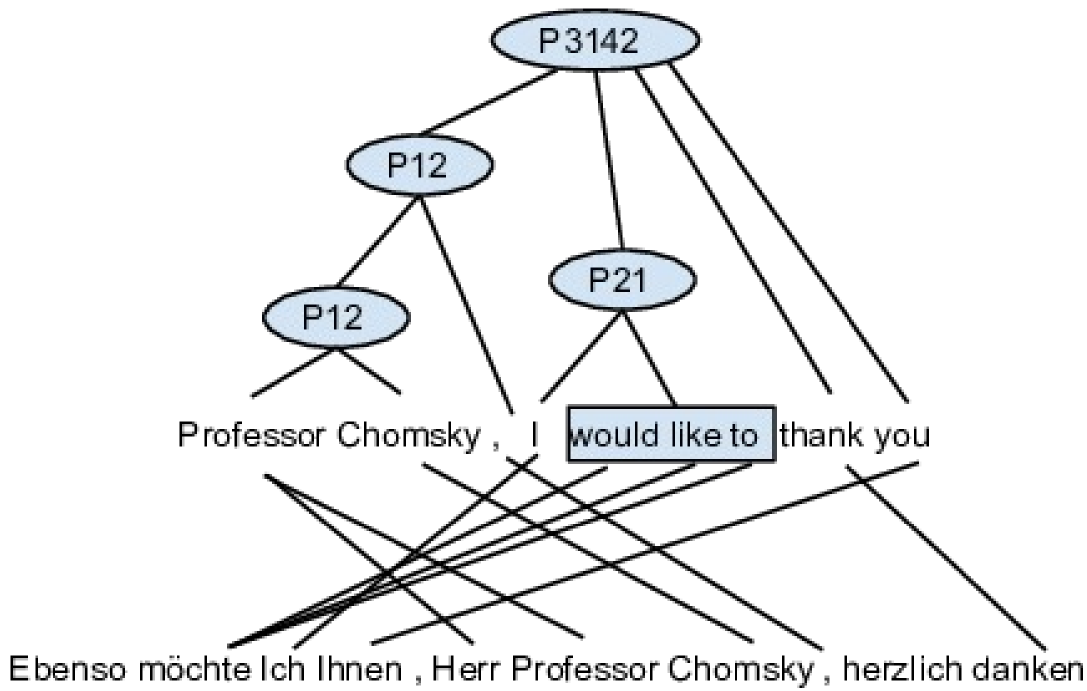
Reordering Grammar

Miloš Stanojević

PETs as a representation of reordering patterns



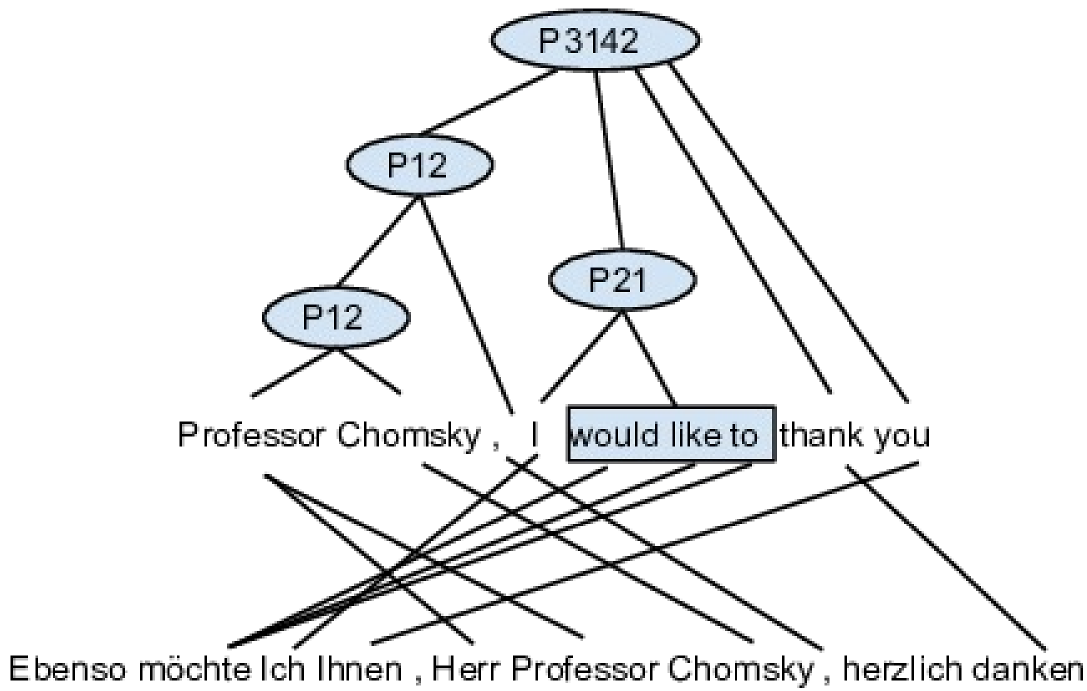
We've already seen few papers that use ITG for preordering (Tromble and Eisner, Neubig ...).

