Data-Driven Parsing with Discontinuous Structures

Wolfgang Maier

Heinrich-Heine-Universität Düsseldorf

GF Summer School 2013
Overview

1. Introduction

2. Data-Driven Parsing with Discontinuous Structures
   - The Data
   - Parsing
   - Making it Faster

3. Going Further
   - Related work
   - Future work
   - Extract a grammar yourself
Overview

1. Introduction

2. Data-Driven Parsing with Discontinuous Structures
   - The Data
   - Parsing
   - Making it Faster

3. Going Further
   - Related work
   - Future work
   - Extract a grammar yourself
Overview

1. Introduction

2. Data-Driven Parsing with Discontinuous Structures
   - The Data
   - Parsing
   - Making it Faster

3. Going Further
   - Related work
   - Future work
   - Extract a grammar yourself
Constituency Parsing

- Determine whether a sentence is admissible given a specific grammar, and find the corresponding structure.
- Different strategies: Top-down/bottom-up, directional/non-directional, ...
Constituency Parsing

- Determine whether a sentence is admissible given a specific grammar, and find the corresponding structure
- Different strategies: Top-down/bottom-up, directional/non-directional, ...

Non-directional bottom-up (CYK)

\[
\begin{align*}
S & \rightarrow \text{NP } \text{VP} \\
\text{VP} & \rightarrow \text{V } \text{NP} \\
\text{VP} & \rightarrow \text{VP } \text{PP} \\
\text{NP} & \rightarrow \text{Det } \text{N} \\
\text{NP} & \rightarrow \text{John} \\
\text{NP} & \rightarrow \text{Sandy} \\
\text{NP} & \rightarrow \text{Mary} \\
\text{V} & \rightarrow \text{sees} \\
\end{align*}
\]

\[\text{John sees Sandy}\]
Constituency Parsing

- Determine whether a sentence is admissible given a specific grammar, and find the corresponding structure.
- Different strategies: Top-down/bottom-up, directional/non-directional, ...

Non-directional bottom-up (CYK)

\[
\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow Det \ N \\
NP & \rightarrow John \\
NP & \rightarrow Sandy \\
NP & \rightarrow Mary \\
V & \rightarrow sees \\
VP & \rightarrow V \ NP \\
VP & \rightarrow VP \ PP \\
VP & \rightarrow VP PP \\
\end{align*}
\]
Constituency Parsing

- Determine whether a sentence is admissible given a specific grammar, and find the corresponding structure
- Different strategies: Top-down/bottom-up, directional/non-directional, ...

Non-directional bottom-up (CYK)

S → NP VP
VP → V NP
VP → VP PP
NP → Det N
NP → John
NP → Sandy
NP → Mary
V → sees

...
Constituency Parsing

- Determine whether a sentence is admissible given a specific grammar, and find the corresponding structure
- Different strategies: Top-down/bottom-up, directional/non-directional, ...

Non-directional bottom-up (CYK)

- $S \rightarrow NP \ VP$
- $VP \rightarrow V \ NP$
- $VP \rightarrow VP \ PP$
- $NP \rightarrow Det \ N$
- $NP \rightarrow John$
- $NP \rightarrow Sandy$
- $NP \rightarrow Mary$
- $V \rightarrow sees$

...
To make parsing data-driven, instead of writing a grammar by hand:

- use a collection of structures which can be interpreted as parse trees of the grammar formalism we are using
- use an algorithm on it which infers the grammar rules which have been used to create a given parse tree
- equip the rules with probabilities (conditional probabilities from rule counts)
- use probabilities for disambiguation
Data

Treebanks are
- corpora in which sentences are annotated with syntactic information
- very small ones contain a few thousand, large ones up to 100k sentences
- typically created from easily accessible text such as news text

Treebank annotation
- mostly aims at neutrality concerning linguistic theories, does not always succeed
- however often has an easily accessible context-free annotation backbone
Grammar Extraction Example

S → NP VP
NP → John
VP → VP PP
VP → V NP
PP → P NP
V → sees
NP → Sandy
P → with
NP → Det N
NP → the telescope

...
Grammar Extraction Example

S → NP VP 1.0
NP → John 0.333
VP → VP PP 0.5
VP → V NP 0.5
PP → P NP 1.0
V → sees 1.0
NP → Sandy 0.333
P → with 1.0
NP → Det N 0.333
...
Discontinuous Structure in Natural Language

