

# L95: Natural Language Syntax and Parsing

## 5) Parsing Efficiency

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

## Reminder...

So far we have looked at:

- grammars (PCFG, Dependency, CCG)
- a parsing algorithm (dynamic programming CKY, Shift-Reduce)
- a scoring model for parses (Bayesian, Log-linear)
- an algorithm for finding best parse (Viterbi)

## Yet to come...

Things still to do:

- grammars (Lexicalised PCFG, Unification-Based grammars)
- a parsing algorithm (heuristic algorithms)
- a scoring model for parses ( $A^*$  cost function)
- an algorithm for finding best parse (n-best parses, parse reranking)

Today: **Heuristic search** and the  **$A^*$  algorithm**

## Yet to come...

Things still to do:

- grammars (Lexicalised PCFG, Unification-Based grammars)
- a parsing algorithm (heuristic algorithms)
- a scoring model for parses ( $A^*$  cost function)
- an algorithm for finding best parse (n-best parses, parse reranking)

Today: **Heuristic search** and the  **$A^*$  algorithm**

# CKY is optimal and exhaustive

The basic CKY algorithm will:

- Finds all possible parses
- Is guaranteed to find the best-parse

If we associate partial derivations with a state then:

- finding the best-parse is traversing the states until we find the goal state (state representing  $[S, 0, n]$ )
- notice that basic CKY discovers all possible states in search of the goal state
- a more efficient algorithm might reach the goal state more directly
- the probabilistic (Viterbi-like) version of CKY reduces the number states—**can we be more efficient?**

## CKY is optimal and exhaustive

The basic CKY algorithm will:

- Finds all possible parses
- Is guaranteed to find the best-parse

If we associate partial derivations with a state then:

- finding the best-parse is traversing the states until we find the goal state (state representing  $[S, 0, n]$ )
- notice that basic CKY discovers all possible states in search of the goal state
- a more efficient algorithm might reach the goal state more directly
- the probabilistic (Viterbi-like) version of CKY reduces the number states—**can we be more efficient?**

# The $A^*$ algorithm uses an agenda

- An agenda is a **priority queue**
- States representing partial solutions are added to an **agenda** based on minimising a **cost function**,  $f(n)$
- The first solution state found it is guaranteed to be optimal
- $f(n)$  has two components:
  - $g(n)$  the exact cost of the partial solution at state  $n$
  - $h(n)$  heuristic approximation that **never overestimates** the cost of a solution using of  $n$

# A\* for PSG, Klein and Manning

Search states correspond to edges (partial derivations)  
 $[category, start, end]$

- Initialise with word edges and order by the costs:  
 $[X, i, i + 1]$  and  $cost(X, i, i + 1) = -\log P(w_i, t_i)$
- Loop:
  - Pop the highest priority edge and associated cost from the agenda:  
 $[X, i, j]$  and  $cost(X, i, j)$
  - If highest priority edge represents  $[S, 0, n]$  then stop.
  - Find all compatible neighbours edges:  
e.g. if  $A \rightarrow XY$  exists in the grammar find things like  $[Y, j, k]$
  - Calculate cost of new parent edge:  
 $cost(X, i, j) + cost(Y, j, k) + -\log P(A \rightarrow XY)$
  - Insert parent edge,  $[A, i, k]$ , into the agenda if new or if  $cost(A, i, k)$  has improved.
  - If the agenda is exhausted then fail.



A\* CFG example...

# A\* for CCG, Lewis and Steedman

**function** CCG-ASTAR-PARSE(*words*) **returns** *table* or **failure**

```
supertags ← SUPERTAGGER(words)
for i ← from 1 to LENGTH(words) do
  for all {A | (words[i], A, score) ∈ supertags}
    edge ← MAKEEDGE(i - 1, i, A, score)
    table ← INSERTEDGE(table, edge)
    agenda ← INSERTEDGE(agenda, edge)
loop do
  if EMPTY?(agenda) return failure
  current ← POP(agenda)
  if COMPLETEDPARSE?(current) return table
  table ← INSERTEDGE(chart, edge)
  for each rule in APPLICABLERULES(edge) do
    successor ← APPLY(rule, edge)
    if successor not ∈ agenda or chart
      agenda ← INSERTEDGE(agenda, successor)
    else if successor ∈ agenda with higher cost
      agenda ← REPLACEEDGE(agenda, successor)
```

Pseudo code from Chapter 14 Jurafsky and Martin version 3; based on <https://www.aclweb.org/anthology/D14-1107/>

## $f(n)$ for the $A^*$ CCG is based on the lexicon

- For PCFG the probability of a tree is the product of the probability of the rules that made up the tree.
- For CCG derivation use the product of the probability of the supertags assigned to the words in the derivation (i.e. ignore the rules)
- Formally, given a sentence  $S$  and derivation  $D$  with supertag sequence  $T$ :

$$P(D, S) = P(T, S) = \prod_{i=1}^n P(t_i | w_i)$$

- Convert for cost function:

$$g(w_{1,n}) = \sum_{i=1}^n -\log P(t_i | w_i)$$

## $h(n)$ for the $A^*$ CCG is based on the lexicon

- For  $h(n)$  assume that each word in the outside span will be assigned its most probable supertag (approximates but never overestimates the actual cost)

$$\begin{aligned} f(w_{i:j}, t_{i:j}) &= g(w_{i:j}) + h(w_{i:j}) \\ &= \sum_{k=i}^j -\log P(t_k|w_k) \\ &+ \sum_{k=1}^{i-1} \max_{t \in \text{tags}} (-\log P(t|w_k)) + \sum_{k=j}^n \max_{t \in \text{tags}} (-\log P(t|w_k)) \end{aligned}$$

A\* CCG example...

# What if I need N-best parses?

More in later lectures:

- Extending CKY for n-best parses
- Extending A\* for n-best parses
- Extending shift-reduce for n-best parses
- Discriminative reranking