Roadmap

- Dependency parsing
  - Graph-based dependency parsing
    - Maximum spanning tree
    - CLE Algorithm
    - Learning weights

- Feature-based parsing
  - Motivation
  - Features
  - Unification
Dependency Parse Example

- They hid the letter on the shelf
Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree $T$ for sentence $S$
  - If $S$ is unambiguous, $T$ is the correct parse.
  - If $S$ is ambiguous, $T$ is the highest scoring parse.
Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree $T$ for sentence $S$
  - If $S$ is unambiguous, $T$ is the correct parse.
  - If $S$ is ambiguous, $T$ is the highest scoring parse.

- Where do scores come from?
  - Weights on dependency edges by machine learning
  - Learned from large dependency treebank
Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree $T$ for sentence $S$
  - If $S$ is unambiguous, $T$ is the correct parse.
  - If $S$ is ambiguous, $T$ is the highest scoring parse.

- Where do scores come from?
  - Weights on dependency edges by machine learning
  - Learned from large dependency treebank

- Where are the grammar rules?
Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree $T$ for sentence $S$
  - If $S$ is unambiguous, $T$ is the correct parse.
  - If $S$ is ambiguous, $T$ is the highest scoring parse.

- Where do scores come from?
  - Weights on dependency edges by machine learning
  - Learned from large dependency treebank

- Where are the grammar rules?
  - There aren’t any; data-driven processing
Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree
Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree

- Idea:
  - Build initial graph: fully connected
    - Nodes: words in sentence to parse
Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree

Idea:
- Build initial graph: fully connected
  - Nodes: words in sentence to parse
  - Edges: Directed edges between all words
    - + Edges from ROOT to all words
Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree

Idea:
- Build initial graph: fully connected
  - Nodes: words in sentence to parse
  - Edges: Directed edges between all words
    - + Edges from ROOT to all words
- Identify maximum spanning tree
  - Tree s.t. all nodes are connected
  - Select such tree with highest weight
Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree

Idea:
- Build initial graph: fully connected
  - Nodes: words in sentence to parse
  - Edges: Directed edges between all words
    - Edges from ROOT to all words
- Identify maximum spanning tree
  - Tree s.t. all nodes are connected
  - Select such tree with highest weight
  - Arc-factored model: Weights depend on end nodes & link
    - Weight of tree is sum of participating arcs
Initial Tree

• Sentence: John saw Mary (McDonald et al, 2005)
  • All words connected; ROOT only has outgoing arcs
Initial Tree

- Sentence: John saw Mary (McDonald et al, 2005)
  - All words connected; ROOT only has outgoing arcs
  - Goal: Remove arcs to create a tree covering all words
  - Resulting tree is dependency parse
Maximum Spanning Tree

- McDonald et al, 2005 use variant of Chu-Liu-Edmonds algorithm for MST (CLE)
Maximum Spanning Tree

- McDonald et al, 2005 use variant of Chu-Liu-Edmonds algorithm for MST (CLE)

- Sketch of algorithm:
  - For each node, greedily select incoming arc with max $w$
  - If the resulting set of arcs forms a tree, this is the MST.
  - If not, there must be a cycle.
Maximum Spanning Tree

- McDonald et al, 2005 use variant of Chu-Liu-Edmonds algorithm for MST (CLE)

- Sketch of algorithm:
  - For each node, greedily select incoming arc with max $w$
  - If the resulting set of arcs forms a tree, this is the MST.
  - If not, there must be a cycle.
    - “Contract” the cycle: Treat it as a single vertex
    - Recalculate weights into/out of the new vertex
    - Recursively do MST algorithm on resulting graph
Maximum Spanning Tree

- McDonald et al, 2005 use variant of Chu-Liu-Edmonds algorithm for MST (CLE)

- Sketch of algorithm:
  - For each node, greedily select incoming arc with max w
  - If the resulting set of arcs forms a tree, this is the MST.
  - If not, there must be a cycle.
    - “Contract” the cycle: Treat it as a single vertex
    - Recalculate weights into/out of the new vertex
    - Recursively do MST algorithm on resulting graph

- Running time: naïve: $O(n^3)$; Tarjan: $O(n^2)$
  - Applicable to non-projective graphs
Initial Tree
CLE: Step 1

- Find maximum incoming arcs
CLE: Step 1

- Find maximum incoming arcs
- Is the result a tree?
CLE: Step 1

- Find maximum incoming arcs
  - Is the result a tree?
    - No
  - Is there a cycle?
CLE: Step 1

- Find maximum incoming arcs
  - Is the result a tree?
    - No
  - Is there a cycle?
    - Yes, John/saw
CLE: Step 2

- Since there’s a cycle:
  - Contract cycle & reweight
  - John+saw as single vertex
CLE: Step 2

