# N3

## 1) Length factors

1.1) **Sentence length**: It's well known that parsing systems (local and global) tend to have lower accuracies for longer sentences [1]. This can be explained by the fact that every word has a certain probability to be attached to a wrong head, so more words might increase the total error.

On the other hand, the longer sentences mean more features that we can use to parse sentences. It's reasonable to assume that meaningful context and corresponding features can be beneficial for accurate parsing.

1.2) **Dependency length**: In [1] the authors point out that MaltParser (variation of local parsers) has lower precision than MST parser, while both parsers perform similarly recall-wise for different dependency length.

The lower precision for local models (parsers) can be explained by the fact that shorter arcs are created before longer arcs in the greedy way and less prone to error propagation when short dependencies are present.

Finally, the authors state that both systems should have similar accuracy.

## 2) Graph factors

2.1) **Distance from the sentence root**: In [1] the authors show that global parser performes worse when the distance from the root increases. This can be precision-wise → explained by the fact that words that have failed to get a head word are attached to root, which can be grammatically correct for commas and shallow words. Recall-wise those models are close to identical.

The local parser processing far from root words last while global can process those in the beginning. So there might be a higher chance that far away from root node will be attached to non-root nodes in global parsing.

Finally, since the local model most likely will parse far words in the end, it can use reach set of dynamic features, such as already set dependencies for other words, to guide the decision of the next state to produce more accurate results.

2.2) **Number of sibling nodes**: I believe that the performance will be very similar as none of those parsers has any advantage over another one. This argument is supported by finding in [1]. On the other hand dynamic dependency features could be beneficial for local parsers.

2.3)

2.4) **Degree of non-projectivity**: Since local parsers produce only projective trees, this non-projectivity will make local parser (arc-eager) unable to produce correct parses, and the accuracy of global parser will be higher.

It is also reasonable to assume that the higher the degree will be, the more divergent trees local parser will return and lower accuracy will have.

3) Linguistic factors

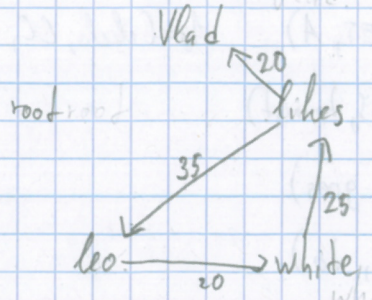
3.1) Part of speech: Since we use greedy search for both parsers, there will no explicit advantage of one over another. But since POS tags can be used as features, it's reasonable to assume that the parser that considers richer range of features will be more accurate, i.e. the one that considers POS tag features.

3.2) Dependency type: local parser can use dependency types as features to guide decisions to next states, so it will have an advantage over disambig. objects and subjects. While global parsers are better at root dependency types. [1].

[1] Characterizing the Errors of Data-Driven Dependency Parsing Models, Ryan McDonald, Joachim Nivre, 2013

## N1.1

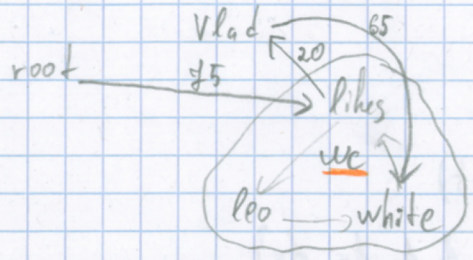
1)




We greedily choose maximum incoming with highest value.

Since the result is not a tree as there is a cycle, we contract: likes, leo, and white into one node, and move to step 2.

2)




$ep(w_c, Vlad)$ = likes ← outgoing arc from $w_c$

$ep(root, w_c)$ = argmax (incoming arc)
1) 0 − 35 = −35 (leo)
2) 20 − 25 = −5 (likes)
3) 0 − 20 = −20 (white)
= likes

$ep(Vlad, w_c)$ = argmax
1) 10 − 35 = −25 (leo)
2) 5 − 25 = −20 (likes)
3) 5 − 20 = −15 (white)
= white

Now we compute arc values:

$\lambda(root, w_c) = 20 - 25 + 80 = 75$

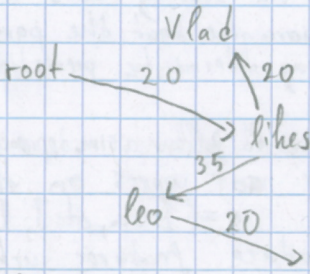
$\lambda(Vlad, w_c) = 5 - 20 + 80 = 65$

Once computed we return and make a recursive call given new node and arcs.

