Light Parsing

- Difficulties with full parsing
- Motivations for Parsing
- Light (or "partial") parsing
- Chunk parsing (a type of light parsing)
  - Introduction
  - Advantages
  - Implementations
- REChunkParser

Full Parsing

Goal: build a complete parse tree for a sentence.

- Problems with full parsing:
  - Low Accuracy
  - Slow
  - Domain Specific
- These problems are relevant for both symbolic and statistical parsers.

Full Parsing: Accuracy

Full parsing gives relatively low accuracy
- Exponential solution space
- Dependence on semantic context
- Dependence on pragmatic context
- Long-range dependencies
- Ambiguity
- Errors propagate

Full Parsing: Domain Specificity

Full parsing tends to be domain-specific
- Importance of semantic/lexical context
- Stylistic differences
Full Parsing: Efficiency

Full parsing is very processor-intensive and memory-intensive.
- Exponential solution space
- Large relevant context
  - Long-range dependencies
  - Need to process lexical content of each word
- Too slow to use with very large sources of text (e.g., the web).

Motivations for Parsing

- Why parse sentences in the first place?
- Parsing is usually an intermediate stage
  - Builds structures that are used by later stages of processing
- Full parsing is a sufficient but not necessary intermediate stage for many NLP tasks.
- Parsing often provides more information than we need.

Light Parsing

Goal: assign a partial structure to a sentence.
- Simpler solution space
- Local context
- Less dependance on semantic context
- Non-recursive
- Restricted (local) domain

Output from Light Parsing

- What kind of partial structures should light parsing construct?
- Different structures useful for different tasks:
  - Partial constituent structure
    \[ [\text{np}] [\text{vp}, \text{saw} [\text{np}, \text{a tall man in the park}]]. \]
  - Prosodic segments (phi phrases)
    \[ [\text{i saw}] [\text{a tall man}] [\text{in the park}]. \]
  - Content word groups
    \[ [\text{i}] [\text{saw}] [\text{a tall man}] [\text{in the park}]. \]
Chunk Parsing

Goal: divide a sentence into a sequence of chunks.

- Chunks are non-overlapping regions of a text
  - [I] saw [a tall man] in [the park].
- Chunks are non-recursive
  - a chunk can not contain other chunks
- Chunks are non-exhaustive
  - not all words are included in chunks

Chunk Parsing Examples

- Noun-phrase chunking:
  - [I] saw [a tall man] in [the park].
- Verb-phrase chunking:
  - The man who [was in the park] [saw me].
- Prosodic chunking:
  - [I saw] [a tall man] [in the park].

Chunks and Constituancy

Constituants: [a tall man in [the park]].
Chunks: [a tall man] in [the park].

- Chunks are not constituents
  - Constituents are recursive
- Chunks are typically subsequences of constituents
  - Chunks do not cross constituent boundaries

Chunk Parsing: Accuracy

Chunk parsing achieves higher accuracy

- Smaller solution space
- Less word-order flexibility within chunks than between chunks
- Better locality:
  - Fewer long-range dependancies
  - Less context dependence
- No need to resolve ambiguity
- Less error propagation
Chunk Parsing: Domain Specificity

Chunk parsing is less domain specific

- Dependencies on lexical/semantic information tend to occur at levels "higher" than chunks:
  - Attachment
  - Argument selection
  - Movement
- Fewer stylistic differences within chunks

Chunk Parsing: Efficiency

Chunk parsing is more efficient

- Smaller solution space
- Relevant context is small and local
- Chunks are non-recursive
- Chunk parsing can be implemented with a finite state machine
  - Fast
  - Low memory requirements
- Chunk parsing can be applied to very large text sources (e.g., the web)

Psycholinguistic Motivations

Chunk parsing is psycholinguistically motivated

- Chunks as processing units
  - Humans tend to read texts one chunk at a time
  - Eye-movement tracking studies
- Chunks are phonologically marked
  - Pauses
  - Stress patterns
- Chunking might be a first step in full parsing

