems, map input $x$ to structured output $y$
  
  $y^* = \underset{y \in Y}{\text{argmax}} h(y)$
  
  - $Y$ is the set of output for $x$
  - $h$ is a scoring function assigning score to the output
  - It can be learned using any learning model

- POS Tagging: $x$ is sentence and $Y$ set of all possible tag sequences

- Parsing: $x$ is sentence and $Y$ set of all parse trees

- Solving the inference problem
  - LP / ILP solvers
  - Dynamic Programming
Difficulty in Inference

- Search space grows exponentially with $x$
- Problem becomes intractable / polynomial but very slow

- 3 different approaches:
  - Dual Decomposition
  - Incremental ILP (cutting plane)
  - Amortized Inference
Lagrangian Relaxation

• Original Problem: \( \text{argmax}_{y \in Y} y \cdot \theta \)

• Choose a set \( Y' \subset R^d \), s.t
  - \( Y \subset Y' \)
  - For any \( \theta \in R^d \), \( \text{argmax}_{y \in Y'} y \cdot \theta \), is easily computable
  - Let \( Y = \{ y : y \in Y', Ay = b \} \)

\[ L(u, y) = y \cdot \theta + u(Ay - b) \]

• Dual Objective: \( L(u) = \max_{y \in Y'} L(u, y) \)

• For any \( u \), \( L(u) \geq \max_{y \in Y'} h(y) \)

• Dual Problem: \( \min_u L(u) \)

Makes the problem NP Hard

Lagrangian Multiplier

Linear Constraint

Use subgradient Algorithm
Dual Decomposition

- Special case of LR
- Decoding problem: $\arg\max_{x \in Y, z \in Z} y \cdot \theta^{(1)} + z \cdot \theta^{(2)}$
  \[ s.t \quad Ay + Cz = b \]
- $W = \{y \in Y, z \in Z, Ay + Cz = b\}, W' = \{y \in Y, z \in Z\}$
- Clearly the problem is similar to LR
- Lagrangian: $L(u, y, z) = y \cdot \theta^{(1)} + z \cdot \theta^{(2)} + u(Ay + Cz - b)$
  - Dual Objective: $L(u) = \max_{(y, z) \in W'} L(u, y, z)$
  - Dual Problem: $\min_u L(u)$
- $L(u)$ is convex, but may not differentiable.
- Iteratively solved using subgradient descent
Dual Decomposition

- Initialize the lagrangian multiplier $u$ to 0
- In each $k^{th}$ iteration find the structure $y^{(k)}, z^{(k)}$
  - If $Ay^{(k)} + Cz^k = b$, then return $y^{(k)}, z^{(k)}$ as soln
    - Will be optimal solution to original decoding problem
  - Else update the multiplier $u$

- DD problem is in fact dual of LP relaxation of original decoding problem (Primal)
  - If this is tight, then optimal soln to original problem always found
Non-Projective Dependency Parsing

- Edge \((i,j)\) indicates \(i\) : head-word, \(j\) : modifier
- Forms a directed spanning tree rooted at “root”
- Non-Projective : Dependency edges can cross
- \(Y\) : Set of well-formed dependency parsers
- optimal parse tree : \(y^* = \text{argmax}_{y \in Y} f(y)\)

\(f\) : Assigns scores to parse trees
Arc-Factored Model

- $f(y) = \text{Sum of Dependency Scores}$
  
  $= \text{score}(\text{root, saw}) + \text{score}(\text{saw, John}) + \text{score}(\text{saw, bird}) + \cdots$

- Scores (e.g. $\text{score}(\text{saw, John})$) can be obtained using any learning method

- Optimal Parser found using simple MST algorithm
Sibling Model

- $y_i =$ Set of modifiers of headword $i$
- $f(y) = \sum_i f_i(y_i)$
  (where each $f_i$ computes sum of pairwise dependency)
  
  $= \text{score(root, NULL, saw)} + \text{score(saw, NULL, John)} + \text{score(saw, NULL, bird)} + \text{score(saw, bird, yesterday)} + \cdots$