- Since there’s a cycle:
  - Contract cycle & reweight
  - John+saw as single vertex

- Calculate weights in & out as:
  - Maximum based on internal arcs
  - and original nodes

- Recurse
Calculating Graph

\[ s(\text{Mary, C}) = 11 + 20 = 31 \]
\[ s(\text{ROOT, C}) = 10 + 30 = 40 \]
CLE: Recursive Step

- In new graph, find graph of
- Max weight incoming arc for each word
CLE: Recursive Step

- In new graph, find graph of
  - Max weight incoming arc for each word

- Is it a tree?
CLE: Recursive Step

- In new graph, find graph of
  - Max weight incoming arc for each word

- Is it a tree? Yes!
  - MST, but must recover internal arcs ➔ parse
CLE: Recovering Graph

- Found maximum spanning tree
  - Need to ‘pop’ collapsed nodes
- Expand “ROOT → John+saw” = 40
CLE: Recovering Graph

- Found maximum spanning tree
  - Need to ‘pop’ collapsed nodes
- Expand “ROOT $\rightarrow$ John+saw” = 40
- MST and complete dependency parse
Learning Weights

- Weights for arc-factored model learned from corpus
- Weights learned for tuple \((w_i, w_j, l)\)
Learning Weights

- Weights for arc-factored model learned from corpus
  - Weights learned for tuple \((w_i, w_j, l)\)

- McDonald et al, 2005 employed discriminative ML
  - Perceptron algorithm or large margin variant
Learning Weights

- Weights for arc-factored model learned from corpus
  - Weights learned for tuple \((w_i, L, w_j)\)
- McDonald et al, 2005 employed discriminative ML
  - Perceptron algorithm or large margin variant
- Operates on vector of local features
Features for Learning Weights

- Simple categorical features for \((w_i,L,w_j)\) including:
  - Identity of \(w_i\) (or char 5-gram prefix), POS of \(w_i\)
  - Identity of \(w_j\) (or char 5-gram prefix), POS of \(w_j\)
  - Label of \(L\), direction of \(L\)
  - Sequence of POS tags b/t \(w_i,w_j\)
  - Number of words b/t \(w_i,w_j\)
  - POS tag of \(w_{i+1}\), POS tag of \(w_{i+1}\)
  - POS tag of \(w_{j+1}\), POS tag of \(w_{j+1}\)

- Features conjoined with direction of attachment and distance b/t words
Dependency Parsing

- Dependency grammars:
  - Compactly represent pred-arg structure
  - Lexicalized, localized
  - Natural handling of flexible word order

- Dependency parsing:
  - Conversion to phrase structure trees
  - Graph-based parsing (MST), efficient non-proj $O(n^2)$
  - Transition-based parser
    - MALTparser: very efficient $O(n)$
      - Optimizes local decisions based on many rich features
Features
Roadmap

• Features: Motivation
  • Constraint & compactness

• Features
  • Definitions & representations

• Unification

• Application of features in the grammar
  • Agreement, subcategorization

• Parsing with features & unification
  • Augmenting the Earley parser, unification parsing

• Extensions: Types, inheritance, etc

• Conclusion
Constraints & Compactness

- Constraints in grammar
  - $S \rightarrow NP \ VP$
    - They run.
    - He runs.
Constraints & Compactness

- Constraints in grammar
  - $S \rightarrow NP \ VP$
    - They run.
    - He runs.
  - But...
    - *They runs
    - *He run
    - *He disappeared the flight
Constraints & Compactness

- Constraints in grammar
  - $S \rightarrow NP \ VP$
    - They run.
    - He runs.
  - But...
    - *They runs
    - *He run
    - *He disappeared the flight

- Violate agreement (number), subcategorization
Enforcing Constraints

- Enforcing constraints
Enforcing Constraints

- Enforcing constraints
- Add categories, rules
Enforcing Constraints

- Enforcing constraints
  - Add categories, rules
    - Agreement:
      - S$\rightarrow$ NPsg3p VPsg3p,
      - S$\rightarrow$ NPpl3p VPpl3p,
Enforcing Constraints

- Enforcing constraints
  - Add categories, rules
    - Agreement:
      - $S \rightarrow NP_{sg3p} \ VP_{sg3p}$,
      - $S \rightarrow NP_{pl3p} \ VP_{pl3p}$,
    
    - Subcategorization:
      - $VP \rightarrow V_{trans} \ NP$,
      - $VP \rightarrow V_{intrans}$,
      - $VP \rightarrow V_{ditrans} \ NP \ NP$
Enforcing Constraints

- Enforcing constraints
  - Add categories, rules
    - Agreement:
      - \( S \rightarrow \text{NPsg3p VPsg3p} \),
      - \( S \rightarrow \text{NPpl3p VPpl3p} \),
    - Subcategorization:
      - \( \text{VP} \rightarrow \text{Vtrans NP} \),
      - \( \text{VP} \rightarrow \text{Vintrans} \),
      - \( \text{VP} \rightarrow \text{Vditrans NP NP} \)
  - Explosive!, loses key generalizations
Why features?

