Evaluating Evaluation Measures for Minimalist Parsing

Thomas Graf
Stony Brook University
mail@thomasgraf.net
http://thomasgraf.net

Bradley Marcinek
Stony Brook University
bradley.marcinek@stonybrook.edu

CMCL 2014
July 26, 2014
MG parser could yield processing predictions for syntactic proposals that differ on abstract level (e.g. head movement VS remnant movement)

But: need a linking hypothesis/difficulty metric

Is there a **simple metric that is good enough** to distinguish syntactic analyses?

---

**Results**

- Counting number of memorized items insufficient
- **Better:** max time pronounced lexical items stay in memory
Outline

1. Overview of Minimalist Grammars

2. Parsing Minimalist Grammars
   - Stabler’s Top-Down Parser
   - Evaluation Metrics for Processing Predictions

3. Predictions for Processing Difficulty
   - SC/RC vs RC/SC
   - Subject Gaps vs Object Gaps
   - Further Considerations

4. Conclusion
Minimalist Grammars (MGs)

- mildly context-sensitive formalization of Minimalist syntax (Chomsky 1995; Stabler 1997)
  - generates all context-free languages
  - generates some context-sensitive languages
- grammar is fully specified by lexicon
- lexicon = finite set of feature-annotated words
- features trigger structure-building operations Merge and Move
- **Merge**: combine two trees into a new tree
- **Move**: move a subtree of tree $t$ to the left of the root of $t$
Sketch of a Simple Merge Derivation

Phrase structure tree

Derivation tree
Sketch of a Derivation with Move

Phrase structure tree

Derivation tree
A More Readable Variant of Derivation Trees

Derivation tree

```
Move
  Merge
    John
    Merge
      likes
      Merge
        the
        girl
```

“Enhanced” derivation tree

```
VP
  Merge
    VP
      John
      likes
      V
        the
        girl
  DP
```
Why Derivation Trees Matter

- All information encoded in derivation trees
- Derivation trees automatically translated into corresponding phrase structure trees

Phrase structure trees are redundant!
Derivation tree = full description of sentence structure

- **Crucial**: derivation trees are context-free.
- Hence we can build on standard parsing techniques for CFGs.
Why Derivation Trees Matter

- All information encoded in derivation trees
- Derivation trees automatically translated into corresponding phrase structure trees

Phrase structure trees are redundant!
Derivation tree = full description of sentence structure

**Crucial:** derivation trees are context-free.
- Hence we can build on standard parsing techniques for CFGs.
Why Derivation Trees Matter

- All information encoded in derivation trees
- Derivation trees automatically translated into corresponding phrase structure trees

Phrase structure trees are redundant!
Derivation tree = full description of sentence structure

- **Crucial**: derivation trees are *context-free*.
- Hence we can build on standard parsing techniques for CFGs.
Stabler (2011, 2012) presents an MG parser similar to top-down CFG parsers.

**Incremental Top-Down CFG Parser**

- Conjecture start symbol
- If the leftmost symbol is non-terminal apply a matching rewrite rule
- terminal scan first unscanned word of input
- Stop if all non-terminals have been expanded, and all terminals have triggered a scan step, and all words have been scanned
- Return derivation tree
Example Parse of *The girl, John likes*

1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes t
9. Scan *likes*
10. Scan t (= empty string)
Example Parse of *The girl, John likes*

1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V’
7. Scan *John*
8. V’ → likes t
9. Scan *likes*
10. Scan *t* (= empty string)
Example Parse of *The girl, John likes*

1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V'
7. Scan *John*
8. V' → likes t
9. Scan *likes*
10. Scan *t* (= empty string)
Example Parse of *The girl, John likes*

1. Start with VP
2. VP $\rightarrow$ DP VP
3. DP $\rightarrow$ the girl
4. Scan *the*
5. Scan *girl*
6. VP $\rightarrow$ John V'
7. Scan *John*
8. V' $\rightarrow$ likes *t*
9. Scan *likes*
10. Scan *t* (≡ empty string)
Example Parse of *The girl, John likes*

1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V'
7. Scan *John*
8. V' → likes t
9. Scan *likes*
10. Scan *t* (= empty string)
Example Parse of *The girl, John likes*

1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V’
7. Scan *John*
8. V’ → likes t
9. Scan *likes*
10. Scan *t* (= empty string)
Example Parse of *The girl, John likes*

1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V′
7. Scan *John*
8. V′ → likes t
9. Scan *likes*
10. Scan *t* (= empty string)
Example Parse of *The girl, John likes*

