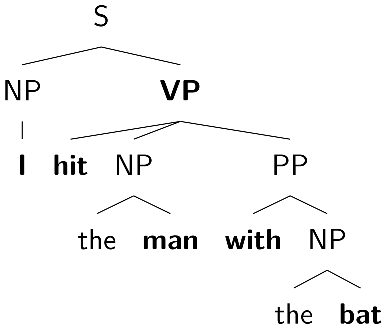
CS 784: Computational Linguistics
Lecture 14: Syntax - Dependency Parsing

Freda Shi

School of Computer Science, University of Waterloo
fhs@uwaterloo.ca

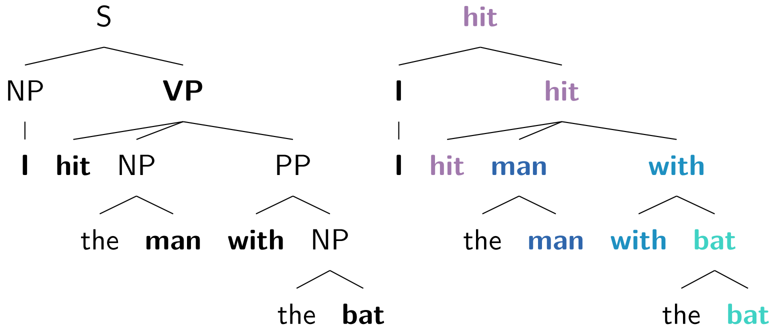
March 6, 2025

From Constituency to Dependency



Boldfaced words: head of the phrase.

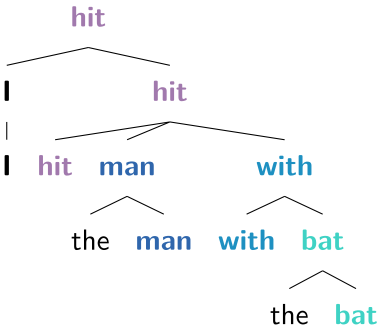
From Constituency to Dependency



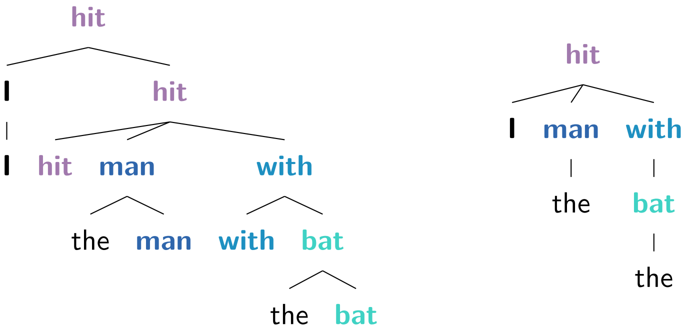
Boldfaced words: head of the phrase.

Propagate the lexical heads up in the tree.

From Constituency to Dependency

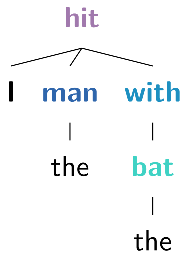


From Constituency to Dependency

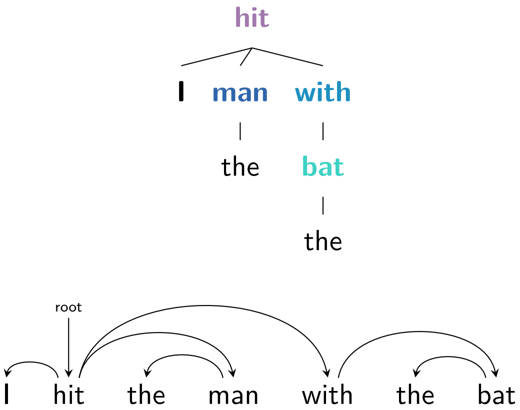


Remove the redundant nodes by keeping the top one.

From Constituency to Dependency

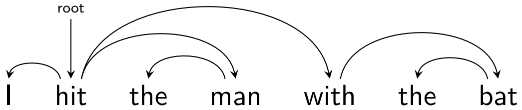


From Constituency to Dependency

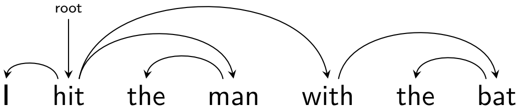


Replace each edge with an arc from the head to the dependent.

