Forest Reranking

Discriminative Parsing with Non-Local Features

Liang Huang
University of Pennsylvania

ACL 2008 talk, Columbus, OH, June 2008
Is Supervised Parsing Done?

is it a done area?

Bod (2007)
Is the End of Supervised Parsing in Sight?
Is Supervised Parsing Done?

Is it a done area?

- motivation: use non-local features
- linguistically-motivated features for $n$-best reranking (Charniak and Johnson, 2005; Collins, 2000)
- but can we integrate them back into chart parsing?
- YES: using a packed forest!
- result: best whole Treebank parsing accuracy to date

Bod (2007)
Is the End of Supervised Parsing in Sight?
Why is $n$-best list a bad idea?

- too few variations (limited scope)
  - 41% correct parses are not in ~30-best (Collins, 2000)
- worse for longer sentences; tiny fraction of whole space
- too many redundancies
  - 50-best usually encodes 5-6 binary decisions ($2^5<50<2^6$)
Why is \textit{n}-best list a bad idea?

- too few variations (limited scope)
  - 41\% correct parses are not in \textasciitilde30-best (Collins, 2000)
  - worse for longer sentences; tiny fraction of whole space
- too many redundancies
  - 50-best usually encodes 5-6 binary decisions ($2^5 < 50 < 2^6$)
Outline

• Packed Forest and General Idea
• Forest Reranking and Non-Local Features
  • Perceptron for Generic Reranking
  • Local vs. Non-Local Features
  • Incremental Computation of Non-Local Features
• Decoding Algorithm
• Experiments
Packed Forest

- a compact representation of many parses
- by sharing common sub-derivations
- polynomial-space encoding of exponentially large set

(Klein and Manning, 2001; Huang and Chiang, 2005)

I saw him with a mirror.
Packed Forest

- a compact representation of many parses
- by sharing common sub-derivations
- polynomial-space encoding of exponentially large set

(Klein and Manning, 2001; Huang and Chiang, 2005)

I saw him with a mirror.
Lattices vs. Forests

- forest generalizes “lattice” from finite-state world
- both are compact encodings of exponentially many derivations (paths or trees)
- graph $\Rightarrow$ hypergraph; regular grammar $\Rightarrow$ CFG
Reranking on a Forest?

- with only local features
  - dynamic programming, tractable
    (Taskar et al. 2004; McDonald et al., 2005)

- with non-local features
  - intractable, so we do approximation
  - on-the-fly reranking at internal
  - use non-locals as early and as much as possible!

<table>
<thead>
<tr>
<th>methods \ features</th>
<th>local</th>
<th>non-local</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-best reranking</td>
<td>only at the root node</td>
<td></td>
</tr>
<tr>
<td>DP-based discrim. parsing</td>
<td>exact</td>
<td>N/A</td>
</tr>
<tr>
<td>forest reranking</td>
<td>exact</td>
<td>on-the-fly</td>
</tr>
</tbody>
</table>
Reranking on a Forest?

- with only local features
  - dynamic programming, tractable
    (Taskar et al. 2004; McDonald et al., 2005)
- with non-local features
  - intractable, so we do approximation
  - on-the-fly reranking at internal
  - use non-locals as early and as much as possible!

<table>
<thead>
<tr>
<th>methods \ features</th>
<th>local</th>
<th>non-local</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>n</em>-best reranking</td>
<td>only at the root node</td>
<td></td>
</tr>
<tr>
<td>DP-based discrim. parsing</td>
<td>exact</td>
<td>N/A</td>
</tr>
<tr>
<td>forest reranking</td>
<td>exact</td>
<td>on-the-fly</td>
</tr>
</tbody>
</table>
Reranking on a Forest?

- with only local features
  - dynamic programming, tractable (Taskar et al. 2004; McDonald et al., 2005)
- with non-local features
  - intractable, so we do approximation
  - on-the-fly reranking at internal
  - use non-locals as early and as much as possible!