1. Start with VP
2. $VP \rightarrow DP \ VP$
3. $DP \rightarrow \text{the girl}$
4. Scan *the*
5. Scan *girl*
6. $VP \rightarrow \text{John } V'$
7. Scan *John*
8. $V' \rightarrow \text{likes } t$
9. Scan *likes*
10. Scan *t* (= empty string)
Example Parse of *The girl, John likes*

1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V'
7. Scan *John*
8. V' → likes t
9. Scan *likes*
10. Scan *t* (= empty string)
Example Parse of *The girl, John likes*

1. Start with VP
2. VP → DP VP
3. DP → the girl
4. Scan *the*
5. Scan *girl*
6. VP → John V’
7. Scan *John*
8. V’ → likes t
9. Scan *likes*
10. Scan *t* (= empty string)
The Problem With Derivation Trees

Derivation trees do not match string order
⇒ left-most terminal ≠ left-most word

Diagram:

```
  VP
   |
  VP
   |
John
   |
V'
   |
likes
   |
DP
   |
the
   |
girl
```

1. Start with Move
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Scan John
   Failure!
The Problem With Derivation Trees

Derivation trees do not match string order
⇒ left-most terminal ≠ left-most word

1. Start with Move
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Scan John
   Failure!
The Problem With Derivation Trees

Derivation trees do not match string order
⇒ left-most terminal ≠ left-most word

Start with Move
2 Move ⇒ Merge
3 Merge ⇒ John Merge
4 Scan John
Failure!
The Problem With Derivation Trees

Derivation trees do not match string order
⇒ left-most terminal ≠ left-most word

1. Start with Move
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Scan John
   Failure!
The Problem With Derivation Trees

Derivation trees do not match string order
⇒ left-most terminal ≠ left-most word

1. Start with Move
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Scan John
   Failure!
Derivation Trees Require Delayed Scanning

Steps must be **delayed** until we have found the leftmost word!

\[ \Rightarrow \text{symbols crossed by mover must be kept in memory} \]

1. Conjecture top-Mover
2. Move $\Rightarrow$ Merge
3. Merge $\Rightarrow$ John Merge
4. Delay Scan *John*
   Merge $\Rightarrow$ likes Merge
5. Delay Scan *likes*
   Merge $\Rightarrow$ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*
Derivation Trees Require Delayed Scanning

Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory

1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ *likes* Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] *girl*
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*
Derivation Trees Require Delayed Scanning

Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory

1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ *likes* Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*
Derivation Trees Require Delayed Scanning

Steps must be **delayed** until we have found the leftmost word!

⇒ symbols crossed by mover must be kept in memory

1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan John Merge ⇒ likes Merge
5. Delay Scan likes Merge ⇒ the[top] girl
6. **Mover found!** Scan the
7. Scan girl
8. Scan John
9. Scan likes
Derivation Trees Require Delayed Scanning

Steps must be **delayed** until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory

1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ *likes* Merge
5. Delay Scan *likes*
   Merge ⇒ *the*[top] *girl*
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*
Derivation Trees Require Delayed Scanning

Steps must be **delayed** until we have found the leftmost word! ⇒ symbols crossed by mover must be kept in memory

1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ *likes* Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] *girl*
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*
Steps must be *delayed* until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory

1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ *likes* Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*
Derivation Trees Require Delayed Scanning

Steps must be 
**delayed** until we have found the leftmost word!

⇒ symbols crossed by mover must be kept in memory

```
1 Conjecture top-Mover
2 Move ⇒ Merge
3 Merge ⇒ John Merge
4 Delay Scan *John*
   Merge ⇒ *likes* Merge
5 Delay Scan *likes*
   Merge ⇒ the[top] girl
6 **Mover found!**
   Scan *the*
7 Scan *girl*
8 Scan *John*
9 Scan *likes*
```
Steps must be delayed until we have found the leftmost word!
⇒ symbols crossed by mover must be kept in memory

1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ *likes* Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] *girl*
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*
Steps must be **delayed** until we have found the leftmost word!

⇒ symbols crossed by mover must be kept in memory

1. Conjecture top-Mover
2. Move ⇒ Merge
3. Merge ⇒ John Merge
4. Delay Scan *John*
   Merge ⇒ *likes* Merge
5. Delay Scan *likes*
   Merge ⇒ the[top] girl
6. **Mover found!**
   Scan *the*
7. Scan *girl*
8. Scan *John*
9. Scan *likes*
Tenure as Linking Hypothesis for Processing

Kobele et al. (2012) link parsing behavior to processing difficulty:

**Tenure**  Time a symbol stays in memory

= **Subscript**—**Superscript**

**Max**  Greatest tenure among all nodes in derivation

<table>
<thead>
<tr>
<th>Max Linking Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Matters for Processing Difficulty</td>
</tr>
<tr>
<td>• <strong>Max</strong> value of the correct derivation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What Doesn’t Matter</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Size of search space/number of conjectured derivations</td>
</tr>
<tr>
<td>• Number of items kept in memory</td>
</tr>
<tr>
<td>• Type of item memorized (e.g. R-expression vs anaphor)</td>
</tr>
<tr>
<td>• Lexical frequency/probabilities</td>
</tr>
</tbody>
</table>
Kobele et al. (2012) link parsing behavior to processing difficulty:

**Tenure** Time a symbol stays in memory

\[
\text{Max} = \text{Subscript} - \text{Superscript}
\]

**Max** Greatest tenure among all nodes in derivation

### Max Linking Hypothesis

**What Matters for Processing Difficulty**

- **Max** value of the correct derivation

**What Doesn’t Matter**

- Size of search space/number of conjectured derivations
- Number of items kept in memory
- Type of item memorized (e.g. R-expression vs anaphor)
- Lexical frequency/probabilities
**Why this is Attractive**

- The MG parser is very simple.
- The linking hypothesis is very simple.
- Nonetheless we get some interesting predictions:
  - Crossing dependencies easier than nested dependencies (Bach et al. 1986)
  - Results can vary with syntactic analysis, for instance head movement VS remnant movement
  ⇒ **processing data differentiates abstract analyses**

**The Big Promise**

- extremely simple processing model (definitely too simple)
  - no number crunching
  - pen and paper is enough
- yet good enough to distinguish between competing proposals from the Minimalist literature
Why this is Attractive

- The MG parser is very simple.
- The linking hypothesis is very simple.
- Nonetheless we get some interesting predictions:
  - Crossing dependencies easier than nested dependencies (Bach et al. 1986)
  - Results can vary with syntactic analysis, for instance head movement VS remnant movement
  ⇒ processing data differentiates abstract analyses

The Big Promise

- extremely simple processing model (definitely too simple)
  - no number crunching
  - pen and paper is enough
- yet good enough to distinguish between competing proposals from the Minimalist literature
Too Good to be True?

Why should Max be the best metric?

MaxLex Max of lexical nodes

Box number of items kept in memory

= number of boxed superscripts

BoxLex number of lexical items kept in memory

± Empty for each metric, another variant that does not count unpronounced nodes

Next Steps

- Pick phenomena that are most likely to be adequately explained by memory limitations
- Mark up correct derivation trees with indices
- See which metric gives best results across the board
Too Good to be True?

Why should Max be the best metric?

MaxLex: Max of lexical nodes

Box: number of items kept in memory
    = number of boxed superscripts

BoxLex: number of lexical items kept in memory

± Empty: for each metric, another variant that does not count unpronounced nodes

Next Steps

- Pick phenomena that are most likely to be adequately explained by memory limitations
- Mark up correct derivation trees with indices
- See which metric gives best results across the board
A sentential complement (SC) containing a relative clause (RC) is easier to parse than an RC containing an SC.

(1) The fact \([SC\, \text{that the employee,} \, RC\, \text{who the manager hired} \, t_i] \text{ stole office supplies}]\) worried the executive.

(2) The executive \([RC\, \text{who the fact} \, SC\, \text{that the employee stole office supplies}] \text{ worried } t_i]\) hired the manager.
Following Kobele et al. we use a promotion analysis of RCs.

- Head noun is merged as argument DP of verb inside RCs
- Head noun moves into Spec,CP of RC

\[
[\text{DP the } [\text{CP } [\text{DP } \varepsilon \text{ employee}] [\text{C' who the manager hired } t_{\text{DP}}]]]
\]

But: Same results with other analyses as long as something moves from within RC to the left of who
SC/RC Derivation

Max 32/32
MaxLex 32/9
Box 9/6
BoxLex 7/4
Max 33/33
MaxLex 33/17
Box 14/11
BoxLex 12/9
Analysis

- **Box** metrics get the contrast.
  - **SC/RC**
    elements of SC preceding RC can be scanned right away, only RC delayed by movement of head noun
  - **RC/SC**
    both RC and SC delayed by movement of head noun

- **Max** metrics give mixed results.
  - **Max**
    - highest value at matrix T-head due to size of subjects
    - both SC/RC and RC/SC yield big subjects
    - difference too small, a single adjective modifying *fact* can tip scale in favor of RC/SC
  - **MaxLex**
    - if only pronounced words are considered, highest value at *who*
    - tenure of *who* increases with distance to head noun
    - RC/SC harder because of increased size of RC
Box metrics get the contrast.

- **SC/RC**
  - elements of SC preceding RC can be scanned right away, only RC delayed by movement of head noun
- **RC/SC**
  - both RC and SC delayed by movement of head noun

Max metrics give mixed results.