Dependency Parse: Properties

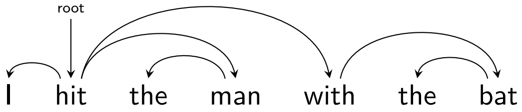


Dependency Parse: Properties



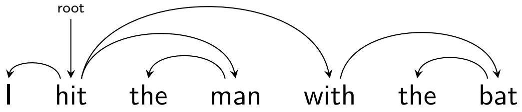
- Each node is a word (in contrast, only leaf nodes are words in constituency trees).

Dependency Parse: Properties



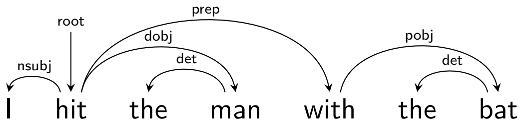
- Each node is a word (in contrast, only leaf nodes are words in constituency trees).
- Each node has at most one parent.

Dependency Parse: Properties



- Each node is a word (in contrast, only leaf nodes are words in constituency trees).
- Each node has at most one parent.
- There is one node that has no parent, called the **root**.

Dependency Parse: Properties



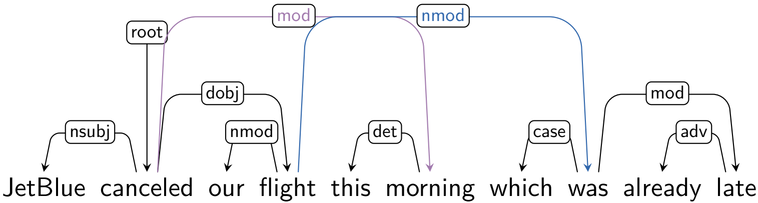
- Each node is a word (in contrast, only leaf nodes are words in constituency trees).
- Each node has at most one parent.
- There is one node that has no parent, called the **root**.
- Each edge can be labeled with a **dependency relation**.

Some Dependency Relations

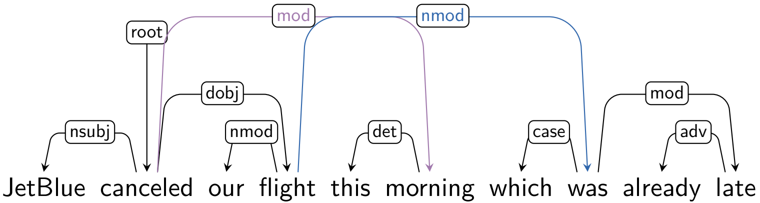
Causal Argument Relations	Description
`nsubj`	Nominal subject
`dobj`	Direct object
`iobj`	Indirect object
`ccomp`	Clausal complement
`xcomp`	Open clausal complement
Modifier Relations	**Description**
`nmod`	Nominal modifier
`amod`	Adjectival modifier
...	

[Source: SLP3]

Projectivity

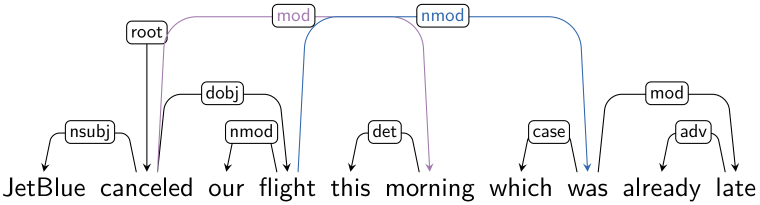


Projectivity



A dependency parse is **nonprojective** if and only if there exist two crossing dependency arcs.

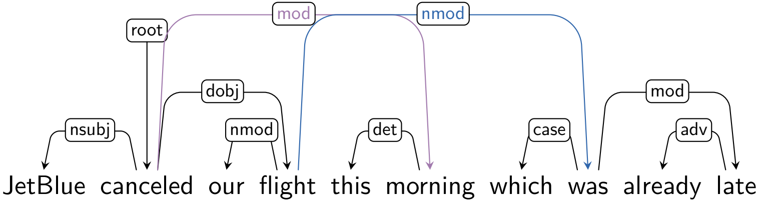
Projectivity



A dependency parse is **nonprojective** if and only if there exist two crossing dependency arcs.