A sequence of words which is discontinuous but forms a linguistically meaningful unit.
Discontinuous Structure in Natural Language

A sequence of words which is discontinuous but forms a linguistically meaningful unit.
Discontinuity

Examples: German

- Extraposed relative clauses

  (1) wieder treffen alle Attribute zu, die auch sonst immer passen
      otherwise always fit
  ‘Again, the same attributes as always apply.’
Discontinuity

Examples: German

- Extraposed relative clauses
  
  (1) wieder treffen alle Attribute zu, die auch
  again match all attributes VPART which also
  sonst immer passen
  otherwise always fit
  ‘Again, the same attributes as always apply.’

- Topicalization

(2) Der CD wird bald ein Buch folgen
  The CD will soon a book follow
  ‘The CD will soon be followed by a book.’
Discontinuity is frequent in natural language, not only in languages with a relatively free word order.
Discontinuity

Discontinuity is frequent in natural language, not only in languages with a relatively free word order.

Examples: English

- Relative clause

(3) They sow a row of male-fertile plants nearby, which then pollinate the male-sterile plants.
Discontinuity

Discontinuity is frequent in natural language, not only in languages with a relatively free word order.

Examples: English

- Relative clause
  
  (3) They sow a row of male-fertile plants nearby, which then pollinate the male-sterile plants.

- Long extraction
  
  (4) Those chains include Bloomingdale’s, which Campeau recently said it will sell.
Annotation in the Penn Treebank

“Movement”: Indirect annotation w/ trace nodes and coindexation
Annotation in the Penn Treebank

“Movement”: Indirect annotation w/ trace nodes and coindexation
Annotation in the German NeGra/TIGER Treebanks

Direct annotation using crossing branches
Annotation in the German NeGra/TIGER Treebanks

Direct annotation using crossing branches

Penn-Treebank-style annotation can be converted into this format

Maier 11/41
Annotation in the German NeGra/TIGER Treebanks

Direct annotation using crossing branches

Penn-Treebank-style annotation can be converted into this format [Evang and Kallmeyer, 2011]
Quantifying Discontinuity

**Discontinuity measures** for constituent structures:

- Gap degree
- Well-nestedness/Ill-nestededness
Quantifying Discontinuity

Discontinuity measures for constituent structures:

- Gap degree
- Well-nestedness/Ill-nestedness

Notion of yield

The yield $\pi(v)$ of a node $v$ in a syntactic structure is the set of position indices of the terminals dominated by $V$. 
Quantifying Discontinuity

Discontinuity measures for constituent structures:
- Gap degree
- Well-nestedness/Ill-nestedness

Notion of yield
The yield $\pi(v)$ of a node $v$ in a syntactic structure is the set of position indices of the terminals dominated by $V$.

$$\pi(v_2) = \{1, 3\}$$
Gap Degree

- **Blocks** of a node $v$: the number of maximal continuous sequences in $\pi(v)$
- **Block degree** of $v$: the number of blocks of $v$
- *Gap degree of $v + 1 = block degree of v*
Gap Degree Example

Example

\[ \begin{array}{c}
\text{set of blocks of } v_2: \\
\end{array} \]
Gap Degree Example

Example

set of blocks of $v_2$: $\{\{1\}, \{3\}\}$
block degree of $v_2$
Gap Degree Example

Example

set of blocks of $v_2$: $\{\{1\}, \{3\}\}$
block degree of $v_2 = 2$
**Well-Nestedness**

There are no **disjoint** yields $\pi(v_1), \pi(v_2)$ of nodes $v_1, v_2$ such that $\pi(v_1), \pi(v_2)$ interleave.
Well-Nestedness

There are no disjoint yields $\pi(v_1), \pi(v_2)$ of nodes $v_1, v_2$ such that $\pi(v_1), \pi(v_2)$ interleave.