<table>
<thead>
<tr>
<th>methods \ features</th>
<th>local</th>
<th>non-local</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>n</em>-best reranking</td>
<td>only at the root node</td>
<td></td>
</tr>
<tr>
<td>DP-based discrim. parsing</td>
<td>exact</td>
<td>N/A</td>
</tr>
<tr>
<td>forest reranking</td>
<td>exact</td>
<td>on-the-fly</td>
</tr>
</tbody>
</table>
Outline

- Packed Forest and General Idea
- Forest Reranking and Non-Local Features
  - Perceptron for Generic Reranking
  - Local vs. Non-Local Features
  - Incremental Computation of Non-Local Features
- Decoding Algorithm
- Experiments
Generic Reranking by Perceptron

- for each sentence $s_i$, we have a set of candidates $\text{cand}(s_i)$
- and an oracle tree $y_i^+$, among the candidates
- a feature mapping from tree $y$ to vector $\mathbf{f}(y)$

1: **Input**: Training examples $\{\text{cand}(s_i), y_i^+\}_{i=1}^N$

2: $\mathbf{w} \leftarrow 0$  \hspace{1cm} \triangleright \text{initial weights}$

3: for $t \leftarrow 1 \ldots T$ do

4: \hspace{0.5cm} for $i \leftarrow 1 \ldots N$ do

5: \hspace{1cm} $\hat{y} = \arg\max_{y \in \text{cand}(s_i)} \mathbf{w} \cdot \mathbf{f}(y)$

6: \hspace{1cm} if $\hat{y} \neq y_i^+$ then

7: \hspace{1.5cm} $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(y_i^+) - \mathbf{f}(\hat{y})$

8: return $\mathbf{w}$  \hspace{1cm} \triangleright T \text{ iterations}$

(Collins, 2002)
Generic Reranking by Perceptron

- for each sentence $s_i$, we have a set of candidates $\text{cand}(s_i)$
- and an oracle tree $y_i^+$, among the candidates
- a feature mapping from tree $y$ to vector $f(y)$

1: **Input**: Training examples $\{\text{cand}(s_i), y_i^+\}_{i=1}^N$

2: $w \leftarrow 0$

3: for $t \leftarrow 1 \ldots T$ do

4:   for $i \leftarrow 1 \ldots N$ do

5:     $\hat{y} = \arg \max_{y \in \text{cand}(s_i)} w \cdot f(y)$

6:     if $\hat{y} \neq y_i^+$ then

7:       $w \leftarrow w + f(y_i^+) - f(\hat{y})$

8: return $w$

(Collins, 2002)
Features

- A feature $f$ is a function from tree $y$ to a real number.
- $f_1(y) = \log \Pr(y)$ is the log Prob from generative parser.
- Every other feature counts the number of times a particular configuration occurs in $y$.

```
TOP
   /\   \\
  S   |
  |   |  \\
NP  VP |
   |   /\  |
PRP VBD NP PP
   |   |   |  |
I saw DT NN IN NP
   |   |   |  |
the boy with DT NN
   |   |  |
   the a telescope
```

Our features are from:
(Charniak & Johnson, 2005)
(Collins, 2000)
Features

- a feature $f$ is a function from tree $y$ to a real number
- $f_1(y) = \log \Pr(y)$ is the log Prob from generative parser
- every other feature counts the number of times a particular configuration occurs in $y$

Our features are from (Charniak & Johnson, 2005) (Collins, 2000)

Instances of Rule feature

$$f_{100}(y) = f_{S \rightarrow NP \ VP}(y) = 1$$
$$f_{200}(y) = f_{NP \rightarrow DT \ NN}(y) = 2$$
Features

- A feature $f$ is a function from tree $y$ to a real number.
- $f_1(y) = \log \Pr(y)$ is the log Prob from generative parser.
- Every other feature counts the number of times a particular configuration occurs in $y$.

