Dynamic Feature Selection for Dependency Parsing

He He, Hal Daumé III and Jason Eisner

EMNLP 2013, Seattle
Structured Prediction in NLP

Part-of-Speech Tagging

N \rightarrow N \rightarrow V \rightarrow \text{Det} \rightarrow N
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow
Fruit flies like a banana

Machine Translation

Fruit flies like a banana

summarization, name entity resolution and many more ...

Fruit flies like a banana

$ Fruit flies like a banana
Structured Prediction in NLP

Part-of-Speech Tagging
N → N → V → Det → N
↓  ↓  ↓  ↓  ↓
Fruit flies like a banana

Machine Translation
Fruit flies like a banana.
果 蝇 喜欢 香蕉。

summarization, name entity resolution and many more ...

Exponentially increasing search space
Millions of features for scoring
Structured Prediction in NLP

Fruit     flies      like         a       banana

$\text{Fruit} \rightarrow \text{flies} \rightarrow \text{like} \rightarrow \text{a} \rightarrow \text{banana}$
Fruit flies like a banana

Feature templates per edge
- token
- left token
- right token
- in-between token
- stem
- form
- bigram
- tag
- coarse tag
- length
- direction
- ...

Structured Prediction in NLP
Structured Prediction in NLP

Fruit flies like a banana

Feature templates per edge

(token + head_token)  
(left_token + head_tag)  
(right_token + mod_token)  
(inbetween_token + mod_tag)  
(stem + tag)  
(form + coarse_tag)  
(bigram + length)  
(direction + ...)  

(head_token + mod_token)  
(X + head_tag + mod_tag)
Structured Prediction in NLP

Fruit flies like a banana

Feature templates per edge:
- token
- left token
- right token
- in-between token
- stem
- form
- bigram
- tag
- coarse tag
- length
- direction
...

(head_token + mod_token) × (head_tag + mod_tag)
Structured Prediction in NLP

Fruit     flies      like         a       banana

Do you need all features everywhere ?

Feature templates per edge

- token
- tag
- coarse tag
- length
- direction
- ...

Feature templates per edge
Structured Prediction in NLP

Fruit flies like a banana

Feature templates per edge

do you need all features everywhere?
Structured Prediction in NLP

Fruit     flies     like     a     banana

Feature templates per edge

token
tag
coarse tag
length
direction
...

Do you need all features everywhere?
Structured Prediction in NLP

Fruit flies like a banana

Feature templates per edge

Dynamic Decisions
Case Study: Dependency Parsing

2x to 6x speedup with little loss in accuracy
Graph-based Dependency Parsing

Scoring: $\phi(E) \cdot w$
Graph-based Dependency Parsing

$\text{Scoring: } \phi(E) \cdot w$

This time, the firms were ready.

<table>
<thead>
<tr>
<th>firms</th>
<th>were</th>
</tr>
</thead>
<tbody>
<tr>
<td>length:</td>
<td>1</td>
</tr>
<tr>
<td>direction:</td>
<td>right</td>
</tr>
<tr>
<td>modifier_token:</td>
<td>were</td>
</tr>
<tr>
<td>head_token:</td>
<td>firms</td>
</tr>
<tr>
<td>head_tag:</td>
<td>noun</td>
</tr>
</tbody>
</table>

And hundreds more!
Graph-based Dependency Parsing

Decoding: find the highest-scoring tree
MST Dependency Parsing
(1st-order projective)

![Graph showing mean time vs. sentence length]
MST Dependency Parsing
(1st-order projective)

Find highest-scoring tree $O(n^3)$
MST Dependency Parsing
(1st-order projective)

Find edge scores

Find highest-scoring tree $O(n^3)$

~268 feature templates
~76M features
Add features only when necessary!

score(This → ready) =

score(the → firms) =
Add features only when necessary!

$$\text{score(This → ready)} = -0.23$$

$$\text{score(the → firms)} = 0.63$$
Add features only when necessary!

score(This → ready) = -0.13
score(the → firms) = 1.33
Add features only when necessary!