Example

$$
\begin{array}{c}
\ \ \\
\ 1 \\
\ v_0 \\
\ 2 \\
\ v_1 \\
\ 3 \\
\ v_2 \\
\ 4 \\
\ v_3 \\
\ v_4 \\
\end{array}
$$

$\rightarrow$ well-nested
Ill-Nestedness

Example

\[ \begin{array}{c}
\text{v}_0 \\
\text{v}_1 & \text{v}_2 \\
\text{v}_3 & \text{v}_4 & \text{v}_5 & \text{v}_6 \\
1 & 2 & 3 & 4 \\
\end{array} \]

→ 1-ill-nested

\[ k \text{-ill-nestedness} \]

There exist disjoint yields \( \pi(v), \pi(v_1), \ldots, \pi(v_k) \) of nodes \( v, v_1, \ldots, v_k \) in a syntactic structure such that \( \pi(v_1), \ldots, \pi(v_k) \) interleave with \( \pi(v) \).
Ill-Nestedness

Example

\[
\begin{align*}
&v_0 \\
&\quad v_1 \\
&\quad \quad v_3 \quad v_4 \quad v_5 \\
&\quad \quad \quad 1 \quad 2 \quad 3 \\
&\quad \quad v_2 \\
&\quad \quad \quad v_8 \quad v_6 \\
&\quad \quad \quad \quad 4 \quad 5 \quad 6 \\
&\quad v_7 \\
&\quad \quad v_9 \\
\end{align*}
\]

$\rightarrow$ 2-ill-nested

\[k\text{-ill-nestedness}\]

There exist disjoint yields $\pi(v), \pi(v_1), \ldots, \pi(v_k)$ of nodes $v, v_1, \ldots, v_k$ in a syntactic structure such that $\pi(v_1), \ldots, \pi(v_k)$ interleave with $\pi(v)$. 
## Empirical Investigation

<table>
<thead>
<tr>
<th></th>
<th>NeGra</th>
<th>TIGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>20597</td>
<td>40013</td>
</tr>
<tr>
<td>gap degree 0</td>
<td>14,648</td>
<td>28,414</td>
</tr>
<tr>
<td>gap degree 1</td>
<td>5,253</td>
<td>10,310</td>
</tr>
<tr>
<td>gap degree 2</td>
<td>687</td>
<td>1,274</td>
</tr>
<tr>
<td>gap degree 3</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>gap degree ≥4</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>well-nested</td>
<td>20339</td>
<td>39573</td>
</tr>
<tr>
<td>1-ill-nested</td>
<td>258</td>
<td>440</td>
</tr>
<tr>
<td>2-ill-nested</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
What about Data-Driven Parsing?

Remember

- Data-driven parsing requires grammar extraction
- However, CFG only supports *continuous* constituents
What about Data-Driven Parsing?

Remember

- Data-driven parsing requires grammar extraction
- However, CFG only supports *continuous* constituents

No (P)CFG from discontinuous constituents!
Resolving Crossing Branches (1)

Reattach non-head children of discontinuous nodes
Introduce non-terminals per continuous block [Boyd, 2007]
What now?

- Resolving crossing branches $\leadsto$ discarding annotation
- What can we do?
Constituency trees: GF extraction

S
  VP
  VP
  PROAV
  VMFIN
  VVPP
VP
  VAINF

Darüber muß nachgedacht werden

It must be thought about it

Maier 22/41
Constituency trees: GF extraction

```
cat VP; VAINF;
```

```
S
  VP
  VP
PROAV über about it
VMFIN müssen must
VVPP nachgedacht thought
VAINF werden be
```

"It must be thought about it"
Constituency trees: GF extraction

```
cat VP; VAINF;
fun funVP : VP -> VAINF -> VP
```

```
PROAV

VP

darüber
about it

VMFIN

VP

muß
must

VVPP

VP

nachgedacht
thought

VAINF

werden
be

“It must be thought about it”
```
Constituency trees: GF extraction

```
cat VP; VAINF;
fun funVP : VP -> VAINF -> VP

lincat VAINF = { p1 : Str };```
Constituency trees: GF extraction

S
  VP
    VP
      PRO
      VMFIN
      VVPP
      VAINF
      "It must be thought about it"

\[
\text{cat VP; VAINF;}
\]
\[
\text{fun funVP : VP -> VAINF -> VP}
\]
\[
\text{lincat VAINF = \{ p1 : Str \};}
\]
\[
\text{lincat VP = \{ p1 : Str ; p2 : Str \};}
\]
Constituency trees: GF extraction