\[
\begin{align*}
\text{TOP} & \quad \text{S} \\
\text{NP} & \quad \text{VP} \\
\text{PRP} & \quad \text{VBD} \\
\text{I} & \quad \text{saw} \\
\text{NP} & \quad \text{DT} \quad \text{NN} \\
\text{IN} & \quad \text{with} \\
\text{NP} & \quad \text{DT} \quad \text{NN} \\
\text{a} & \quad \text{telescope}
\end{align*}
\]

Our features are from (Charniak & Johnson, 2005) (Collins, 2000)

Instances of Rule feature:
\[
\begin{align*}
f_{100}(y) &= f_{S \rightarrow \text{NP VP}}(y) = 1 \\
f_{200}(y) &= f_{\text{NP} \rightarrow \text{DT NN}}(y) = 2
\end{align*}
\]
Local vs. Non-Local Features

- a feature is **local** iff. it can be factored among local productions of a tree (i.e., hyperedges in a forest)
- local features can be precomputed on each hyperedge in the forest; non-locals cannot
Local vs. Non-Local Features

- A feature is **local** iff. it can be factored among local productions of a tree (i.e., hyperedges in a forest).
- Local features can be pre-computed on each hyperedge in the forest; non-locals cannot.

```
TOP
    | S
    | NP
    |   VP
    |    | PRP VBD NP PP
    |    |   I saw DT NN IN
    |    | the boy with
    |    |     NP DT NN
    |    |       a telescope

Rule is local
```
Local vs. Non-Local Features

- A feature is **local** iff. it can be factored among local productions of a tree (i.e., hyperedges in a forest).
- Local features can be pre-computed on each hyperedge in the forest; non-locals cannot.

```
TOP
  \--- S
    \--- NP
        \--- PRP
        \--- VBD
        \--- I
        \--- saw
    \--- VP
        \--- NP
            \--- DT
            \--- NN
            \--- IN
            \--- with
        \--- PP
    \--- NP
        \--- DT
        \--- NN
        \--- a
depen with
```

**ParentRule** is non-local.

**Rule** is local.
Local vs. Non-Local: Examples

- **CoLenPar** feature captures the difference in lengths of adjacent conjuncts (Charniak and Johnson, 2005)

CoLenPar: 2
Local vs. Non-Local: Examples

• **CoLenPar** feature captures the difference in lengths of adjacent conjuncts *(Charniak and Johnson, 2005)*

![Diagram]

- CoLenPar: 2

+ 4 words
+ 6 words

local!
CoPar feature captures the depth to which adjacent conjuncts are isomorphic (Charniak and Johnson, 2005)
Local vs. Non-Local: Examples

- **CoPar** feature captures the depth to which adjacent conjuncts are isomorphic (Charniak and Johnson, 2005)

CoPar: 4

non-local!
Factorizing non-local features

- going bottom-up, at each node
- compute (partial values of) feature instances that become computable at this level
- postpone those uncomputable to ancestors

unit instance of ParentRule feature at VP node
Factorizing non-local features

- going bottom-up, at each node
- compute (partial values of) feature instances that become computable at this level
- postpone those uncomputable to ancestors

Unit instance of ParentRule feature at VP node
Factorizing non-local features

- going bottom-up, at each node
- compute (partial values of) feature instances that become computable at this level
- postpone those uncomputable to ancestors

Example parse tree:

```
  (TOP
    (S
      (VP
        (VBD saw)
        (NP
          (DT the)
          (NN boy))
        (PP
          (IN with)
          (NP
            (DT a)
            (NN telescope))))
    )
  )
```
Factorizing non-local features

- going bottom-up, at each node
- compute (partial values of) feature instances that become computable at this level
- postpone those uncomputable to ancestors

unit instance of ParentRule feature at S node
Factorizing non-local features

- going bottom-up, at each node
- compute (partial values of) feature instances that become computable at this level
- postpone those uncomputable to ancestors
Factorizing non-local features