ITG as a representation of reordering has few restructions:
- only permutation
- only binarizable permutations

We try to solve this two problems with:
- permutation trees
- minimal phrases

Just like previous models we are making a parsing model which predicts these “reordering trees” before translation.

PETs as a representation of reordering patterns



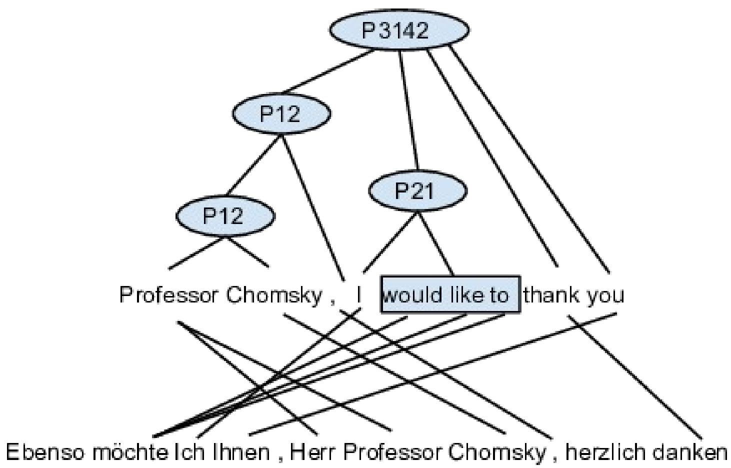
Let's pretend this is a standard parsing task.

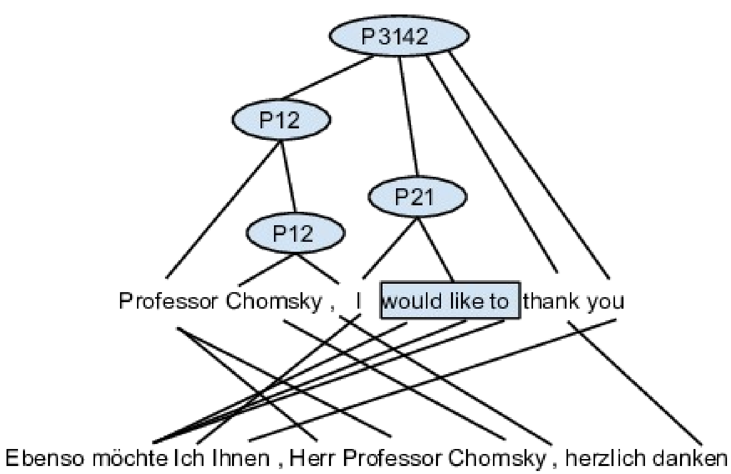
How would we learn a parsing model?