- Need compact, general constraints
  - $S \rightarrow NP \ VP$
Why features?

- Need compact, general constraints
  - $S \rightarrow NP \; VP$
    - Only if NP and VP agree
Why features?

- Need compact, general constraints
  - S $\rightarrow$ NP VP
    - Only if NP and VP agree
- How can we describe agreement, subcat?
Why features?

- Need compact, general constraints
  - S \rightarrow NP VP
    - Only if NP and VP agree

- How can we describe agreement, subcat?
  - Decompose into elementary features that must be consistent
  - E.g. Agreement
Why features?

- Need compact, general constraints
  - S \rightarrow NP VP
    - Only if NP and VP agree

- How can we describe agreement, subcat?
  - Decompose into elementary features that must be consistent
  - E.g. Agreement
    - Number, person, gender, etc
Why features?

- Need compact, general constraints
  - S \rightarrow \text{NP VP}
    - Only if NP and VP agree

- How can we describe agreement, subcat?
  - Decompose into elementary features that must be consistent
  - E.g. Agreement
    - Number, person, gender, etc

- Augment CF rules with feature constraints
  - Develop mechanism to enforce consistency
  - Elegant, compact, rich representation
Feature Representations

- Fundamentally, Attribute-Value pairs
  - Values may be symbols or feature structures
    - Feature path: list of features in structure to value
    - “Reentrant feature structures”: share some struct

- Represented as
  - Attribute-value matrix (AVM), or
  - Directed acyclic graph (DAG)
Unification

- Two key roles:
Unification

- Two key roles:
  - Merge compatible feature structures
Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures
Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if
Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if
  - Feature structures are identical
    - Result in same structure
Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if
  - Feature structures are identical
    - Result in same structure
  - Feature structures match where both have values, differ in missing or underspecified
    - Resulting structure incorporates constraints of both
Subsumption

• Relation between feature structures
  • Less specific f.s. subsumes more specific f.s.
  • F.s. F subsumes f.s. G iff
    • For every feature x in F, F(x) subsumes G(x)
    • For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)
Subsumption

- Relation between feature structures
  - Less specific f.s. subsumes more specific f.s.
  - F.s. F subsumes f.s. G iff
    - For every feature x in F, F(x) subsumes G(x)
    - For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)

- Examples:
  - A: [Number SG], B: [Person 3]
  - C:[Number SG]
    - [Person 3]
Subsumption

- Relation between feature structures
  - Less specific f.s. subsumes more specific f.s.
  - F.s. F subsumes f.s. G iff
    - For every feature x in F, F(x) subsumes G(x)
    - For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)

- Examples:
  - A: [Number SG], B: [Person 3]
  - C:[Number SG]
    - [Person 3]
  - A subsumes C; B subsumes C; B,A don’t subsume
    - Partial order on f.s.
Unification Examples

- Identical
- [Number SG] U [Number SG]
Unification Examples

- Identical
  - \([\text{Number SG}] \cup [\text{Number SG}]= [\text{Number SG}]\)

- Underspecified
  - \([\text{Number SG}] \cup [\text{Number [ ]}]\)
Unification Examples

- Identical
  - \([\text{Number SG}] \cup \text{[Number SG]} = [\text{Number SG}]\)

- Underspecified
  - \([\text{Number SG}] \cup \text{[Number []]} = [\text{Number SG}]\)

- Different specification
  - \([\text{Number SG}] \cup \text{[Person 3]}\)
Unification Examples

- Identical
  - \([\text{Number SG}] \cup [\text{Number SG}] = [\text{Number SG}]\)

- Underspecified
  - \([\text{Number SG}] \cup [\text{Number [ ]]} = [\text{Number SG}]\)

- Different specification
  - \([\text{Number SG}] \cup [\text{Person 3}] = [\text{Number SG}]\)
  - \([\text{Number SG}] \cup [\text{Person 3}]\)
  - \([\text{Number SG}] \cup [\text{Number PL}]\)
Unification Examples

- Identical
  - [Number SG] U [Number SG] = [Number SG]

- Underspecified
  - [Number SG] U [Number [ ] ] = [Number SG]

- Different specification
  - [Number SG] U [Person 3] = [Number SG] 
    - [Person 3]