• going bottom-up, at each node
• compute (partial values of) feature instances that become computable at this level
• postpone those uncomputable to ancestors

unit instance of ParentRule feature at S node
Factorizing non-local features

- going bottom-up, at each node
- compute (partial values of) feature instances that become computable at this level
- postpone those uncomputable to ancestors

unit instance of ParentRule feature at TOP node

TOP
   S
   NP
   VP
   PRP VBD NP PP
   I saw DT NN IN NP
   the boy with DT NN
   a telescope
Factorizing non-local features

- going bottom-up, at each node
- compute (partial values of) feature instances that become computable at this level
- postpone those uncomputable to ancestors

unit instance of ParentRule feature at TOP node

I saw the boy with a telescope.
Factorizing non-local features

- going bottom-up, at each node
- compute (partial values of) feature instances that become computable at this level
- postpone those uncomputable to ancestors

unit instance of *ParentRule*

feature at TOP node

non-local features factor across nodes *dynamically*

local features factor across hyperedges *statically*
an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)

unit instances are boundary words between subtrees
NgramTree (C&J 05)

- an NgramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

(unit instance of node A)

\[
\begin{align*}
A_{i,k} & \quad B_{i,j} \\
& \quad C_{j,k}
\end{align*}
\]

\[
\begin{align*}
w_i \ldots w_{j-1} & \quad w_j \ldots w_{k-1}
\end{align*}
\]
an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)

unit instances are boundary words between subtrees
NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

![Diagram of NGramTree]

Unit instance of node A

A_{i,k}

B_{i,j}

C_{j,k}

w_i \ldots w_{j-1}

w_j \ldots w_{k-1}

VBD saw

NP

PTP

VP

DT the

IN with

DT

NN

NP

a telescope
an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)

unit instances are boundary words between subtrees
• an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)

• unit instances are **boundary words** between subtrees
an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)

unit instances are **boundary words** between subtrees

NggramTree (C&J 05)
an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)

unit instances are boundary words between subtrees
NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees
Outline

• Packed Forest and General Idea
• Forest Reranking and Non-Local Features
  • Perceptron for Generic Reranking
  • Local vs. Non-Local Features
  • Incremental Computation of Non-Local Features
• Decoding Algorithm
  \[ \hat{y} = \arg\max_{y \in \text{cand}(s_i)} w \cdot f(y) \]
• Experiments
General Idea of Decoding

Diagram:

- VP<sub>1,6</sub>
- e<sub>2</sub>
- e<sub>1</sub>
- VBD<sub>1,2</sub>
- NP<sub>2,6</sub>
- NP<sub>2,3</sub>
- PP<sub>3,6</sub>
General Idea of Decoding

- bottom-up (chart parsing)
General Idea of Decoding

- bottom-up (chart parsing)
- keep top $k$ trees at each node
  - combine top subtrees
- score unit non-local features
General Idea of Decoding

- bottom-up (chart parsing)
- keep top \( k \) trees at each node
  - combine top subtrees
  - score unit non-local features
- similar to machine translation decoding with integrated language models
  - non-local features <=> LM combo
  - so we use forest rescoring from MT (Chiang 2007; Huang and Chiang 2007) to speed up the computation
General Idea of Decoding

- bottom-up (chart parsing)
- keep top $k$ trees at each node
  - combine top subtrees
  - score unit non-local features
- similar to machine translation decoding with integrated language models
  - non-local features $\leftrightarrow$ LM combo
- so we use forest rescoring from MT (Chiang 2007; Huang and Chiang 2007) to speed up the computation
General Idea of Decoding