Nonprojective dependency parse tree ⇔ discontinuous constituents in constituency parse tree.

Projectivity



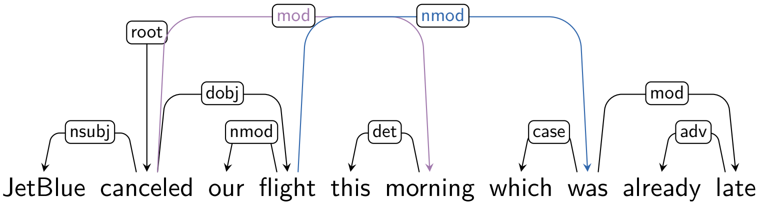
A dependency parse is **nonprojective** if and only if there exist two crossing dependency arcs.

Nonprojective dependency parse tree ⇔ discontinuous constituents in constituency parse tree.

English dependency treebanks are mostly projective.

- When focusing more on semantic relations, it often becomes more nonprojective.

Projectivity



A dependency parse is **nonprojective** if and only if there exist two crossing dependency arcs.

Nonprojective dependency parse tree ⇔ discontinuous constituents in constituency parse tree.

English dependency treebanks are mostly projective.

- When focusing more on semantic relations, it often becomes more nonprojective.

Languages with relatively free word orders, like Czech, are fairly nonprojective.

Universal Dependencies

The Universal Dependencies (UD) project aims to provide a cross-linguistically consistent treebank annotation scheme.

https://universaldependencies.org/

	Nominals	Clauses	Modifier words	Function Words
Core arguments	nsubj obj iobj	csubj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod* discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	**Headless**	**Loose**	**Special**	**Other**
conj cc	fixed flat	list parataxis	compound orphan goeswith reparandum	punct root dep

Universal Dependencies: An Intuitive Example

While detailed grammatical realizations differ across languages, the underlying syntactic structure is often similar.

Universal Dependencies: An Intuitive Example

While detailed grammatical realizations differ across languages, the underlying syntactic structure is often similar.

Shi et al. (2022): multilingual language models enables zero-shot cross-lingual dependency analysis, even for quite different language pairs.

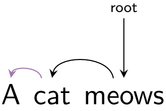
Evaluation of Dependency Parsing

Unlabeled attachment score (UAS): the proportion of words that are assigned the correct head (suppose each word is assigned with one head, and the dummy "root" is considered as a valid head).

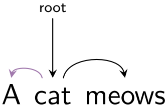
Evaluation of Dependency Parsing

Unlabeled attachment score (UAS): the proportion of words that are assigned the correct head (suppose each word is assigned with one head, and the dummy "root" is considered as a valid head).

$\mathcal{T}_1$:



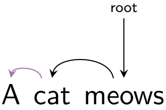
$\mathcal{T}_2$:



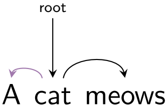
Evaluation of Dependency Parsing

Unlabeled attachment score (UAS): the proportion of words that are assigned the correct head (suppose each word is assigned with one head, and the dummy "root" is considered as a valid head).

$\mathcal{T}_1$:



$\mathcal{T}_2$:



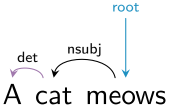
$\text{UAS}(\mathcal{T}_1, \mathcal{T}_2) = \frac{1}{3}$

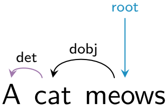
Evaluation of Dependency Parsing

Labeled attachment score (LAS): the proportion of words that are assigned the correct head and the correct dependency relation.

Evaluation of Dependency Parsing

Labeled attachment score (LAS): the proportion of words that are assigned the correct head and the correct dependency relation.

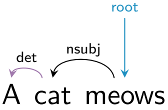
$\mathcal{T}_1$: 

$\mathcal{T}_2$: 

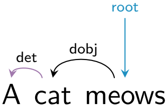
Evaluation of Dependency Parsing

Labeled attachment score (LAS): the proportion of words that are assigned the correct head and the correct dependency relation.