This → ready: score = -0.13

the → firms: score = 1.33

WINNER

Add features only when necessary!

score(This $\rightarrow$ ready) = -1.33

score(the $\rightarrow$ firms) = 1.33
Add features only when necessary!

score(This $\rightarrow$ ready) = -1.88
score(the $\rightarrow$ firms) = 1.33
Add features only when necessary!

This is a **structured** problem! Should not look at scores independently.
Dynamic Dependency Parsing

1. Find the highest-scoring tree after adding some features *fast non-projective decoding*
Dynamic Dependency Parsing

1. Find the highest-scoring tree after adding some features *fast non-projective decoding*

2. Only edges in the current best tree can win
Dynamic Dependency Parsing

1. Find the highest-scoring tree after adding some features
   *fast non-projective decoding*

2. Only edges in the current best tree can win
   
   😊 are chosen by a classifier
   😞 are killed because they fight with the winners
Dynamic Dependency Parsing

1. Find the highest-scoring tree after adding some features. *fast non-projective decoding*

2. Only edges in the current best tree can win. 😊 are chosen by a classifier \( \leq n \) decisions 😞 are killed because they fight with the winners

3. Add features to undetermined edges *by group*
Dynamic Dependency Parsing

1. Find the highest-scoring tree after adding some features \textit{fast non-projective decoding}

2. Only edges in the current best tree can win \begin{itemize}
  \item [+] are chosen by a classifier \( \leq n \) decisions
  \item [-] are killed because they fight with the winners
\end{itemize}

3. Add features to undetermined edges \textit{by group}

Max \# of iterations = \# of feature groups
This time, the firms were ready...
This time, the firms were ready

51 gray edges with unknown fate...

5 features per gray edge

Non-projective decoding to find new 1-best tree
This time, the firms were ready. $50$ gray edges with unknown fate...

5 features per gray edge

---

Undetermined edge

Current 1-best tree

Winner edge (permanently in 1-best tree)

- Loser edge

Classifier picks \textit{winners} among the \textit{blue edges}
This time, the firms were ready.

44 gray edges with unknown fate...
5 features per gray edge

Remove losers in conflict with the winners

- Undetermined edge
- Current 1-best tree
- Winner edge (permanently in 1-best tree)
- Loser edge
This time, the firms were ready

44 gray edges with unknown fate...
5 features per gray edge

Remove losers in conflict with the winners
This time, the firms were ready.

+ next feature group

44 gray edges with unknown fate...

27 features per gray edge

- Undetermined edge

$\text{Current 1-best tree}$

$\text{Winner edge}$

(permanently in 1-best tree)

$\text{- Loser edge}$
This time, the firms were ready

+ next feature group

44 gray edges with unknown fate...

27 features per gray edge

Non-projective decoding to find new 1-best tree

- Loser edge

Current 1-best tree

Winner edge (permanently in 1-best tree)

Undetermined edge
This time, the firms were ready.

42 gray edges with unknown fate... 27 features per gray edge

Classifier picks winners among the blue edges.
This time, the firms were ready $31$ gray edges with unknown fate...
$27$ features per gray edge

Remove losers in conflict with the winners

- Undetermined edge
- Current 1-best tree
- Winner edge (permanently in 1-best tree)
- Loser edge
This time, the firms were ready

31 gray edges with unknown fate...
27 features per gray edge

Remove losers in conflict with the winners
+ next feature group
31 gray edges with unknown fate...
74 features per gray edge

$\text{This time, the firms were ready}$

- Undetermined edge
- Current 1-best tree
- Winner edge (permanently in 1-best tree)
- - Loser edge
This time, the firms were ready with

31 gray edges with unknown fate...
74 features per gray edge

Undetermined edge

Current 1-best tree (permanently in 1-best tree)

Winner edge

Loser edge

Non-projective decoding to find new 1-best tree
This time, the firms were ready

$28$ gray edges with unknown fate...