- bottom-up (chart parsing)
- keep top $k$ trees at each node
  - combine top subtrees
  - score unit non-local features
- similar to machine translation decoding with integrated language models
  - non-local features $\leftrightarrow$ LM combo
- so we use forest rescoring from MT
  (Chiang 2007; Huang and Chiang 2007)
  to speed up the computation
General Idea of Decoding

- bottom-up (chart parsing)
- keep top $k$ trees at each node
  - combine top subtrees
  - score unit non-local features
- similar to machine translation decoding with integrated language models
  - non-local features $\leftrightarrow$ LM combo
- so we use forest rescoring from MT (Chiang 2007; Huang and Chiang 2007)
  to speed up the computation
Faster Decoding

- best-first exploration of hyperedges *simultaneously!* significant savings of computation
- most of the item combinations are neglected

(Huang and Chiang, 2005, 2007; Chiang, 2007)
Experiments

scaled to the whole Penn Treebank
Data Preparation

- use Charniak parser as baseline
- standard split: train: sec 02-21, dev: sec 22, test: sec 23
- training set split into 20 fold (cross-validation style)
- modify Charniak parser to output forests!
  - pruned by an Inside-Outside style algorithm
- use 15 features templates from (Charniak and Johnson, 2005; Collins, 2000); 800, 582 feature instances (~70% local)
- both $n$-best and forest reranking systems implemented in pure Python, on 64-bit Dual-core 3.0 GHz machines
Forest vs. n-best Oracles

- forests enjoy higher oracle scores than n-best lists
- a dynamic programming algorithm for forest oracle
Forest vs. n-best Oracles

- forests enjoy higher oracle scores than n-best lists
- a dynamic programming algorithm for forest oracle

![Graph showing the comparison between 1-best, n-best, and forest oracles. The graph plots Parseval F-score (%) against the average number of hyperedges or brackets per sentence. The x-axis represents the average number of hyperedges or brackets, ranging from 0 to 2000. The y-axis represents the Parseval F-score, ranging from 89.0 to 99.0. The graph includes markers for different n-values: n=10, n=50, and n=100. The highest F-score for each n-value is indicated by a red circle, with scores of 97.8 for n=10, 98.6 for n=50, and 97.2 for n=100. The 1-best oracle and n-best oracle are represented by dashed lines. The forest oracle is represented by a solid line. The graph also includes a data point for the 1-best oracle at 89.7.]
Main Results

- forest reranking outperforms both 50-best and 100-best reranking
- and can be trained on the whole treebank in ~1 day even with a pure Python implementation!

<table>
<thead>
<tr>
<th>approach</th>
<th>training time</th>
<th>F1%</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline: 1-best Charniak parser</td>
<td></td>
<td>89.72</td>
</tr>
<tr>
<td>50-best reranking</td>
<td>4 x 0.3h</td>
<td>91.43</td>
</tr>
<tr>
<td>100-best reranking</td>
<td>4 x 0.7h</td>
<td>91.49</td>
</tr>
<tr>
<td>forest reranking</td>
<td>4 x 6.1h</td>
<td>91.69</td>
</tr>
</tbody>
</table>

details in the paper.
## Comparison with Others

<table>
<thead>
<tr>
<th>approach</th>
<th>system</th>
<th>F₁%</th>
</tr>
</thead>
<tbody>
<tr>
<td>reranking</td>
<td>Collins (2000)</td>
<td>89.7</td>
</tr>
<tr>
<td></td>
<td>Charniak and Johnson (2005)</td>
<td>91.0</td>
</tr>
<tr>
<td>dynamic programming</td>
<td>Petrov and Klein (2008)</td>
<td>88.3</td>
</tr>
<tr>
<td></td>
<td><em>this work</em></td>
<td>91.7</td>
</tr>
<tr>
<td>generative</td>
<td>Bod (2000)</td>
<td>90.7</td>
</tr>
<tr>
<td></td>
<td>Petrov and Klein (2007)</td>
<td>90.1</td>
</tr>
<tr>
<td>semi-supervised</td>
<td>McClosky et al. (2006)</td>
<td>92.1</td>
</tr>
</tbody>
</table>
Conclusion