$\mathcal{T}_1$:



$\mathcal{T}_2$:



$\text{LAS}(\mathcal{T}_1, \mathcal{T}_2) = \frac{2}{3}$

Recap: The General NLP Problem Formulation

$$\text{parse}(s) = \arg\max_{\mathcal{Y}} \text{score}(s, \mathcal{Y}; \boldsymbol{\Theta})$$

Recap: The General NLP Problem Formulation

$$\text{parse}(s) = \arg\max_{\mathcal{Y}} \text{score}(s, \mathcal{Y}; \boldsymbol{\Theta})$$

In the dependency parsing context, the score is usually

$$\text{score}(s, \mathcal{Y}; \boldsymbol{\Theta}) = \sum_{w_i \to w_j \in \mathcal{Y}} \text{score}(w_i \to w_j; \boldsymbol{\Theta})$$

Recap: The General NLP Problem Formulation

$$\text{parse}(s) = \arg\max_{\mathcal{Y}} \text{score}(s, \mathcal{Y}; \boldsymbol{\Theta})$$

In the dependency parsing context, the score is usually

$$\text{score}(s, \mathcal{Y}; \boldsymbol{\Theta}) = \sum_{w_i \to w_j \in \mathcal{Y}} \text{score}(w_i \to w_j; \boldsymbol{\Theta})$$

The inference problem: assume we are already given the scores for each possible dependency arc (among the $n \times (n-1)$), how to find the best dependency tree?

Recap: The General NLP Problem Formulation

$$\text{parse}(s) = \arg\max_{\mathcal{Y}} \text{score}(s, \mathcal{Y}; \boldsymbol{\Theta})$$

In the dependency parsing context, the score is usually

$$\text{score}(s, \mathcal{Y}; \boldsymbol{\Theta}) = \sum_{w_i \to w_j \in \mathcal{Y}} \text{score}(w_i \to w_j; \boldsymbol{\Theta})$$

The inference problem: assume we are already given the scores for each possible dependency arc (among the $n \times (n-1)$), how to find the best dependency tree?

If there are no structural constraints, it becomes the problem of **directed minimum spanning tree**.

Recap: The General NLP Problem Formulation

$$\text{parse}(s) = \arg\max_{\mathcal{Y}} \text{score}(s, \mathcal{Y}; \boldsymbol{\Theta})$$

In the dependency parsing context, the score is usually

$$\text{score}(s, \mathcal{Y}; \boldsymbol{\Theta}) = \sum_{w_i \to w_j \in \mathcal{Y}} \text{score}(w_i \to w_j; \boldsymbol{\Theta})$$

The inference problem: assume we are already given the scores for each possible dependency arc (among the $n \times (n-1)$), how to find the best dependency tree?

If there are no structural constraints, it becomes the problem of **directed minimum spanning tree**.

In practice, we also sometimes only consider **projective** trees.

Collins' Algorithm

Assume projectivity and unlabeled arcs—it can be easily extended to labeled arcs by considering an additional dimension.

Collins' Algorithm

Assume projectivity and unlabeled arcs—it can be easily extended to labeled arcs by considering an additional dimension.

Collins (1996): a dynamic programming algorithm for finding the highest scoring projective dependency tree, which shares the spirit with the CKY algorithm for constituency parsing.

Collins' Algorithm

Assume projectivity and unlabeled arcs—it can be easily extended to labeled arcs by considering an additional dimension.

Collins (1996): a dynamic programming algorithm for finding the highest scoring projective dependency tree, which shares the spirit with the CKY algorithm for constituency parsing.

$F[\ell, r, t]$: the highest scoring tree for the span $[\ell, r]$ rooted at index $t (\ell \leq t \leq r)$.

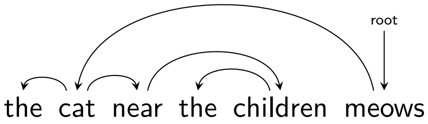
Collins' Algorithm

Assume projectivity and unlabeled arcs—it can be easily extended to labeled arcs by considering an additional dimension.

Collins (1996): a dynamic programming algorithm for finding the highest scoring projective dependency tree, which shares the spirit with the CKY algorithm for constituency parsing.