74 features per gray edge

Undetermined edge

Current 1-best tree

Winner edge

(permanently in 1-best tree)

Loser edge

Classifier picks winners among the blue edges
This time, the firms were ready...

8 gray edges with unknown fate...

74 features per gray edge

Remove losers in conflict with the winners
This time, the firms were ready.

8 gray edges with unknown fate...
74 features per gray edge

Remove losers in conflict with the winners

- Undetermined edge
- Current 1-best tree
- Winner edge (permanently in 1-best tree)
- Loser edge
This time, the firms were ready.

+ next feature group
8 gray edges with unknown fate...
107 features per gray edge

Undetermined edge
Current 1-best tree
Winner edge (permanently in 1-best tree)
- Loser edge
This time, the firms were ready.

8 gray edges with unknown fate...

107 features per gray edge

Non-projective decoding to find new 1-best tree
This time, the firms were ready.

7 gray edges with unknown fate...
107 features per gray edge

Current 1-best tree

Winner edge (permanently in 1-best tree)

Loser edge

Classifier picks winners among the blue edges
This time, the firms were ready...

3 gray edges with unknown fate...

107 features per gray edge

Remove losers in conflict with the winners

- Undetermined edge
- Current 1-best tree
- Winner edge (permanently in 1-best tree)
- Loser edge
This time, the firms were ready.

3 gray edges with unknown fate...
107 features per gray edge

Remove losers in conflict with the winners

- Undetermined edge
- Current 1-best tree
- Winner edge (permanently in 1-best tree)
- Loser edge
$\text{This time, the firms were ready.}$

+ last feature group

3 gray edges with unknown fate...

268 features per gray edge
This time, the firms were ready.

0 gray edge with unknown fate...
268 features per gray edge

Projective decoding to find final 1-best tree
What Happens During the Average Parse?
What Happens During the Average Parse?

Most edges win or lose early

![Graph showing Time/Accuracy/Edge Percentage over Feature selection stage](image-url)

- runtime %
- UAS %
- remaining edge %
- winner edge %
What Happens During the Average Parse?

Most edges win or lose early

Some edges win late

![Graph showing the relationship between feature selection stage and time/accuracy/edge percentage]
What Happens During the Average Parse?

Later features are helpful

Most edges win or lose early

Some edges win late

Time/Accuracy/Edge Percentage

Feature selection stage

runtime %
UAS %
remaining edge %
winner edge %
What Happens During the Average Parse?

Most edges win or lose early

Linear increase in runtime

Later features are helpful

Some edges win late
Summary: How Early Decisions Are Made

- **Winners**: Will definitely appear in the 1-best tree
- **Losers**: Have the same child as a winning edge; Form cycle with winning edges; Cross a winning edge (optional); Share root ($) with a winning edge (optional)
- **Undetermined**: Add the next feature group to the remaining gray edges
Feature Template Ranking

• Forward selection
Feature Template Ranking

• Forward selection

A  0.60
B  0.49
C  0.55
Feature Template Ranking

- Forward selection

A 0.60  
B 0.49  
C 0.55  

1 A
Feature Template Ranking

- Forward selection

  - A 0.60
  - B 0.49
  - C 0.55

  A & B 0.80
  A & C 0.85
Feature Template Ranking

- Forward selection

A  0.60  \rightarrow A \rightarrow A&B  0.80 \rightarrow A&C  0.85 \rightarrow C
B  0.49  
C  0.55
Feature Template Ranking

- Forward selection

- A 0.60
- B 0.49
- C 0.55
Feature Template Ranking

• Forward selection

A 0.60  \rightarrow  A&B 0.80  \rightarrow  A&C 0.85  \rightarrow  C 0.9

• Grouping

- head cPOS + mod cPOS + in-between punct # 0.49
- in-between cPOS 0.59
- head POS + mod POS + in-between conj # 0.71
- head POS + mod POS + in-between POS + dist 0.72
- head token + mod cPOS + dist 0.80
- \vdots
Feature Template Ranking