- Again individual scores obtained using any learning
- $y^* = \arg\max_{y \in Y} \sum_i f_i(y_i)$ is NP-Hard

$Y : \text{Set of well-formed parse trees}$
Optimal Modifiers for each Head-word

- Easy to find set of modifier which maximizes $f_i(y_i)$
- $2^{n-1}$ possible choices for each head-word
- Can be solved using dynamic programming in $O(n^2)$
A Simple Algorithm

• Find the optimal set of modifiers for each word
• If it’s a valid parse tree we are done !!
• Resulting parser may not be well formed
  • May contain cycles !
• Computes $z^* = \arg \max_{z \in Z} f(z) = \arg \max_{z \in Z} f_i(z_i)$

$Z$ : Set of graphs (Relaxed from set of valid parse trees)
Consider a more generalized model:

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

Rewrite as:

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

s.t. \[ z = y \]

Without constraint Obj Fn:

\[ z^* = \arg \max_{z \in Z} f(z), \quad y^* = \arg \max_{y \in Y} h(y) \]
Use Lagrangian multiplier and move constraint to obj fn
\[ L(u, y, z) = f(z) + h(y) + u(y - z) \]
\[ L^* = \max_{\{z \in Z, y \in Y, y = z\}} L(u, y, z) \]
Dual Objective fn: \[ L(u) = \max_{\{z \in Z, y \in Y\}} L(u, y, z) \]
\[ = \max_{\{z \in Z\}} f(z) - u(z) + \max_{\{y \in Y\}} h(y) + u(y) \]
\[ L^* \leq L(u) \]
Dual Problem: min \[ L(u) \]
Use subgradient algorithm
Algorithm Outline

- Set $u^{(1)} = 0$
- For $k = 1$ to $K$ do
  
  $$ z^{(k)} = \arg\max_{\{z \in Z\}} f(z) - u^{(k)}(z) \text{ (By Individual Decoding)} $$

  $$ y^{(k)} = \arg\max_{\{y \in Y\}} h(y) + u^k(y) \text{ (By MST)} $$

  if ($y^{(k)} = z^{(k)}$)
    
    return $y^{(k)}, z^{(k)}$

  else
    
    update $u^{(k+1)} = u^{(k)} + \alpha_k (z^{(k)} - y^k)$
## Comparison LP/ILP

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Integral Solution</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP(M)</td>
<td>92.17</td>
<td>93.18</td>
<td>0.58</td>
</tr>
<tr>
<td>ILP (Gurobi)</td>
<td>92.19</td>
<td>100</td>
<td>1.44</td>
</tr>
<tr>
<td>DD (K=5000)</td>
<td>92.19</td>
<td>98.82</td>
<td>0.08</td>
</tr>
<tr>
<td>DD (K = 250)</td>
<td>92.23</td>
<td>89.29</td>
<td>0.03</td>
</tr>
</tbody>
</table>
A Tutorial on Dual Decomposition and Lagrangian Relaxation for Inference in Natural Language Processing, Alexander M. Rush and Michael Collins, JAIR’13

INFERENCE

Part 2
Incremental Integer Linear Programming

1. Many problems and models can be reformulated into ILP ex. HMM

2. Incorporate more constraints
Non-projective Dependency Parsing

"come" is a Head
"I'll" is a Child of "come"
"subject" is the Label between "come" and "I'll"
Non-projective: Dependency can cross
Model

Object function

\[ s(x, y) = \sum_{(i,j,l) \in y} s(i, j, l) = \sum_{(i,j,l) \in y} w \cdot f(i, j, l) \]

\( x \) is sentence, \( y \) is a set of labelled dependencies, \( f(i,j,l) \) is feature for token \( i, j \) with label \( l \)

\[ y' = \arg \max_{y} s(x, y) \]
Constraints

T1: For every non-root token in x there exists exactly one head; the root token has no head.

T2: There are no cycles

A1: Head are not allowed to have more than one outgoing edge labeled l for all l in a set of labels U

C1: In a symmetric coordination there is exactly one argument to the right of the conjunction and at least one argument to the left

C4: Arguments of a coordination must have compatible word classes.