Would it be good enough?

What could be potential problems?

PETs as a representation of reordering patterns





Let's pretend this is a standard parsing task.

How would we learn a parsing model?

Would it be good enough?

What could be potential problems?

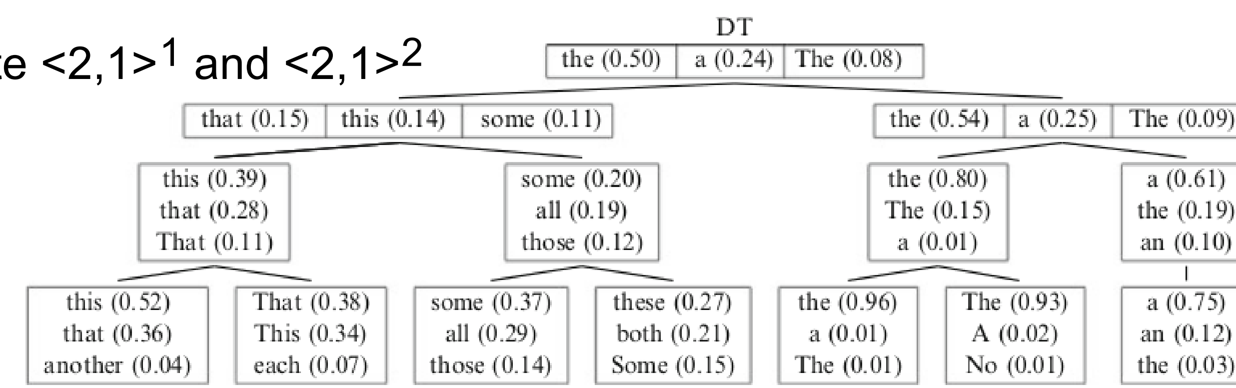
- **Labels are too abstract**
  - Lexicalization
  - Label splitting
- **Many PETs per permutation**
  - In training we have (exponentially) many trees per permutation
  - In testing we need to sum over many trees per permutation

Inducing Reordering Grammar

- We could see that this is an unsupervised (or at least partially supervised) task:
  - We don't know exact trees from which to learn but just have constraints of what are not possible trees
  - We know how do we reorder but we don't know the cause (no linguistic cues). We need to find these more specific labels that explain the “cause”

Let's talk about Latent Variable parsing

- Why do it?
  - Labels that are visible are actually not specific enough. For example:
    - There is a difference between NP subject and NP object
    - a subject NP is 8.7 times more likely than an object NP to expand as just a pronoun
  - In our case this problem is even more extreme: instead of labels like NP we have <1,2> and <2,1> which hide behind themself the reason why they are doing that operation for example
    - <2,1> could mean
      - “I am doing inversion because on this span there is a verb phrase” or it could also be
      - “I am doing inversion because on this span there is a subject”...
    - Which one is right? We don't know, but we know that these kinds of things exist and they are hidden
    - That's why we create <2,1>^{1} and <2,1>^{2}