**Max**

- highest value at matrix T-head due to size of subjects
- both SC/RC and RC/SC yield big subjects
- difference too small, a single adjective modifying *fact* can tip scale in favor of RC/SC

**MaxLex**

- if only pronounced words are considered, highest value at *who*
- tenure of *who* increases with distance to head noun
- RC/SC harder because of increased size of RC
Subject Gaps vs Object Gaps

An RC containing a subject gap is easier to parse than an RC containing an object gap.

(3) The reporter; \([_{\text{CP}} \text{ who } t_i \text{ attacked the senator}]\) admitted the error.

(4) The reporter; \([_{\text{CP}} \text{ who the senator attacked } t_i]\) admitted the error.
Subject Gap Derivation

Max 19/19
MaxLex 19/7
Box 5/3
BoxLex 3/1
Object Gap Derivation

Max 19/19
MaxLex 19/9
Box 7/5
BoxLex 6/4
Box metrics get the contrast, again.
- object gap leaves more material between landing site and mover
- number of delayed scan steps increases with moved distance

Max metrics give mixed results, again.

Max
- highest value at matrix T-head due to size of subjects
- type of RC has no effect on size of subject
- both derivations must have same maximum tenure

MaxLex
- if only pronounced words are considered, highest value at who
- tenure of who increases with distance to head noun
- object gap harder because of increased distance
Analysis

- Box metrics get the contrast, again.
  - object gap leaves more material between landing site and mover
  - number of delayed scan steps increases with moved distance

- Max metrics give mixed results, again.
  **Max**
  - highest value at matrix T-head due to size of subjects
  - type of RC has no effect on size of subject
  - both derivations must have same maximum tenure

  **MaxLex**
  - if only pronounced words are considered, highest value at *who*
  - tenure of *who* increases with distance to head noun
  - object gap harder because of increased distance
All Metrics are Insufficient

**Box/BoxLex**
- good results for relative clauses
- **Box**: increasing difficulty for all left embedding constructions
- **BoxLex**: constant difficulty for some left embedding
- **But**: do not capture difference between crossing and nested dependencies.

**Max/MaxLex**
- Only **MaxLex** restricted to overt material captures RC contrasts.
- Both capture difference between crossing and nesting.
- **Max**: increasing difficulty for all left embedding constructions
- **MaxLex**: constant difficulty for some left embedding
All Metrics are Insufficient

Box/BoxLex

- good results for relative clauses
- **Box**: increasing difficulty for all left embedding constructions
- **BoxLex**: constant difficulty for some left embedding
- **But**: do not capture difference between crossing and nested dependencies.

Max/MaxLex

- Only **MaxLex** restricted to overt material captures RC contrasts.
- Both capture difference between crossing and nesting.
- **Max**: increasing difficulty for all left embedding constructions
- **MaxLex**: constant difficulty for some left embedding
Next Step: Head-Final RCs

- Even in languages with head-final RCs, subject gaps are preferred.
- This is **not captured** by the metrics. At best we get a tie.
- **Further complication**
  Basque may have a preference for object gaps.
  (Carreiras et al. 2010)
Summary

- **MG derivation trees** allow for very simple top-down parsing
- **Idea:** test syntactic proposals by linking parser behavior to processing difficulty
- **Problem:** Is there a simple yet good enough metric?

<table>
<thead>
<tr>
<th>Phenomenon</th>
<th>Max</th>
<th>MaxLex</th>
<th>Box</th>
<th>BoxLex</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC/RC vs RC/SC</td>
<td>~ / ~</td>
<td>~ /yes</td>
<td>yes/yes</td>
<td>yes/yes</td>
</tr>
<tr>
<td>S-Gap vs O-Gap</td>
<td>no/no</td>
<td>no/yes</td>
<td>yes/yes</td>
<td>yes/yes</td>
</tr>
<tr>
<td>Nesting vs Crossing</td>
<td>yes/yes</td>
<td>yes/yes</td>
<td>no/no</td>
<td>no/no</td>
</tr>
<tr>
<td>Left embedding</td>
<td>no/no</td>
<td>no/ ~</td>
<td>no/no</td>
<td>no/ ~</td>
</tr>
<tr>
<td>Head-Initial RC</td>
<td>no/no</td>
<td>no/no</td>
<td>no/no</td>
<td>no/no</td>
</tr>
</tbody>
</table>

Carreiras, Manuel, Jon Andoni Duñabeitia, Marta Vergara, Irene de la Cruz-Pavía, and Itziar Laka. 2010. Subject relative clauses are not universally easier to process: Evidence from basque. *Cognition* 115:79–92.