```
S
  VP
  PROAV
  VMFIN
  VP
  VVPP
  VAINF

darüber must nachgedacht werden
about it must thought be
“It must be thought about it”
```

```
cat VP; VAINF;
fun funVP : VP -> VAINF -> VP

lincat VAINF = { p1 : Str };
lincat VP = { p1 : Str ; p2 : Str };
lin funVP rhs1 rhs2 rhs3 = { p1 = rhs1.p1; p2 = rhs1.p2 ++ rhs2.p1 };
```
From GF to LCFRS

cat VP; VAINF;
fun funVP : VP -> VAINF -> VP

lincat VAINF = { p1 : Str };
lincat VP = { p1 : Str ; p2 : Str };
lin funVP rhs1 rhs2 rhs3 = { p1 = rhs1.p1; p2 = rhs1.p2 ++ rhs2.p1 };
From GF to LCFRS

cat VP; VAINF;
fun funVP : VP -> VAINF -> VP

lincat VAINF = { p1 : Str };
lincat VP = { p1 : Str ; p2 : Str };
lin funVP rhs1 rhs2 rhs3 = { p1 = rhs1.p1; p2 = rhs1.p2 ++ rhs2.p1 };

- Omit cat and lincat
From GF to LCFRS

fun funVP : VP -> VAINF -> VP

$$\text{lin } \text{funVP } \text{rhs1 rhs2 rhs3} = \{ \text{p1} = \text{rhs1.p1}; \text{p2} = \text{rhs1.p2} +\text{ rhs2.p1} \};$$

- Omit cat and lincat
- Take the fun and add arity given by lincat to cats ...
From GF to LCFRS

fun funVP : VP -> VAINF -> VP

lin funVP rhs1 rhs2 rhs3 = { p1 = rhs1.p1; p2 = rhs1.p2 ++ rhs2.p1 };

VP2 → VP2 VAINF1

- Omit cat and lincat
- Take the fun and add arity given by lincat to cats...
From GF to LCFRS

fun funVP : VP -> VAINF -> VP

lin funVP rhs1 rhs2 rhs3 = { p1 = rhs1.p1; p2 = rhs1.p2 ++ rhs2.p1 };

VP2 → VP2 VAINF1

- Omit cat and lincat
- Take the fun and add arity given by lincat to cats...
- ...and factor in the linearization
From GF to LCFRS

fun funVP : VP -> VAINF -> VP

lin funVP rhs1 rhs2 rhs3 = { p1 = rhs1.p1; p2 = rhs1.p2 ++ rhs2.p1 };

VP2(X₁,X₂X₃) → VP2(X₁,X₂) VAINF(X₃)

- Omit `cat` and `lincat`
- Take the `fun` and add arity given by `lincat` to `cats` ...
- ...and factor in the linearization
Constituency structure: The LCFRS rules

```
S → VP
S → VP

VP → PROAV(1) VVPP(2) VAINF(3)

VP → VMFIN(1) VMFIN(2) VMFIN(3)

VP → VP(1, 2) VP(2, 3) VMFIN(3)

S1(X1X2X3) → VP2(X1, X3) VMFIN(X2)

VP2(X1, X2X3) → VP2(X1, X2) VAINF(X3)

VP2(X1, X2) → PROAV(X1) VVPP(X2)
```

Handling of lexicon left out

---

"It must be thought about it"
Instead of hierachical constituent structure, use labeled dependencies between words

Each word has a single head and zero or more dependents

Example: “nachgedacht” is the head of “darüber” and a dependent of “werden”
Dependency structure

- Note: Assume extra root node (position 0)
- Yield of a word: Set of own position index and all position indices of words reachable from it
- Example: Yield of “werden” is \{1, 3, 4\}
- Gap degree and well-nestedness work here, too; a structure with gap degree 0 (resp. \(\geq 1\)) is called “projective” (resp. ”non-projective”)