- A Framework for Reranking on Packed Forests
  - forests have more variations and smaller sizes
  - dynamic programming algorithm for forest oracles
- Two Key Ideas that made it work
  - incremental, recursive computation of features
  - forest rescoring for approximate decoding
- Discriminative training scaled to the whole PTB
  - better than both 50-best and 100-best reranking
  - better than any previous results trained on PTB
Conclusion

- more akin to traditional chart parsing, not reranking!
  - multipass search (Goodman, 1997)
    - non-local features in the pruned forest
    - but without blowing up the forest
  - better search algorithms should help!
  - could in principle incorporate fancier features
- also applicable to other problems involving forest
  - sequence segmentation/labeling, dependency parsing, machine translation, generation, ...
Forest is your friend.  Save the forest.

Thank you!

Forest-dumping Charniak parser will be available online.
Global Feature - RightBranch

- length of rightmost (non-punctuation) path
- English has a right-branching tendency

(Charniak and Johnson, 2005)
• a **WordEdges** feature classifies a node by its label, (binned) span length, and surrounding words

• a **POSEdges** feature uses surrounding POS tags

**WordEdges** is local

\[ f_{400}(y) = f_{NP \ 2 \ saw \ with}(y) = 1 \]
• a **WordEdges** feature classifies a node by its label, (binned) span length, and surrounding words

• a **POSEdges** feature uses surrounding POS tags

**WordEdges** is local

\[ f_{400}(y) = f_{NP \: saw \: with}(y) = 1 \]

**POSEdges** is non-local

\[ f_{800}(y) = f_{NP \: 2 \: VBD \: IN}(y) = 1 \]
WordEdges (C&J 05)

- A WordEdges feature classifies a node by its label, (binned) span length, and surrounding words.
- A POSEdges feature uses surrounding POS tags.

WordEdges is local:

\[ f_{400}(y) = f_{NP \ 2 \ \text{saw \ with}}(y) = 1 \]

POSEdges is non-local:

\[ f_{800}(y) = f_{NP \ 2 \ \text{VBD \ IN}}(y) = 1 \]

Local features comprise ~70% of all instances!
Heads (C&J 05, Collins 00)

- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children
Heads (C&J 05, Collins 00)

- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children

```
TOP/saw
  \|-- S/saw
    \|-- VP/saw
        \|-- NP/I
            \|-- PRP/I
                \|-- VBD/saw
                    \|-- I
                        \|-- saw
                            \|-- DT/the
                                    \|-- NN/boy
                                        \|-- IN/with
                                            \|-- NP/a
                                                \|-- DT/a
                                                    \|-- NN/telescope
                                                        \|-- a
                                                            \|-- telescope
```
Heads (C&J 05, Collins 00)

- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children
Approximate Decoding

- bottom-up, keeps top \( k \) derivations at each node
  - forest rescoring from MT (Chiang 2007; Huang and Chiang 07)
- priority queue for next-best (Huang and Chiang, 2005)
  - each iteration pops the best and pushes successors
- unit non-local feature costs as a non-monotonic cost
Approximate Decoding

- bottom-up, keeps top $k$ derivations at each node
  - forest rescoring from MT (Chiang 2007; Huang and Chiang 07)
- priority queue for next-best (Huang and Chiang, 2005)
  - each iteration pops the best and pushes successors
- unit non-local feature costs as a non-monotonic cost

\[
\mathbf{w} \cdot \mathbf{f_N}(\cdot) = 0.5
\]
Approximate Decoding

- bottom-up, keeps top $k$ derivations at each node
  - forest rescoring from MT (Chiang 2007; Huang and Chiang 2007)
- priority queue for next-best (Huang and Chiang, 2005)
- each iteration pops the best and pushes successors
- unit non-local feature costs as a non-monotonic cost

$$\mathbf{w} \cdot \mathbf{f}_N(\ ) = 0.5$$

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
<td>9.5</td>
</tr>
<tr>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
<td>9.4</td>
</tr>
<tr>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
<td>12.1</td>
</tr>
</tbody>
</table>
Approximate Decoding