$F[\ell, r, t]$: the highest scoring tree for the span $[\ell, r]$ rooted at index $t (\ell \leq t \leq r)$.

Example: $F[3, 5, 3]$ is the highest scoring tree for the span $[3, 5]$ (near the children) rooted at index 3 (near).



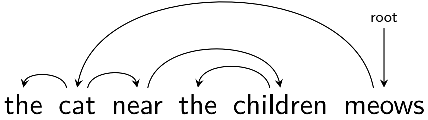
Collins' Algorithm

Assume projectivity and unlabeled arcs—it can be easily extended to labeled arcs by considering an additional dimension.

Collins (1996): a dynamic programming algorithm for finding the highest scoring projective dependency tree, which shares the spirit with the CKY algorithm for constituency parsing.

$F[\ell, r, t]$: the highest scoring tree for the span $[\ell, r]$ rooted at index $t (\ell \leq t \leq r)$.

Example: $F[3, 5, 3]$ is the highest scoring tree for the span $[3, 5]$ (near the children) rooted at index 3 (near).



Implicit assumption: when calculating $F[\ell, r, t]$, we are thinking about the hypothetical condition that $[\ell, r]$ is a **constituent**.

Collins' Algorithm

$$F[\ell, r, t] = \max \begin{cases} \max\limits_{\substack{\ell \leq m < t \\ \ell \leq t_\ell \leq m}} F[\ell, m, t_\ell] + F[m+1, r, t] + \text{score}(t \to t_\ell), \\ \max\limits_{\substack{t \leq m < r \\ m \leq t_r \leq r}} F[\ell, m, t] + F[m+1, r, t_r] + \text{score}(t \to t_r) \end{cases}$$

Collins' Algorithm

$$F[\ell, r, t] = \max \begin{cases} \max_{\substack{\ell \le m < t \\ \ell \le t_\ell \le m}} F[\ell, m, t_\ell] + F[m+1, r, t] + \text{score}(t \rightarrow t_\ell), \\ \max_{\substack{t \le m < r \\ m \le t_r \le r}} F[\ell, m, t] + F[m+1, r, t_r] + \text{score}(t \rightarrow t_r) \end{cases}$$

Key idea: if t is the root of the tree, it must be the root of the left/right subtree as well.

- m: the split point.
- t_ℓ: the head of the left child.
- t_r: the head of the right child.
- $score(t_\ell \rightarrow t_r)$: the score of the arc from t_ℓ to t_r.
- $score(t_r \rightarrow t_\ell)$: the score of the arc from t_r to t_ℓ.

Collins' Algorithm

$$
F[\ell, r, t] = \max \begin{cases} \max\limits_{\substack{\ell \le m < t \\ \ell \le t_\ell \le m}} F[\ell, m, t_\ell] + F[m+1, r, t] + \text{score}(t \to t_\ell), \\ \max\limits_{\substack{t \le m < r \\ m \le t_r \le r}} F[\ell, m, t] + F[m+1, r, t_r] + \text{score}(t \to t_r) \end{cases}
$$

Key idea: if t is the root of the tree, it must be the root of the left/right subtree as well.

- m: the split point.
- t_ℓ: the head of the left child.
- t_r: the head of the right child.
- $score(t_\ell \to t_r)$: the score of the arc from t_ℓ to t_r.
- $score(t_r \to t_\ell)$: the score of the arc from t_r to t_ℓ.

Final Answer: $\max_{1 \le t \le n} F[1, n, t]$.

Collins' Algorithm

$$F[\ell, r, t] = \max \begin{cases} \max\limits_{\substack{\ell \le m < t \\ \ell \le t_\ell \le m}} F[\ell, m, t_\ell] + F[m+1, r, t] + \text{score}(t \to t_\ell), \\ \max\limits_{\substack{t \le m < r \\ m \le t_r \le r}} F[\ell, m, t] + F[m+1, r, t_r] + \text{score}(t \to t_r) \end{cases}$$

Time complexity:

Collins' Algorithm

$$
\begin{aligned}
&F[\ell, r, t] \\
&= \max \begin{cases} \max\limits_{\substack{\ell \le m < t \\ \ell \le t_\ell \le m}} F[\ell, m, t_\ell] + F[m+1, r, t] + \text{score}(t \to t_\ell), \\ \max\limits_{\substack{t \le m < r \\ m \le t_r \le r}} F[\ell, m, t] + F[m+1, r, t_r] + \text{score}(t \to t_r) \end{cases}
\end{aligned}
$$

Time complexity: $\mathcal{O}(n^5)$.