● Forward selection

A 0.60 \rightarrow A \rightarrow A&B 0.80 \rightarrow A&C 0.85 \rightarrow C \rightarrow A&C&B 0.9

● Grouping

- head cPOS + mod cPOS + in-between punct # 0.49
- in-between cPOS 0.59
- head POS + mod POS + in-between conj # 0.71
- head POS + mod POS + in-between POS + dist 0.72
- head token + mod cPOS + dist 0.80

\[ \{ + \sim 0.1 \} \]
Partition Feature List Into Groups

Unlabeled attachment score (UAS)

Number of feature templates used

1st-order non-proj
How to pick the winners?
How to pick the winners?

- Learn a classifier
How to pick the winners?

- Learn a classifier
- Features
  - Currently added parsing features
  - Meta-features -- confidence of a prediction
How to pick the winners?

• Learn a classifier

• Features
  – Currently added parsing features
  – Meta-features -- confidence of a prediction

• Training examples
  – Input: each blue edge in current 1-best tree
  – Output: is the edge in the gold tree? If so, we want it to win!
Classifier Features

- Currently added parsing features
- Meta-features
  - the firms : ..., 0.5, 0.8, 0.85
    (scores are normalized by the sigmoid function)
  - Margins to the highest-scoring competing edge
  - Index of the next feature group

This time the firms were 

\[
\begin{align*}
0.72 & \\
0.65 & \\
0.30 & \\
0.23 & \\
0.12 & 
\end{align*}
\]
Classifier Features

- **Currently added** parsing features
- **Meta-features**
  - the firms: ..., 0.5, 0.8, 0.85
    - (scores are normalized by the sigmoid function)
  - Margins to the highest-scoring competing edge
  - Index of the next feature group

This time the firms were $0.72, 0.65, 0.30, 0.23, 0.12$. 

Classifier Features

- Currently added parsing features

- Meta-features
  - the firms: ..., 0.5, 0.8, 0.85 (scores are normalized by the sigmoid function)
  - Margins to the highest-scoring competing edge
  - Index of the next feature group

Dynamic Features

$0.72$

0.65

0.30

0.23

the firms were

0.12

This time
How To Train With Dynamic Features

- Training examples are not fixed in advance!
- Winners/losers from stages $< k$ affect:
  - Set of edges to classify at stage $k$
  - The dynamic *features* of those edges at stage $k$
- Bad decisions can cause future errors
How To Train With Dynamic Features

• Training examples are not fixed in advance!!
• Winners/losers from stages < k affect:
  – Set of edges to classify at stage k
  – The dynamic features of those edges at stage k
• Bad decisions can cause future errors

Reinforcement / Imitation Learning

• Dataset Aggregation (DAgger) (Ross et al., 2011)
  – Iterates between training and running a model
  – Learns to recover from past mistakes
Upper Bound of Our Performance

“Labels”
- Gold edges always win
- 96.47% UAS with 2.9% first-order features
How To Train Our Parser

1. Train parsers (non-projective, projective) using all features

2. Rank and group feature templates

3. Iteratively train a classifier to decide winners/losers
Experiment

• Data
  – Penn Treebank: English
  – CoNLL-X: Bulgarian, Chinese, German, Japanese, Portuguese, Swedish

• Parser
  – MSTParser (McDonald et al., 2006)

• Dynamically-trained Classifier
  – LibLinear (Fan et al., 2008)
Dynamic Feature Selection Beats Static Forward Selection

![Graph showing runtime vs. unlabeled attachment score (UAS)]
Dynamic Feature Selection Beats Static Forward Selection

Always add the next feature group to all edges

Add features as needed
Experiment: 1st-order 2x to 6x speedup
Experiment: 1st-order
~0.2% loss in accuracy

\[ \text{relative accuracy} = \frac{\text{accuracy of the pruning parser}}{\text{accuracy of the full parser}} \]
Second-order Dependency Parsing