P1: Two dependencies must not cross if one of their labels is in a set of labels P.
Reformulate into ILP

Labelled edges:

\[ e_{i,j,l} \quad \forall i \in 0..n, j \in 1..n, \quad \text{such that } l \in \text{best}_k(i,j) \]

Existence of a dependency between tokens i and j:

\[ d_{i,j} = \sum_{l \in \text{best}_k(i,j)} e_{i,j,l} \]

Objective function:

\[ \sum_{i,j} \sum_{l \in \text{best}_k(i,j)} s(i, j, l) \cdot e_{i,j,l} \]
Reformulate constraints

Only one head (T1):

\[ \sum_i d_{i,j} = 1 \]

Label Uniqueness (A1):

\[ \sum_j e_{i,j,l} \leq 1 \]

Symmetric Coordination (C1):

\[ \sum_{j<i} d_{i,j} \geq 1 \quad \sum_{j>i} d_{i,j} = 1 \]

No Cycles (T2):

\[ \sum_{(i,j) \in e} d_{i,j} \leq |c| - 1 \]
Algorithm

$B_x$: Constraints added in advance
$O_x$: Objective function
$V_x$: Variables
$W$: Violated constraints

Algorithm 1 Incremental Integer Linear Programming

\[
\begin{align*}
C & \leftarrow B_x \\
\text{repeat} & \\
\text{y} & \leftarrow \text{solve}(C, O_x, V_x) \\
W & \leftarrow \text{violated}(y, I_x) \\
C & \leftarrow C \cup W \\
\text{until } V = \emptyset & \\
\text{return } y
\end{align*}
\]
## Experiment results

<table>
<thead>
<tr>
<th></th>
<th>LAC</th>
<th>UAC</th>
<th>LC</th>
<th>UC</th>
</tr>
</thead>
<tbody>
<tr>
<td>bl</td>
<td>84.6%</td>
<td>88.9%</td>
<td>27.7%</td>
<td>42.2%</td>
</tr>
<tr>
<td>cnstr</td>
<td>85.1%</td>
<td>89.4%</td>
<td>29.7%</td>
<td>43.8%</td>
</tr>
</tbody>
</table>

**LAC**: Labelled accuracy  
**UAC**: Unlabelled accuracy  
**LC**: Percentage of sentences with 100% labelled accuracy  
**UC**: Percentage of sentences with 100% unlabelled accuracy
# Runtime Evaluation

<table>
<thead>
<tr>
<th>Tokens</th>
<th>1-10</th>
<th>11-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
<th>&gt;50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>5242</td>
<td>4037</td>
<td>1835</td>
<td>650</td>
<td>191</td>
<td>60</td>
</tr>
<tr>
<td>Avg. ST(bl)</td>
<td>0.27ms</td>
<td>0.98ms</td>
<td>3.2ms</td>
<td>7.5ms</td>
<td>14ms</td>
<td>23ms</td>
</tr>
<tr>
<td>Avg. ST(cnstr)</td>
<td>5.6ms</td>
<td>52ms</td>
<td>460ms</td>
<td>1.5s</td>
<td>7.2s</td>
<td>33s</td>
</tr>
</tbody>
</table>
Amortizing Inference Algorithm

Idea: Under some conditions, we can re-use earlier solutions for certain instances

Why?

1. Only a small fraction of structure occurs, compared with large space of possible structures
2. The distribution of observed structures is heavily skewed towards a small number of them

# of Examples >> # of ILPs >> # of Solutions
Amortizing Inference Conditions

Considering solving a 0-1 ILP problem

Idea of Theorem 1:
For every inference variable that is active in the solution, increasing the corresponding objective value will not change the optimal assignment to the variables.