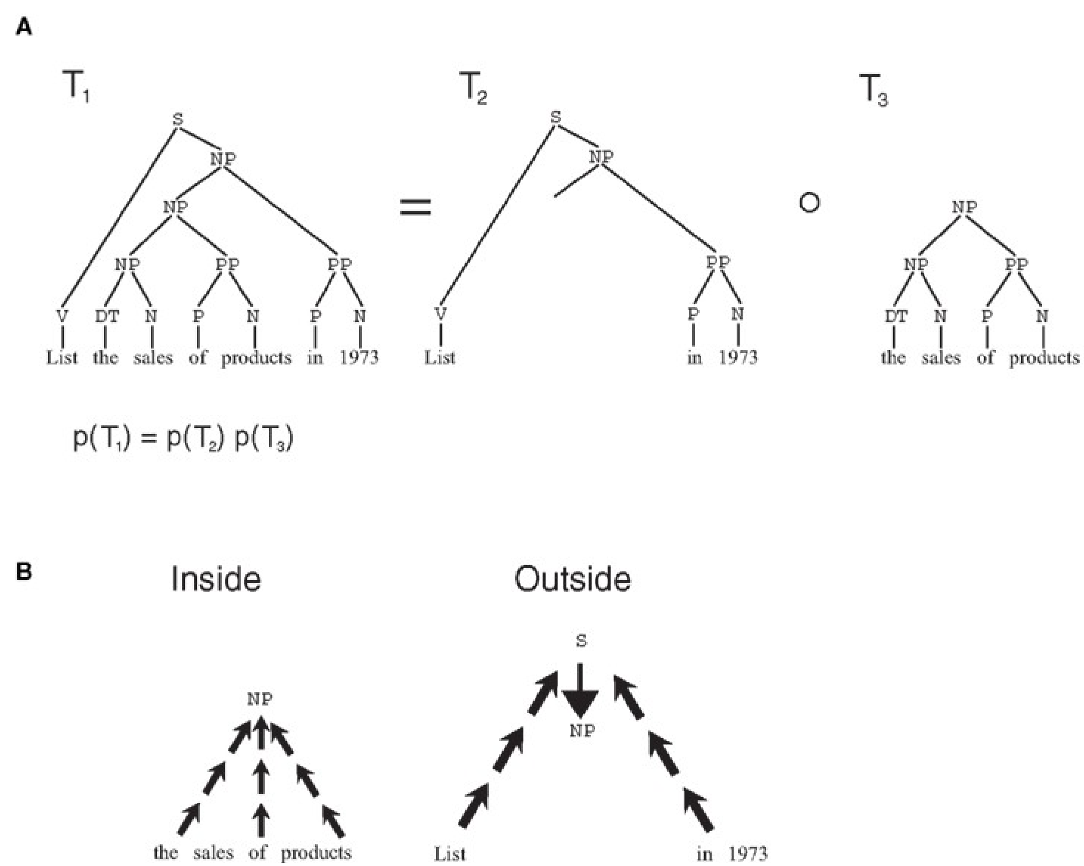


Let's talk about Latent Variable parsing

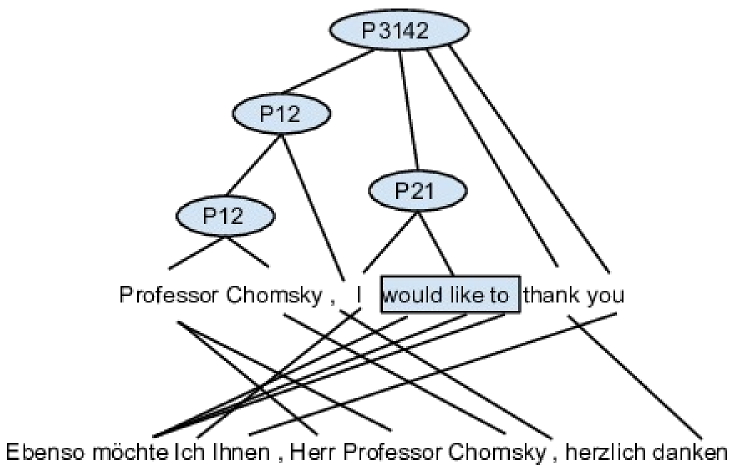
- How is it done?
  - EM (Expectation Maximization)
  - In standard setting we collect counts and then minimize
  - Because counts are hidden we instead collect expected counts and normalize them (for several iterations)
  - So if we have only two trees T1 and T2 with probability 0.1 and 0.2 which contain some rule what would be the expected count of that rule?

