School of Computing and Information Systems
The University of Melbourne
COMP90042
WEB SEARCH AND TEXT ANALYSIS (Semester 1, 2019)

Sample solutions for discussion exercises: Week 10

Discussion

1. What is **chart parsing**? Why is it important?
   - **Parsing** in general is the process of identifying the structure(s) of a sentence, according to a grammar of the language.
   - In general, the search space is too large to do this efficiently, so we use a dynamic programming method to keep track of partial solutions. The data structure we use for this is a **chart**, where entries in the chart correspond to partial parses (licensed structures) for various spans (sequences of tokens) within the sentence.
2. Consider the following simple **context–free grammar**:

```
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "walked"
NP -> "John" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "cat" | "telescope" | "park"
P -> "on" | "by" | "with"
```

   (a) What changes need to be made to the grammar to make it suitable for **CYK parsing**?
   - For CYK parsing, a grammar needs to be written in Chomsky Normal Form, where each rule consists of either:
     – a (single) non-terminal which re-writes as a single terminal, or
     – a (single) non-terminal which re-writes as exactly two non-terminals
   - Here, we have two rules where a non-terminal re-writes as three non-terminals (`VP -> V NP PP` and `NP -> Det N PP`); we remove these rules and replace them with the following:

```
VP -> V X
X -> NP PP
NP -> Det Y
Y -> N PP
```

   (b) Using the CYK strategy and the above grammar in CNF, parse the following sentences:
   i. "a man saw John"
   ii. "an park by Bob walked an park with Bob"
   iii. "park by the cat with my telescope"
   - This sentence has no parse, because the cell [0,7] doesn't have an `S`.

a	*man*	*saw*	*John*
[0,1] Det	[0,2] NP	[0,3] -	[0,4] S
	[1,2] N	[1,3] -	[1,4] -
		[2,3] V	[2,4] VP
			[3,4] NP

an	*park*	*by*	*Bob*	*walked*	*an*	*park*	*with*	*Bob*
[0,1] Det	[0,2] NP	[0,3] -	[0,4] NP, X	[0,5] -	[0,6] -	[0,7] S	[0,8] -	[0,9] S, S
	[1,2] N	[1,3] -	[1,4] Y	[1,5] -	[1,6] -	[1,7] -	[1,8] -	[1,9] -
		[2,3] P	[2,4] PP	[2,5] -	[2,6] -	[2,7] -	[2,8] -	[2,9] -
			[3,4] NP	[3,5] -	[3,6] -	[3,7] S	[3,8] -	[3,9] S, S
				[4,5] V	[4,6] -	[4,7] VP	[4,8] -	[4,9] VP, VP
					[5,6] Det	[5,7] NP	[5,8] -	[5,9] NP, X
						[6,7] N	[6,8] -	[6,9] Y
							[7,8] P	[7,9] PP
								[8,9] NP

park	*by*	*the*	*cat*	*with*	*my*	*telescope*
[0,1] N	[0,2] -	[0,3] -	[0,4] Y	[0,5] -	[0,6] -	[0,7] Y
	[1,2] P	[1,3] -	[1,4] PP	[1,5] -	[1,6] -	[1,7] PP
		[2,3] Det	[2,4] NP	[2,5] -	[2,6] -	[2,7] NP, X
			[3,4] N	[3,5] -	[3,6] -	[3,7] Y
				[4,5] P	[4,6] -	[4,7] PP
					[5,6] Det	[5,7] NP
						[6,7] N

3. What differentiates **probabilistic parsing** from **chart parsing**? Why is this important? How does this affect the algorithms used for parsing?

   - **Parsing** in general is the process of identifying the structure(s) of a sentence, according to a grammar of the language.
   - In general, the search space is too large to do this efficiently, so we use a dynamic programming method to keep track of partial solutions. The data structure we use for this is a **chart**, where entries in the chart correspond to partial parses (licensed structures) for various spans (sequences of tokens) within the sentence. **Probabilistic parsing** adds real-valued weights (probability) to each production in the grammar, such that parse trees can be assigned a score, namely the product of the probabilities of the productions in the tree. This is important as it allows for discrimination between likely and unlikely parses, rather than just provide a list of all parses, as in standard chart parsing. This affects the algorithms as they need to track the maximum probability analysis for each span, rather than the set of grammar symbols. However the parsing algorithms are very closely related.

4. What is a **probabilistic grammar** and what problem does it attempt to solve?

   - A **probabilistic grammar**, as described above, adds real-valued weights (probability) to each production in the grammar. This attempts to provide a "language model", that is, describe the likely sentences in a language, which is facilitated by their grammatical structure.

5. A hidden Markov model assigns each word in a sentence with a tag, e.g.,

   Donald/NNP has/VBZ small/JJ hands/NNS

   The probability of the sequence is based on the tag-word pairs, and the pairs of adjacent tags. Show how this process can be framed as a CFG, and how the various probabilities (e.g., observation, transition, and initial state) can be assigned to productions. What are the similarities and differences between CYK parsing with this grammar, and the HMM's Viterbi algorithm for finding the best scoring state sequence?

   - The first step is to write the probability assigned by the HMM to the tagged sentence.
   - This can be drawn as a chain with the tag-tag transitions as the "spine", with each observation as a leaf. The probabilities of initial / transitions / observations can be attached to each edge. Overall this gives a right-branching tree.
   - If we treat this tree as a "parse tree", then each clique from the tree (parent and direct children) can be treated as a CFG rule, parent $\rightarrow$ left-child right-child. E.g., NNP $\rightarrow$ Donald VBZ.
   - The score for this rule would be $A_{\text{NNP,VBZ}} \times O_{\text{NNP,Donald}}$. Note that we could use different grammar symbols, such that each production maps to a single HMM component. E.g., split the above rule to NNP′ $\rightarrow$ NNP VBZ ($A_{\text{NNP,VBZ}}$); and NNP $\rightarrow$ Donald ($O_{\text{NNP,Donald}}$); where the "prime" version of the tag is a newly introduced grammar symbol.

- CYK parsing under this grammar will do the same as Viterbi in the HMM, but will waste effort assigning analysis to all word spans in the sentence, while Viterbi effectively only considers spans that start at the first word of the sentence.