```
Darüber muß nachgedacht werden
PROAV  VMFIN  VVPP  VAINF
```

Maier 25/41
Dependency structures: LCFRS extraction

- Select word, LHS label is head dep. label, RHS labels are POS tag and dependent dep. labels
Dependency structures: LCFRS extraction

- Select word, LHS label is head dep. label, RHS labels are POS tag and dependent dep. labels

root → aux VMFIN
Dependency structures: LCFRS extraction

- Select word, LHS label is head dep. label, RHS labels are POS tag and dependent dep. labels
- Argument of POS tag on RHS is single variable

root → aux VMFIN
Dependency structures: LCFRS extraction

- Select word, LHS label is head dep. label, RHS labels are POS tag and dependent dep. labels
- Argument of POS tag on RHS is single variable

root → aux VMFIN(X)
Dependency structures: LCFRS extraction

- Select word, LHS label is head dep. label, RHS labels are POS tag and dependent dep. labels
- Argument of POS tag on RHS is single variable
- Argument of other RHS non-terminals: One one-variable argument per continuous block

root → aux VMFIN(X)
Dependency structures: LCFRS extraction

- Select word, LHS label is head dep. label, RHS labels are POS tag and dependent dep. labels
- Argument of POS tag on RHS is single variable
- Argument of other RHS non-terminals: One one-variable argument per continuous block

\[ \text{root} \rightarrow \text{aux}(X_1, X_3) \text{ VMFIN}(X) \]
Dependency structures: LCFRS extraction

- Select word, LHS label is head dep. label, RHS labels are POS tag and dependent dep. labels
- Argument of POS tag on RHS is single variable
- Argument of other RHS non-terminals: One one-variable argument per continuous block
- Correct concatenation of all introduced variables into arguments

\[
\text{root} \rightarrow \text{aux}(X_1, X_3) \text{ VMFIN}(X)
\]
Dependency structures: LCFRS extraction

- Select word, LHS label is head dep. label, RHS labels are POS tag and dependent dep. labels
- Argument of POS tag on RHS is single variable
- Argument of other RHS non-terminals: One one-variable argument per continuous block
- Correct concatenation of all introduced variables into arguments

\[
\text{root}(X_1X_2X_3) \rightarrow \text{aux}(X_1,X_3) \text{ VMFIN}(X_2)
\]
Dependency structures: The LCFRS rules
Dependency structures: The LCFRS rules

\[
\begin{align*}
\text{pp}(X) & \rightarrow \text{PROAV}(X) \\
\text{root}(X_1 X_2 X_3) & \rightarrow \text{aux}(X_1, X_3) \text{ VMFIN}(X_2) \\
\text{aux}(X_1, X_2) & \rightarrow \text{pp}(X_1) \text{ VVPP}(X_2) \\
\text{aux}(X_1, X_2 X_3) & \rightarrow \text{aux}(X_1, X_2) \text{ VAINF}(X_3) \\
\text{top}(X_1) & \rightarrow \text{root}(X_1)
\end{align*}
\]
Optimization?

- Discontinuous constituency trees and non-projective dependencies directly interpretable as LCFRS derivations
- However, treebank grammars do not perform well [Charniak, 1996]
- Luckily proximity to PCFG can be exploited
Manual label splitting

- We have seen before how to extract a grammar
- Problem: Some labels are too coarse
- Manual splitting using linguistic criteria can help
  [Klein and Manning, 2003b, Versley, 2005]
Manual label splitting

- We have seen before how to extract a grammar
- Problem: Some labels are too coarse
- Manual splitting using linguistic criteria can help
  [Klein and Manning, 2003b, Versley, 2005]

Splits

- NP split: To all NP labels, we add their respective grammatical function label
- S relative clauses split: We change the label of all relative clauses from S to S-RC.
Binarization: CFG CNF

Binarization reduces length of RHSs (*rank*) to two, lower complexity for CYK parsing
Binarization: CFG CNF

Binarization reduces length of RHSs (rank) to two, lower complexity for CYK parsing

- Leave one non-terminal on the RHS of the original rule and introduce a unique non-terminal which rewrites to the other non-terminals

\[
A \rightarrow B C D E \\
\sim\ A \rightarrow B \ @_1, \ @_1 \rightarrow C D E
\]
Binarization: CFG CNF

Binarization reduces length of RHSs (rank) to two, lower complexity for CYK parsing