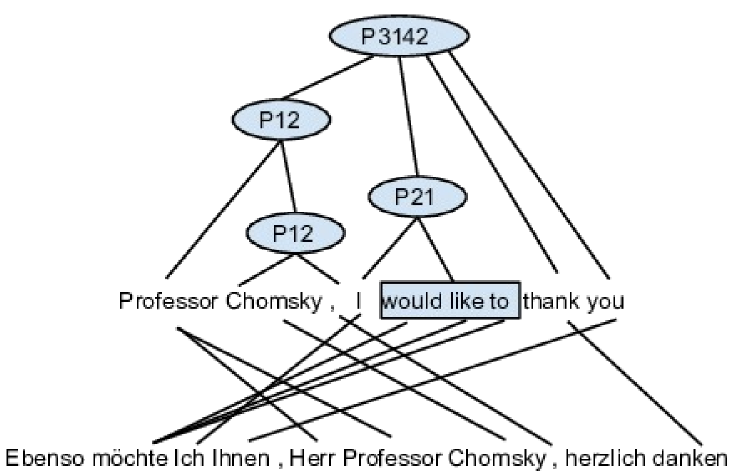
Summing all trees

- Tricky part is how to sum over all trees of which there could be exponentially many
- We use dynamic programming – Inside-Outside