- bottom-up, keeps top $k$ derivations at each node
  - forest rescoring from MT (Chiang 2007; Huang and Chiang 2007)
- priority queue for next-best (Huang and Chiang, 2005)
  - each iteration pops the best and pushes successors
- unit non-local feature costs as a non-monotonic cost

$$w \cdot f_N(\text{ }) = 0.5$$

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>3.0</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2.5</td>
<td>9.0</td>
<td>9.5</td>
</tr>
<tr>
<td>1.1</td>
<td>2.4</td>
<td>9.5</td>
<td>9.4</td>
</tr>
<tr>
<td>3.5</td>
<td>5.1</td>
<td>17.0</td>
<td>12.1</td>
</tr>
</tbody>
</table>
Approximate Decoding

- process all hyperedges simultaneously!
  significant savings of computation

complexity: $O(E + V \cup k \log k)$,
bottom-neck: the time for on-the-fly extraction

(Huang and Chiang, 2005; 2007; Chiang, 2007)
Forest Oracle

the candidate tree that is closest to gold-standard
Optimal Parseval F-score

- find the tree in the forest with highest F-score
- Parseval F₁-score is the harmonic mean between labeled precision and labeled recall
  - can not optimize F-scores on sub-forests separately
  - can not optimize precision and recall simultaneously
- we instead use dynamic programming
  - optimizes the number of matched brackets per given number of test brackets
  - “when the test (sub-) parse has 5 brackets, what is the max. number of matched brackets?”
to combine two nodes along a hyperedge, we need to **distribute** test brackets between the two, and **optimize** the number of matches

\[
(f \otimes g)(t) \triangleq \max_{t_1 + t_2 = t} f(t_1) + g(t_2)
\]
Combining Oracle Functions

- to combine two nodes along a hyperedge, we need to **distribute** test brackets between the two, and **optimize** the number of matches.

\[
(f \otimes g)(t) \triangleq \max_{t_1+t_2=t} f(t_1) + g(t_2)
\]

![Diagram of nodes and hyperedge](image)

<table>
<thead>
<tr>
<th>t</th>
<th>f(t)</th>
<th>g(t)</th>
<th>(f\otimes g)(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ora[w]**

this node matched?
Combining Oracle Functions

- to combine two nodes along a hyperedge, we need to **distribute** test brackets between the two, and **optimize** the number of matches.

\[(f \otimes g)(t) \triangleq \max_{t_1 + t_2 = t} f(t_1) + g(t_2)\]

**final answer:**

\[F(y^+, y^*) = \max_t \frac{2 \cdot ora[\text{TOP}](t)}{t + |y^*|}\]

<table>
<thead>
<tr>
<th>(t)</th>
<th>((f \otimes g) \uparrow (1,0) (t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>

\[\otimes\]

<table>
<thead>
<tr>
<th>(t)</th>
<th>(g(t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

\[\overset{\uparrow (1,0)}{=}\]

<table>
<thead>
<tr>
<th>(t)</th>
<th>((f \circ g)(t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

\[ora[w]\]

**this node matched?**

<table>
<thead>
<tr>
<th>(t)</th>
<th>((f \otimes g) \uparrow (1,1) (t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
</tr>
</tbody>
</table>
**Forest vs. $n$-best Oracles**

- forests enjoy higher oracle scores than $n$-best lists
- a dynamic programming algorithm for forest oracle

![Graph showing Parseval F-score vs. average # of hyperedges or brackets per sentence for $n=10$, $n=50$, and $n=100$. The graph indicates that forest oracles perform better than $n$-best oracles.]