- Features depend on the siblings as well
- First-order:
  - $O(n^2)$ substructure to score
- Second-order:
  - $O(n^3)$ substructure to score
    - $\sim 380$ feature templates
    - $\sim 96M$ features
- Decoding: still $O(n^3)$
Experiment: 2nd-order 2x to 8x speedup

- Bulgarian: 5x speedup (DynFS) vs. 1x baseline
- Chinese: 5x speedup (DynFS) vs. 1x baseline
- English: 6x speedup (DynFS) vs. 1x baseline
- German: 7x speedup (DynFS) vs. 1x baseline
- Japanese: 8x speedup (DynFS) vs. 1x baseline
- Portuguese: 7x speedup (DynFS) vs. 1x baseline
- Swedish: 5x speedup (DynFS) vs. 1x baseline

Legend:
- DynFS
- Baseline
Experiment: 2nd-order
~0.3% loss in accuracy
Ours vs Vine Pruning (Rush and Petrov, 2012)

- **Vine pruning**: a very fast parser that speeds up using orthogonal techniques
  - Start with short edges (*fully* scored)
  - Add long edges in if needed

- **Ours**
  - Start with all edges (*partially* scored)
  - Quickly remove unneeded edges

- Could be combined for further speedup!
VS Vine Pruning: 1st-order comparable performance

![Chart showing speedup for different languages with DynFS, VineP, and Baseline]
VS Vine Pruning: 1st-order

![Bar chart showing relative accuracy for different languages and pruning methods. The chart compares DynFS, VineP, and Baseline for Bulgarian, English, German, Japanese, Portuguese, and Swedish. The accuracy ranges from 99.3% to 100.3%.]
VS Vine Pruning: 2nd-order

The diagram compares the speedup of different pruning methods—DynFS, VineP, and Baseline—for various languages: Bulgarian, Chinese, English, German, Japanese, Portuguese, and Swedish.
VS Vine Pruning: 2nd-order

![Relative accuracy comparison for various languages using VineP, DynFS, and Baseline models.](image-url)
Conclusion

- **Feature computation** is expensive in structured prediction
- Commitment should be made *dynamically*
- Early commitment to edges reduce both searching and scoring time
- Can be used in other feature-rich models for structured prediction
Backup Slides
This time, the firms were ready.
Reinforcement Learning 101

• Markov Decision Process (MDP)
  – **State**: all the information helping us to make decisions
  – **Action**: things we choose to do
  – **Reward**: criteria for evaluating actions
  – **Policy**: the “brain” that makes the decision

• Goal
  – Maximize the expected future reward
Policy Learning

• Markov Decision Process (MDP)
  \[ \pi ( \text{the firms} + \text{context}) = \text{add / lock} \]
  - reward = accuracy + \( \lambda \cdot \text{speed} \)

• Reinforcement learning
  - Delayed reward
  - Long time to converge

• Imitation learning
  - Mimic the oracle
  - Reduced to supervised classification problem
Imitation Learning

- (near) optimal performance
- generate target action in any given state
  \[ \pi \left( \text{the firms} + \text{context} \right) = \text{lock} \]
  \[ \pi \left( \text{time} , \text{the} + \text{context} \right) = \text{add} \]

\[ \{ \psi(s), \pi^*(s) \} \]

Binary classifier
Dataset Aggregation (DAgger)

- Collect data from the oracle only
  - Different distribution at training and test time

- Iterative policy training
  - Correct the learner's mistake
  - Obtain a policy performs well under its own policy distribution
Experiment (1st-order)

\[
cost = \frac{\text{# feature templates used}}{\text{total # feature templates on the statically pruned graph}}
\]
Experiment (2nd-order)

Feature cost

- Bulgarian
- Chinese
- English
- German
- Japanese
- Portuguese
- Swedish

DynFS
Second-order Parsing
Second-order Parsing

![Graph showing mean time vs. sentence length]