Our case vs. monolingual parsing



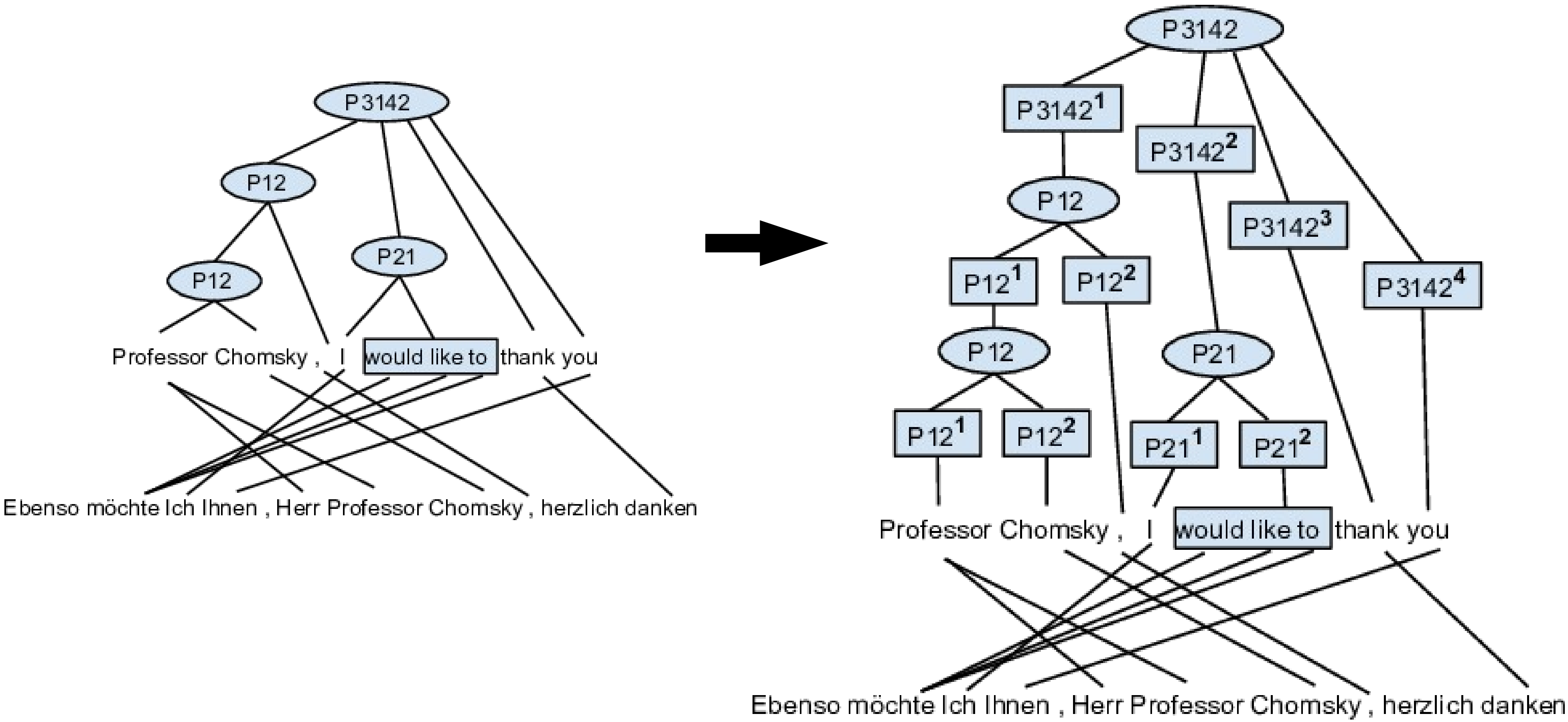


- In monolingual parsing we know the bracketing while in our case even that is uncertain
- Luckily Inside-Outside easily covers this case too

Ok, so we split the non-terminals Is that all?

- Let's say we have a rule
- P2413 -> P12 P21 P12 P21
- And we split every non-terminal.
- What could be the problem?
- Imagine we split every non-terminal into 30 new non-terminals.

“Unarization”



$30^6 = 729000000$

$30+30^2*5 = 4530$

Some details

- We convert alignments to permutations by using minimal phrases where necessary
- Rare words (count<3) are replaced with "UNKNOWN" token
- Unaligned words make use of operator P01 and P10

Now that we have a grammar how do we decode?

- Standard CKY+ to build the chart of all trees
- We can do Viterbi to get the best tree.
  - Is that a good idea?
- We want to find a permutation with highest probability and not the derivation with highest probability
- For computing the probability of permutation we want to sum:
  - Over all non-terminals
  - Over all bracketings
- that produce the same permutation
- Exact solution is NP-complete.
  - What can we do to approximate it?

Sampling

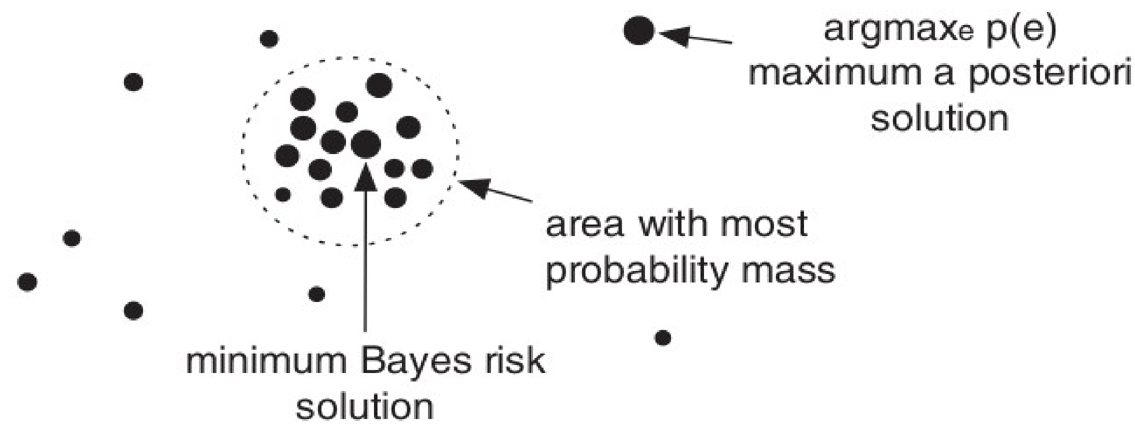
- We sample lots of trees (10000) and then compute their probability by relative frequency
- Problem: space is huuuuge so most of the trees appear only once which makes distribution unreliable
- Still, the distribution says something
- If by sampling we get these three permutations what can we guess from them:
  - 4 3 2 1 6 5
  - 4 2 3 1 6 5
  - 3 2 4 1 5 6