Collins' Algorithm

$$F[\ell, r, t] = \max \begin{cases} \max\limits_{\substack{\ell \leq m < t \\ \ell \leq t_\ell \leq m}} F[\ell, m, t_\ell] + F[m+1, r, t] + \text{score}(t \to t_\ell), \\ \max\limits_{\substack{t \leq m < r \\ m \leq t_r \leq r}} F[\ell, m, t] + F[m+1, r, t_r] + \text{score}(t \to t_r) \end{cases}$$

Time complexity: $\mathcal{O}(n^5)$.

Space complexity:

Collins' Algorithm

$$
\begin{aligned}
& F[\ell, r, t] \\
& = \max \begin{cases} \max\limits_{\substack{\ell \le m < t \\ \ell \le t_\ell \le m}} F[\ell, m, t_\ell] + F[m+1, r, t] + \text{score}(t \to t_\ell), \\ \max\limits_{\substack{t \le m < r \\ m \le t_r \le r}} F[\ell, m, t] + F[m+1, r, t_r] + \text{score}(t \to t_r) \end{cases}
\end{aligned}
$$

Time complexity: $\mathcal{O}(n^5)$.

Space complexity: $\mathcal{O}(n^3)$ to store all $F[\ell, r, t]$.

Collins' Algorithm

$$
\begin{aligned}
&F[\ell, r, t] \\
&= \max \begin{cases} \max\limits_{\substack{\ell \le m < t \\ \ell \le t_\ell \le m}} F[\ell, m, t_\ell] + F[m+1, r, t] + \mathrm{score}(t \to t_\ell), \\ \max\limits_{\substack{t \le m < r \\ m \le t_r \le r}} F[\ell, m, t] + F[m+1, r, t_r] + \mathrm{score}(t \to t_r) \end{cases}
\end{aligned}
$$

Time complexity: $\mathcal{O}(n^5)$.

Space complexity: $\mathcal{O}(n^3)$ to store all $F[\ell, r, t]$.

The Eisner's algorithm (1996) improves the time complexity to $\mathcal{O}(n^3)$ and space complexity to $\mathcal{O}(n^2)$, with some smart realization of the "trapezoid" structure.

Chu-Liu/Edmonds' Algorithm

Proposed independently by Yoeng-Jin Chu and Tseng-Hong Liu (1965) and Jack Edmonds (1967).

- Finds the maximum spanning tree (arborescence) in a directed graph.

Chu-Liu/Edmonds' Algorithm

Proposed independently by Yoeng-Jin Chu and Tseng-Hong Liu (1965) and Jack Edmonds (1967).

- Finds the maximum spanning tree (arborescence) in a directed graph.
- Does not require projectivity constraint.

Chu-Liu/Edmonds' Algorithm

Proposed independently by Yoeng-Jin Chu and Tseng-Hong Liu (1965) and Jack Edmonds (1967).

- Finds the maximum spanning tree (arborescence) in a directed graph.
- Does not require projectivity constraint.
- Algorithm overview:

1. Start with selecting the best incoming edge for each node.
2. If this creates a tree, we're done.

Chu-Liu/Edmonds' Algorithm

Proposed independently by Yoeng-Jin Chu and Tseng-Hong Liu (1965) and Jack Edmonds (1967).

- Finds the maximum spanning tree (arborescence) in a directed graph.
- Does not require projectivity constraint.
- Algorithm overview:

1. Start with selecting the best incoming edge for each node.
2. If this creates a tree, we're done.
3. If there's a cycle, contract it into a single node.

Chu-Liu/Edmonds' Algorithm

Proposed independently by Yoeng-Jin Chu and Tseng-Hong Liu (1965) and Jack Edmonds (1967).