Chunk Parsing Techniques

- Chunk parsers usually ignore lexical content
- Only need to look at part-of-speech tags
- Techniques for implementing chart parsing
  - Regular expression matching
  - Chinking
  - Transformational regular expressions
  - Finite state transducers
Regular Expression Matching

- Define a regular expression that matches the sequences of tags in a chunk
  - A simple noun phrase chunk regexp:
    \(<DT>\)? \(<JJ>\)* \(<NN.?>\)
- Chunk all matching subsequences:
  - the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN
  - [the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]
- If matching subsequences overlap, the first one gets priority
- Regular expressions can be cascaded

Chinking

- A chink is a subsequence of the text that is not a chunk.
- Define a regular expression that matches the sequences of tags in a chink
  - A simple chink regexp for finding NP chunks:
    \((<VB.?>|<IN>)+\)
- Chunk anything that is not a matching subsequence:
  - the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN
  - [the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]

Transformational Regular Exprs

- Define regular-expression transformations that add brackets to a string of tags
  - A transformational regexp for NP chunks:
    \((<DT>\)? \(<JJ>\)* \(<NN.?>\)> \{1\}
  - Note: use {} for bracketing because [] has special meaning for regular expressions
- Use the regexp to add brackets to the text:
  - the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN
  - [the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]
- Improper bracketing is an error.

Transformational Regular Exprs (2)

Chinking with transformational regular exprs:

- Put the entire text in one chunk:
  - \(<.*?>\) \{1\}
- Then, add brackets that exclude chinks:
  - \((<VB.?>|<IN>)+\) \{1\}
- Cascade these transformations:
  - the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN
  - [the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]
Transformational Regular Exprs (3)

- Transformational regular expressions can remove brackets added by previous stages:
  \[
  \{<\VB.?> | <\IN>\} \Rightarrow \1
  \]
  \[
  \{\text{the/DT} \} \{\text{little/JJ} \} \{\text{sat/VBD} \} \{\text{on/IN} \} \{\text{the/DT} \} \{\text{mat/NN} \}
  \]
  \[
  \{\text{the/DT} \} \{\text{little/JJ} \} \{\text{sat/VBD} \} \{\text{on/IN} \} \{\text{the/DT} \} \{\text{mat/NN} \}
  \]
  
- Transformational regular expressions can merge two chunks together:
  \[
  (<\DT>|<\JJ>)(?=<(\JJ)|<\NN>)) \Rightarrow \1
  \]
  \[
  \{\text{the/DT} \} \{\text{little/JJ} \} \{\text{sat/VBD} \} \{\text{on/IN} \} \{\text{the/DT} \} \{\text{mat/NN} \}
  \]
  \[
  \{\text{the/DT} \} \{\text{little/JJ} \} \{\text{sat/VBD} \} \{\text{on/IN} \} \{\text{the/DT} \} \{\text{mat/NN} \}
  \]

Finite State Transducers

- A finite state machine that adds bracketing to a text.
- Efficient
  - Other techniques can be implemented using finite state transducers:
    - Matching regular expressions
    - Chinking regular expressions
    - Transformational regular expressions

Evaluating Performance

- Basic measures:
  
<table>
<thead>
<tr>
<th></th>
<th>Target</th>
<th>¬Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selected</td>
<td>True positive</td>
<td>False positive</td>
</tr>
<tr>
<td>¬Selected</td>
<td>False negative</td>
<td>True negative</td>
</tr>
</tbody>
</table>

  - Precision = \(\frac{\text{tp}}{\text{tp} + \text{fp}}\)
  - What proportion of selected items are correct?

  - Recall = \(\frac{\text{tp}}{\text{tp} + \text{fn}}\)
  - What proportion of target items are selected?

REChunkParser

- A regular expression-driven chunk parser
- Chunk rules are defined using transformational regular expressions
- Chunk rules can be cascaded