We incorporate this "similarity" in our decision rule

- Instead of taking the most probable permutation we take "the least risky one" under some loss (or similarity) function

$$\hat{\pi} = \underset{\pi}{\operatorname{argmin}} \sum_{\pi_r} Loss(\pi, \pi_r) P(\pi_r)$$

- Used very often in many structure prediction tasks (for example machine translation with BLEU loss)



- Imagine we have n samples of permutations. Length of each permutation is k. What is the complexity of this algorithm if our loss function is Kendall tau?

Fast MBR with linear loss

$$Kendall(\pi, \pi_r) = \sum_b \frac{1 - \delta(\pi, b)}{\frac{k(k-1)}{2}} \delta(\pi_r, b)$$

$$\hat{\pi} = \underset{\pi}{\operatorname{argmin}} \sum_{\pi_r} \sum_b \frac{1 - \delta(\pi, b)}{\frac{k(k-1)}{2}} \delta(\pi_r, b) P(\pi_r)$$

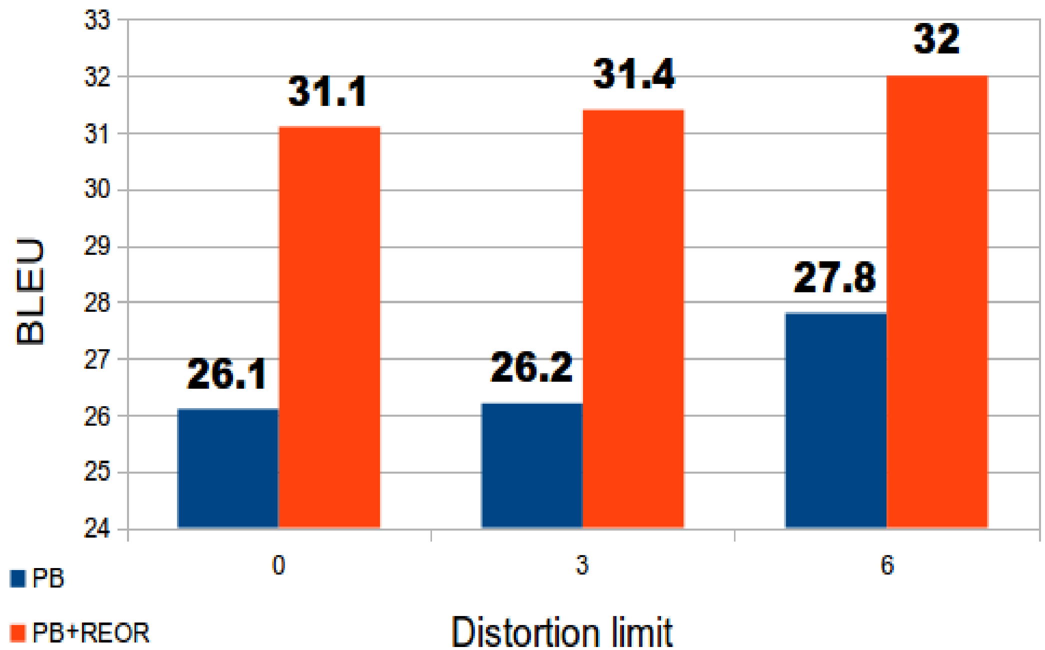
$$= \underset{\pi}{\operatorname{argmin}} \sum_b (1 - \delta(\pi, b)) \left[\sum_{\pi_r} \delta(\pi_r, b) P(\pi_r) \right]$$

$$= \underset{\pi}{\operatorname{argmin}} \sum_b (1 - \delta(\pi, b)) \mathbb{E}_{P(\pi_r)} \delta(\pi_r, b)$$

$$= \underset{\pi}{\operatorname{argmax}} \sum_b \delta(\pi, b) \mathbb{E}_{P(\pi_r)} \delta(\pi_r, b)$$

What is the complexity now?