- Finds the maximum spanning tree (arborescence) in a directed graph.
- Does not require projectivity constraint.
- Algorithm overview:
  1. Start with selecting the best incoming edge for each node.
  2. If this creates a tree, we're done.
  3. If there's a cycle, contract it into a single node.
  4. Recalculate edge scores in the contracted graph.

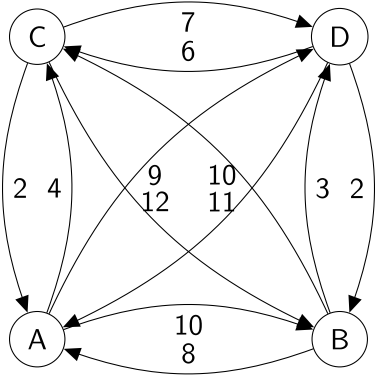
Chu-Liu/Edmonds' Algorithm

Proposed independently by Yoeng-Jin Chu and Tseng-Hong Liu (1965) and Jack Edmonds (1967).

- Finds the maximum spanning tree (arborescence) in a directed graph.
- Does not require projectivity constraint.
- Algorithm overview:
  1. Start with selecting the best incoming edge for each node.
  2. If this creates a tree, we're done.
  3. If there's a cycle, contract it into a single node.
  4. Recalculate edge scores in the contracted graph.
  5. Recursively find the best tree in the new graph.
  6. Expand the contracted node back into a cycle.

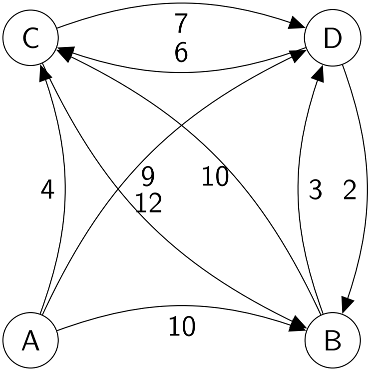
Running Chu-Liu/Edmonds' Algorithm with an Example

Without loss of generality, we assume the root is node A—in practice, we may need to enumerate.



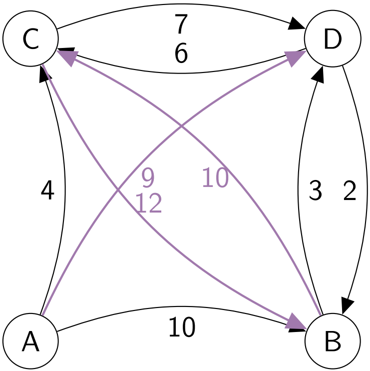
Running Chu-Liu/Edmonds' Algorithm with an Example

Step 1: since we assume A is the root, we remove all incoming edges to A.



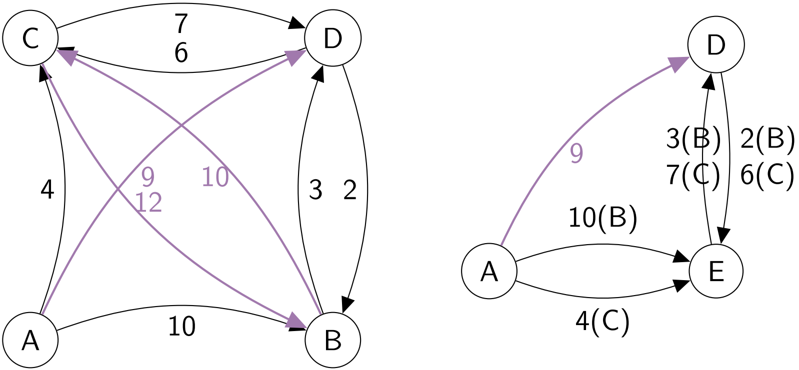
Running Chu-Liu/Edmonds' Algorithm with an Example

Step 2: find the highest scoring incoming edge for each node.