- Leave one non-terminal on the RHS of the original rule and introduce a unique non-terminal which rewrites to the other non-terminals
- Repeat until all productions have rank 2

\[
\begin{align*}
A & \to B \ C \ D \ E \\
\leadsto A & \to B \ @_1, \ @_1 \to C \ D \ E \\
\leadsto A & \to B \ @_1, \ @_1 \to C \ @_2, \ @_2 \to D \ E
\end{align*}
\]
Binarization: CFG CNF

Binarization reduces length of RHSs (*rank*) to two, lower complexity for CYK parsing

- Leave one non-terminal on the RHS of the original rule and introduce a unique non-terminal which rewrites to the other non-terminals
- Repeat until all productions have rank 2
- Note: with unique non-terminals, binarized grammar is equivalent to the unbinarized one

\[
A \rightarrow B \ C \ D \ E
\]

\[
\searrow A \rightarrow B \ @_1, \ @_1 \rightarrow C \ D \ E
\]

\[
\searrow A \rightarrow B \ @_1, \ @_1 \rightarrow C \ @_2, \ @_2 \rightarrow D \ E
\]
Binarization: LCFRS

- Works like CFG reduction to Chomsky Normal Form plus handling of linearization
- Different re-orderings of the RHS before binarization give different binarization techniques from the PCFG literature

**Binarizations**

- *Left-to-right*: Binarize strictly left-to-right.
- *Head-outward binarization* [Collins, 1999]:
  - *Head marking* with Collins-style head-rules
  - Expand head first, then sisters to the left, then to the right, or vice versa
- *Optimal binarization*: minimal fan-out and number of variables per production and binarization step
Markovization

- Generalize grammar by adding markovization
- Use a single base binarization non-terminal instead of unique ones
- Information from rule occurrence in treebank added to binarization non-terminals
Markovization

- Generalize grammar by adding markovization
- Use a single base binarization non-terminal instead of unique ones
- Information from rule occurrence in treebank added to binarization non-terminals

Markovization information for bin. non-terminal that comprises original RHS elements $A_i \ldots A_m$:
- Vertical: First $v$ elements of path from $A_i$ to root
- Horizontal: First $h$ elements of $A_i \ldots A_0$
Eventually, we need a probabilistic grammar.
Eventually, we need a probabilistic grammar.

- Count all rule/label occurrences
- Estimate probabilities with Maximum Likelihood Estimation
- Works as for PCFG, sum of probabilities for rules with same LHS must be 1
Example

After extraction and head marking

\[ VP_2(X_1X_2, X_3X_4) \rightarrow ADV_1(X_1) VVPP_1'(X_2) PPER_1(X_3) ADV_1(X_4) \]

occurring below \( S_1 \)

Binarized

- Head-outward binarization, unary top and bottom
- Markovization with \( \nu = 2, h = 1 \)

\[ VP_2(X_1, X_2) \rightarrow \hat{\bigwedge} VP_2^\wedge S_1-ADV_1|_2(X_1, X_2) \]
\[ \hat{\bigwedge} VP_2^\wedge S_1-ADV_1|_2(X_1, X_2X_3) \rightarrow \hat{\bigwedge} VP_2^\wedge S_1-PPER_1|_2(X_1, X_2) ADV_1(X_3) \]
\[ \hat{\bigwedge} VP_2^\wedge S_1-PPER_1|_2(X_1, X_2) \rightarrow \hat{\bigwedge} VP_2^\wedge S_1-ADV_1|_1(X_1) PPER_1(X_2) \]
\[ \hat{\bigwedge} VP_2^\wedge S_1-ADV_1|_1(X_1, X_2) \rightarrowADV_1(X_1) \hat{\bigwedge} VP_2^\wedge S_1-VVPP_1|_1(X_2) \]
\[ \hat{\bigwedge} VP_2^\wedge S_1-VVPP_1|_1(X_1) \rightarrow VVPP_1(X_1) \]
Actual parsing

rpars (http://phil.hhu.de/rparse)
- CYK Parser with weighted deductive parsing
  [Seki et al., 1991, Nederhof, 2003]

GF (http://www.grammaticalframework.org)
- Main difference: left-to-right and prefix valid, means
  binarization is done “on-line”

Disco-DOP (http://www.github.com/andreasvc/disco-dop)
Disco-DOP [van Cranenburgh et al., 2011] integrates LCFRS
parsing with Data-Oriented Parsing [Bod and Scha, 1996]
## Qualitative behavior

### Constituents: OK

- Results lie in the vicinity of results of state-of-the-art PCFG parsing (plus crossing branches)
- Unfortunately no standard test suite for long distance dependencies yet
Qualitative behavior

**Constituents: OK**
- Results lie in the vicinity of results of state-of-the-art PCFG parsing (plus crossing branches)