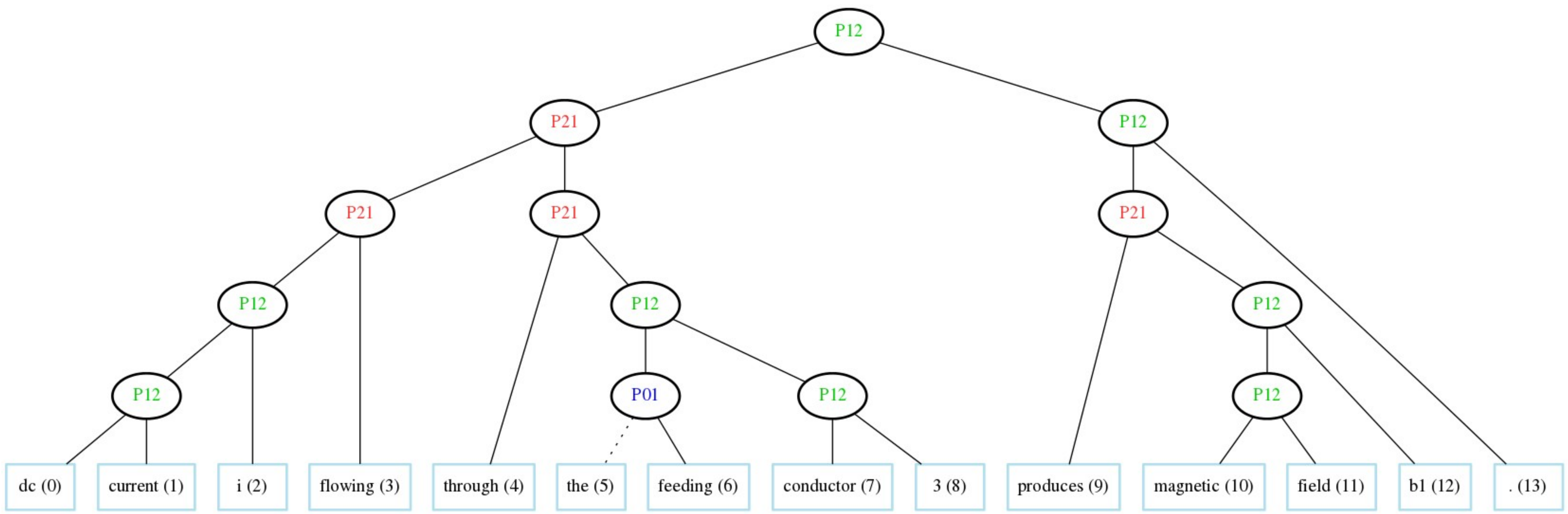
Results with English-Japanese



Metric	**System**	**Avg**	***p*-value**
BLEU ↑	PB MSD	29.6	-
	PB MSD + REOR	32.4	0.00
METEOR ↑	PB MSD	50.1	-
	PB MSD + REOR	51.3	0.00
TER ↓	PB MSD	58.0	-
	PB MSD + REOR	55.3	0.00

Table 3: Phrase-Based MSD with/out preordering

Results with English-Japanese



How does it compare to other work we've seen before?

How does it compare to other work we've seen before?

- Does not have two steps bud does **everything in one go**:
  - Predict the brackets (tree structure)
  - Predict the labels on the tree
- Does not pick only one tree but considers them **all trees** both during training and during testing
- It is **fully probabilistic** (no perceptrons and similar stuff)
- Uses no syntax (unlike Dyer) and captures **syntactically non-isomorphic** reordering patterns
- Can capture **non-ITG reorderings**
- Basically **standard parsing** (use can extend it with all the standard stuff you might use in parsing).
- Suggestions on how it could be improved?
- Questions?