Note that ep is used like a backpointer so we know where exactly point incoming arcs.

3)


4) Since the current graph does not have any cycles, we expand the contracted node and find final arcs.




## N1.2

1) It will be likely to parse both sentences if we consider more features, such as adjacent words to a dependent one. Therefore, the parsing problem becomes:

$$h(S, \Gamma, \lambda) = \underset{G=(V,A)\in G_S}{\operatorname{argmax}} \sum_{(w_i, r, w_j)\in A} \lambda(w_i, r, w_j) = \underset{G=(V,A)\in G_S}{\operatorname{argmax}} \sum_{(w_i, r, w_j)\in A} \underline{\theta} \cdot \underline{g}(w_i, r, w_j, w_{j+1})$$

(right adj. word)

(the notation is from McDonald's book p 56).

We will be interested in scoring low ars (ate, adv, with) if "tuna" is adj. to "with" (adjacent to the right) and high if adj. word is friend.

$\underline{\theta}$ - a vector of parameters

$\underline{g}(w_i, r, w_j, w_{j+1})$ - is a global (binary) feature function that will consider features like:

1) $w_i$ = salad, $r$ = modif, $w_j$ = with, $w_{j+1}$ = tuna
2) $w_i$ = ate, $r$ = adj, $w_j$ = with, $w_{j+1}$ = friends

## N 2.1

stack (↓) Buffer (↓)

1) SH ([root, A], [tourist, ...], {})  $A_1$ = (tourist, DET, A)  $A_6$ = (clubs, CC, and)
2) $LA_{DET}$ ([root], [tourist, ...], {$A_1$})  $A_2$ = (goes, NSUBJ, tourist)
3) SH ([root, tourist], [goes, ...], {$A_1$})  $A_3$ = (root, ROOT, goes)
4) $LA_{NSUBJ}$ ([root], [goes, ...], {$A_1$, $A_2$})  $A_4$ = (clubs, CASE, to)
5) $RA_{root}$ ([root, goes], [to, ...], {$A_1$, $A_2$, $A_3$})  $A_5$ = (goes, NMOD, clubs)
6) SH ([root, goes, to], [clubs, ...], {$A_1$, $A_2$, $A_3$})
7) $LA_{CASE}$ ([root, goes], [clubs, ...], {$A_1$, $A_2$, $A_3$, $A_4$})
8) $RA_{NMOD}$ ([root, goes, clubs], [and, ...], {$A_1$, $A_2$, $A_3$, $A_4$, $A_5$})
9) $RA_{CC}$ ([root, goes, clubs, and], [parties, hard], {$A_1$, $A_2$, $A_3$, $A_4$, $A_5$, $A_6$})

⇓ should have been (i.e. $RA_{CC}$ is a mistake)

9) RD ([root, goes], [parties, hard], {$A_1$:5})

## N 2.2

Assuming that we still use an oracle based parsing, we will end up having multiple feasible trees, as the oracle will rely on the beam search and return multiple candidate transitions. Therefore, in the end we will need a way to find the correct tree among obtained candidates. If the scoring does not consider adjacent words to the dependent ones, we will can't guarantee that the parser will return the correct tree. This is less efficient that making deterministic predictions of next states via classifier.

To obtain correct tree via classifier we could follow a sim. approach as in 1.2 and introduce feature functions that consider adj. words on the buffer. In our case that would be configuration: c = ([..., ate], [with, tuna], ...) that has wrong transition. We could introduce features such as below to make the classifier predict transitions correctly

$f_1$: BUF[0] == with, BUF[1] == tuna, STK[0] == ate.

Obviously the corresponding parameters for each feature has to be trained in order for us to make correct predictions. The deterministic parser with a classifier is shown in Fig 3.4 MacDonald's book