- Unfortunately no standard test suite for long distance dependencies yet

**Dependencies: Bad**
- Low results. Possible reasons:
  - Lack of graph-global features
  - Unsuitable arc labeling scheme
The Problem

- Parsing complexity for binary $k$-LCFRS: $O(n^{3k})$.
- In practice: PCFG $k = 1$, PLCFRS $k \geq 4$
The Problem

- Parsing complexity for binary $k$-LCFRS: $O(n^{3k})$.
- In practice: PCFG $k = 1$, PLCFRS $k \geq 4$

Too slow already with less than 30 words per sentence
The solutions

- Use A* search with outside estimates [Maier and Kallmeyer, 2010]
  - Improve sorting of partial results such that those are processed first which more quickly lead to goal
The solutions

- Use A* search with outside estimates [Maier and Kallmeyer, 2010]
  - Improve sorting of partial results such that those are processed first which more quickly lead to goal
- Assuring that $k = 2$ [Maier et al., 2012]
  - transformations for treebank trees which preserve discontinuity information
  - specialized, much faster parser
The solutions

- Use $A^*$ search with outside estimates [Maier and Kallmeyer, 2010]
  - Improve sorting of partial results such that those are processed first which more quickly lead to goal
- Assuring that $k = 2$ [Maier et al., 2012]
  - transformations for treebank trees which preserve discontinuity information
  - specialized, much faster parser
- Coarse-to-fine [van Cranenburgh, 2012]
  - build a CFG from LCFRS in which each block gets its own non-terminal
  - use CFG chart as filtering stage for LCFRS parsing
The solutions

- Use \( A^* \) search with outside estimates [Maier and Kallmeyer, 2010]
  - Improve sorting of partial results such that those are processed first which more quickly lead to goal
- Assuring that \( k = 2 \) [Maier et al., 2012]
  - transformations for treebank trees which preserve discontinuity information
  - specialized, much faster parser
- Coarse-to-fine [van Cranenburgh, 2012]
  - build a CFG from LCFRS in which each block gets its own non-terminal
  - use CFG chart as filtering stage for LCFRS parsing
- Decrease in probability (GF)
  - Watch decreases in probability when advancing in the sentence
The solutions

- Use $A^*$ search with outside estimates [Maier and Kallmeyer, 2010]
  - Improve sorting of partial results such that those are processed first which more quickly lead to goal
- Assuring that $k = 2$ [Maier et al., 2012]
  - Transformations for treebank trees which preserve discontinuity information
  - Specialized, much faster parser
- Coarse-to-fine [van Cranenburgh, 2012]
  - Build a CFG from LCFRS in which each block gets its own non-terminal
  - Use CFG chart as filtering stage for LCFRS parsing
- Decrease in probability (GF)
  - Watch decreases in probability when advancing in the sentence

All of these can be combined!
Related work

Related work aiming at producing parse trees with non-local information:

Pre-/post-processing of PCFG parses:
- [Dienes and Dubey, 2003]: Preprocessing: Inject traces in parser input (ML)
- [Cai et al., 2011]: Preprocessing: Inject traces (Lattice)
- [Johnson, 2002]: Postprocessing: Insert traces in postprocessing

Dependency parsing:
- [Hall and Nivre, 2008]: Reconstructing CB via non-projective dependencies

Formalisms directly encoding discontinuities in derived trees:
- [Plaehn, 2004]: First, using Discontinuous Phrase Structure Grammar (DPSG), up to 15 words
- [Levy, 2005]: Comparable setup to rparse, but no results reported
Related work

Related work aiming at producing parse trees with non-local information:

- Pre-/post-processing of PCFG parses:
  - [Dienes and Dubey, 2003]: Preprocessing: Inject traces in parser input (ML)
  - [Cai et al., 2011]: Preprocessing: Inject traces (Lattice)
  - [Johnson, 2002]: Postprocessing: Insert traces in postprocessing

- Dependency parsing:
  - [Hall and Nivre, 2008]: Reconstructing CB via non-projective dependencies