- Mismatched
  - [Number SG] U [Number PL] \rightarrow Fails!
More Unification Examples

\[
\begin{align*}
&\text{AGREEMENT} [1] \quad U \\
&\text{SUBJECT} \quad \text{AGREEMENT} [1] \\
&\text{SUBJECT} \quad \text{AGREEMENT} \\
&\text{PERSON} \quad 3 \quad \text{SG} \\
&\text{AGREEMENT} [1] \\
&\text{SUBJECT} \quad \text{AGREEMENT} [1] \\
&\text{PERSON} \quad 3 \quad \text{SG}
\end{align*}
\]
Features in CFGs: Agreement

- **Goal:**
  - Support agreement of NP/VP, Det Nominal

- **Approach:**
  - Augment CFG rules with features
  - Employ head features
    - Each phrase: VP, NP has head
      - Head: child that provides features to phrase
        - Associates grammatical role with word
        - VP – V; NP – Nom, etc
Agreement with Heads and Features

VP \rightarrow \text{Verb NP}
<VP \text{HEAD}> = <\text{Verb HEAD}>

NP \rightarrow \text{Det Nominal}
<NP \text{HEAD}> = <\text{Nominal HEAD}>
<\text{Det HEAD AGREEMENT}> = <\text{Nominal HEAD AGREEMENT}>

\text{Nominal} \rightarrow \text{Noun}
<\text{Nominal HEAD}> = <\text{Noun HEAD}>

\text{Noun} \rightarrow \text{flights}
<\text{Noun HEAD AGREEMENT NUMBER}> = \text{PL}

\text{Verb} \rightarrow \text{serves}
<\text{Verb HEAD AGREEMENT NUMBER}> = \text{SG}
<\text{Verb HEAD AGREEMENT PERSON}> = 3
Feature Applications

- **Subcategorization:**
  - Verb-Argument constraints
    - Number, type, characteristics of args (e.g. animate)
    - Also adjectives, nouns

- **Long distance dependencies**
  - E.g. filler-gap relations in wh-questions, rel
Implementing Unification

- Data Structure:
  - Extension of the DAG representation
  - Each f.s. has a content field and a pointer field
    - If pointer field is null, content field has the f.s.
    - If pointer field is non-null, it points to actual f.s.
Implementing Unification: II

- Algorithm:
  - Operates on pairs of feature structures
    - Order independent, destructive
  - If fs1 is null, point to fs2
  - If fs2 is null, point to fs1
  - If both are identical, point fs1 to fs2, return fs2
    - Subsequent updates will update both
  - If non-identical atomic values, fail!
Implementing Unification: III

- If non-identical, complex structures
  - Recursively traverse all features of fs2
  - If feature in fs2 is missing in fs1
    - Add to fs1 with value null
  - If all unify, point fs2 to fs1 and return fs1
Example

\[
\begin{align*}
&\text{[AGREEMENT [1]] U [AGREEMENT [PERSON 3]]} \\
&\text{[NUMBER SG] U [PERSON 3]} \\
&\text{[NUMBER SG] U [PERSON 3]} \\
&\text{[PERSON NULL]}
\end{align*}
\]
Unification and the Earley Parser

- Employ constraints to restrict addition to chart
- Actually pretty straightforward
Unification and the Earley Parser

- Employ constraints to restrict addition to chart
  - Actually pretty straightforward
    - Augment rules with feature structure
Unification and the Earley Parser

- Employ constraints to restrict addition to chart
- Actually pretty straightforward
  - Augment rules with feature structure
  - Augment state (chart entries) with DAG
    - Prediction adds DAG from rule
    - Completion applies unification (on copies)
      - Adds entry only if current DAG is NOT subsumed
Conclusion

- Features allow encoding of constraints
  - Enables compact representation of rules
  - Supports natural generalizations

- Unification ensures compatibility of features
  - Integrates easily with existing parsing mech.

- Many unification-based grammatical theories
Unification Parsing

- Abstracts over categories
  - $S \rightarrow NP \; VP \Rightarrow$
    - $X_0 \rightarrow X_1 \; X_2; \; <X_0 \; cat> = S; \; <X_1 \; cat> = NP;$
    - $<X_2 \; cat> = VP$
  - Conjunction:
    - $X_0 \rightarrow X_1 \; and \; X_2; \; <X_1 \; cat> = <X_2 \; cat>;$
    - $<X_0 \; cat> = <X_1 \; cat>$

- Issue: Completer depends on categories

- Solution: Completer looks for DAGs which unify with the just-completed state's DAG
Extensions

- Types and inheritance
  - Issue: generalization across feature structures
    - E.g. many variants of agreement
      - More or less specific: 3rd vs sg vs 3rdsg
  - Approach: Type hierarchy
    - Simple atomic types match literally
    - Multiple inheritance hierarchy
      - Unification of subtypes is most general type that is more specific than two input types
    - Complex types encode legal features, etc