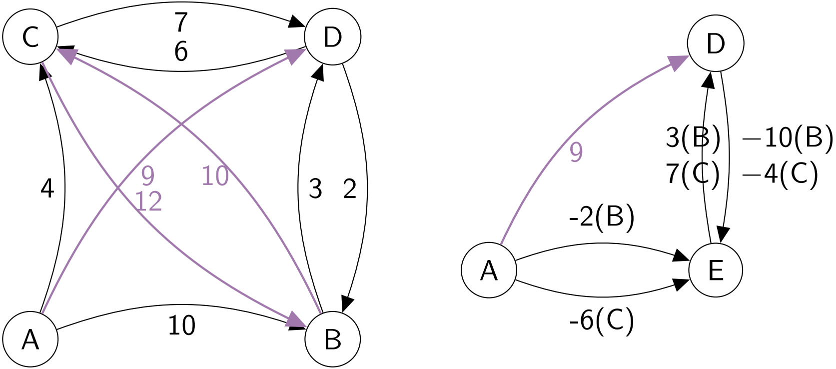
Running Chu-Liu/Edmonds' Algorithm with an Example

Step 3.1: Contract the loop (B-C) into one node, and create a new graph.



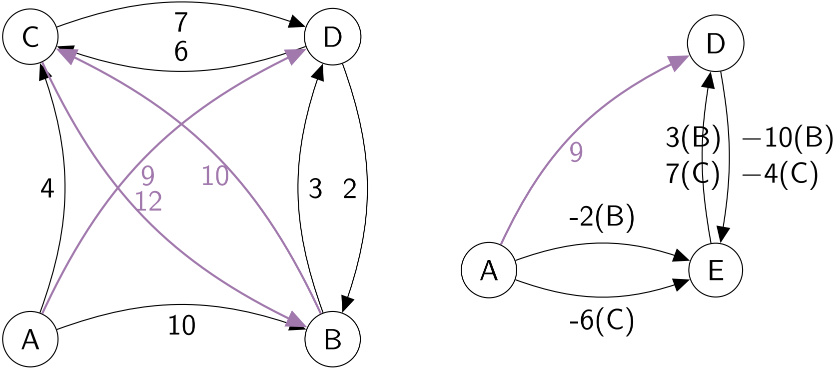
Running Chu-Liu/Edmonds' Algorithm with an Example

Step 3.2: Contract the loop (B-C) into one node, and create a new graph, with the adjusted weights.



Running Chu-Liu/Edmonds' Algorithm with an Example

Step 3.2: Contract the loop (B-C) into one node, and create a new graph, with the adjusted weights.



Repeat the process until we find the maximum spanning tree.

Complexity Analysis

- Time complexity: $O(EV)$ for dense graphs, can be improved to $O(E \log V)$ with optimized implementations.
- Space complexity: $O(E + V)$ to store the sparse graph, or $O(V^2)$ for dense graphs.
- Handles non-projective dependencies.

Complexity Analysis

- Time complexity: $O(EV)$ for dense graphs, can be improved to $O(E \log V)$ with optimized implementations.
- Space complexity: $O(E + V)$ to store the sparse graph, or $O(V^2)$ for dense graphs.
- Handles non-projective dependencies.

When to use:

- Use Collins' (or Eisner's) when you can assume projectivity.
- Use Chu-Liu-Edmonds when non-projective structures are important.
- Many languages with free word order benefit from non-projective parsing.

Neural Dependency Parsers

Take (contextualized) word representations, and predict the scores between each pair of words.

Maximize the ground-truth arc scores in the training set.

Neural Dependency Parsers

Take (contextualized) word representations, and predict the scores between each pair of words.

Maximize the ground-truth arc scores in the training set.

We will still need the inference algorithms on top of the neural scorer to obtain the trees!

Neural Dependency Parsers

Take (contextualized) word representations, and predict the scores between each pair of words.

Maximize the ground-truth arc scores in the training set.

We will still need the inference algorithms on top of the neural scorer to obtain the trees!

See more in Dozat and Manning (2017) for an example of neural dependency parsing.

Neural Dependency Parsers

Take (contextualized) word representations, and predict the scores between each pair of words.

Maximize the ground-truth arc scores in the training set.

We will still need the inference algorithms on top of the neural scorer to obtain the trees!

See more in Dozat and Manning (2017) for an example of neural dependency parsing.

Now: `https://spacy.io/` offers high-quality off-the-shelf dependency parsers.

Next

Semantics: Compositionality, Semantic Role Labeling, Lambda Calculus