For variable whose value in solution is 0, decreasing the objective value will not change the optimal solution.
Theorem 1. Let $p$ denote an inference problem posed as an integer linear program belonging to an equivalence class $[P]$, and $q \sim [P]$ be another inference instance in the same equivalence class. Define $d(c) = c_q - c_p$ to be the difference of the objective coefficients of the ILPs. Then $y_p$ is the solution of problem $q$ if for each $i$ in $\{1, \ldots, n_p\}$, we have

$$(2y_{p,i} - 1)d(c_i) > 0$$
Amortizing Inference Conditions

Idea of Theorem 2:
Suppose \( y^* \) is optimal solution,
\[
    c_p y \leq c_p y^*, \quad c_q y \leq c_q y^*
\]
Then
\[
    (x_1 c_p + x_2 c_q) y \leq (x_1 c_p + x_2 c_q) y^*
\]

Theorem 2:
\( y^* \) is solution which has objective coefficients
\( (x_1 c_p + x_2 c_q) \)
Amortizing Inference Conditions

Theorem 3:
Define $D(c,x) = c_q - \sum(x_jc_{p,j})$, if there is some $x$ s.t. $x$ positive and for any $i$

$$(2y_{p,i}-1)D(c,x) \geq 0$$

then $q$ has the same optimal solution.
## Conditions

<table>
<thead>
<tr>
<th>Theorem</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theorem 1</td>
<td>$\forall i \in {1, \cdots, n_p}$, $(2y_{p,i} - 1)\delta c_i \geq 0$; $\forall i$.</td>
</tr>
<tr>
<td>Theorem 2</td>
<td>$\exists \mathbf{x} \in \mathbb{R}^m$, such that $\mathbf{x} \geq 0$ and $c_q = \sum_j x_j c^j_p$</td>
</tr>
<tr>
<td>Theorem 3</td>
<td>$\exists \mathbf{x} \in \mathbb{R}^m$, such that $\mathbf{x} \geq 0$ and $(2y_{p,i} - 1) \Delta c_i \geq 0$; $\forall i$.</td>
</tr>
</tbody>
</table>
Approximation schemes

1. Most frequent solution
2. Top-K approximation
3. Approximations to theorem 1

\[(2y_{p,i} - 1)\delta c_i + \epsilon \geq 0.\]

and theorem 3

\[(2y_{p,i} - 1)\Delta c_i + \epsilon \geq 0\]
## Experiment Result

<table>
<thead>
<tr>
<th>Type</th>
<th>Algorithm</th>
<th># instances</th>
<th># solver calls</th>
<th>Speedup</th>
<th>Clock speedup</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>Baseline</td>
<td>5127</td>
<td>5217</td>
<td>1.0</td>
<td>1.0</td>
<td>75.85</td>
</tr>
<tr>
<td>Exact</td>
<td>Theorem 1</td>
<td>5127</td>
<td>2134</td>
<td>2.44</td>
<td>1.54</td>
<td>75.90</td>
</tr>
<tr>
<td>Exact</td>
<td>Theorem 2</td>
<td>5127</td>
<td>2390</td>
<td>2.18</td>
<td>1.14</td>
<td>75.79</td>
</tr>
<tr>
<td>Exact</td>
<td>Theorem 3</td>
<td>5127</td>
<td>2089</td>
<td>2.50</td>
<td>1.36</td>
<td>75.77</td>
</tr>
<tr>
<td>Approx.</td>
<td>Most frequent (Support = 50)</td>
<td>5127</td>
<td>2812</td>
<td>1.86</td>
<td>1.57</td>
<td>62.00</td>
</tr>
<tr>
<td>Approx.</td>
<td>Top-10 solutions (Support = 50)</td>
<td>5127</td>
<td>2812</td>
<td>1.86</td>
<td>1.58</td>
<td>70.06</td>
</tr>
<tr>
<td>Approx.</td>
<td>Theorem 1 (approx, $\epsilon = 0.3$)</td>
<td>5127</td>
<td>1634</td>
<td>3.19</td>
<td>1.81</td>
<td>75.76</td>
</tr>
<tr>
<td>Approx.</td>
<td>Theorem 3 (approx, $\epsilon = 0.3$)</td>
<td>5127</td>
<td>1607</td>
<td>3.25</td>
<td>1.50</td>
<td>75.46</td>
</tr>
</tbody>
</table>
Thank you!