- Formalisms directly encoding discontinuities in derived trees:
  - [Plaehn, 2004]: First, using Discontinuous Phrase Structure Grammar (DPSG), up to 15 words
  - [Levy, 2005]: Comparable setup to rparse, but no results reported
Related work

Related work aiming at producing parse trees with non-local information:

- Pre-/post-processing of PCFG parses:
  - [Dienes and Dubey, 2003]: Preprocessing: Inject traces in parser input (ML)
  - [Cai et al., 2011]: Preprocessing: Inject traces (Lattice)
  - [Johnson, 2002]: Postprocessing: Insert traces in postprocessing

- Dependency parsing:
  - [Hall and Nivre, 2008]: Reconstructing CB via non-projective dependencies
Related work

Related work aiming at producing parse trees with non-local information:

- Pre-/post-processing of PCFG parses:
  - [Dienes and Dubey, 2003]: Preprocessing: Inject traces in parser input (ML)
  - [Cai et al., 2011]: Preprocessing: Inject traces (Lattice)
  - [Johnson, 2002]: Postprocessing: Insert traces in postprocessing

- Dependency parsing:
  - [Hall and Nivre, 2008]: Reconstructing CB via non-projective dependencies

- Formalisms directly encoding discontinuities in derived trees:
  - [Plaehn, 2004]: First, using Discontinuous Phrase Structure Grammar (DPSG), up to 15 words
  - [Levy, 2005]: Comparable setup to rparse, but no results reported
Where to go from here

- More improvements from the PCFG world:
  - *LCFRS-LA* with automatic category splitting
  - Approximations of LCFRS parsing ("beam search") which raise speed while maintaining output quality
- Create more data, e.g. an evaluation suite for discontinuous structures
- Investigate the impact of discontinuous structures in downstream applications
How to get a GF from TIGER

1. Get the TIGER treebank from http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger.html
2. Get rparse from http://phil.hhu.de/rparse, Compile rparse using ant
3. Run rparse with
   java -jar rparse.jar -doTrain -train [TIGERfile] -trainIntervals 1-10 -trainSave [output-dir] -trainSaveFormat gf -trainExtractOnly
4. Check GF files in your output directory
5. Import the concrete syntax into GF
Data-oriented language processing: An overview.

Discontinuity revisited: An improved conversion to context-free representations.
In Proceedings of The Linguistic Annotation Workshop.

Language-independent parsing with empty elements.

Tree-bank grammars.

Head-Driven Statistical Models for Natural Language Parsing.

Antecedent recovery: Experiments with a trace tagger.

PLCFRS parsing of English discontinuous constituents.
In Proceedings of IWPT.

Efficient parsing of well-nested Linear Context-Free Rewriting Systems.
In *Proceedings of HLT-NAACL*.

Parsing discontinuous phrase structure with grammatical functions.

A simple pattern-matching algorithm for recovering empty nodes and their antecedents.

*Parsing beyond Context-Free Grammar.*
Springer.

Data-driven parsing with Probabilistic Linear Context-Free Rewriting Systems.
In *Proceedings of COLING*.

In *Proceedings of NAACL*.

Accurate unlexicalized parsing.

*Probabilistic Models of Word Order and Syntactic Discontinuity.*
Related work

Extract a grammar yourself

Introduction

Data-Driven Parsing with Discontinuous Structures

Going Further


Data-driven plcfrs parsing revisited: Restricting the fan-out to two.
In Proceedings of the Eleventh International Conference on Tree Adjoining Grammars and Related Formalisms (TAG+11), Paris, France.


Discontinuity and non-projectivity: Using mildly context-sensitive formalisms for data-driven parsing.
In Proceedings of TAG+10.


Weighted deductive parsing and Knuth’s algorithm.


Computing the most probable parse for a Discontinuous Phrase-Structure Grammar.


On Multiple Context-Free Grammars.


Efficient parsing with linear context-free rewriting systems.
In Proceedings of EACL.


Discontinuous data-oriented parsing: A mildly context-sensitive all-fragments grammar.
In Proceedings of SPMRL.
Parser evaluation across text types.
In *Proceedings of TLT*. 