Transformational Priors Over Grammars

Jason Eisner
Johns Hopkins University

July 6, 2002 — EMNLP

This talk is called "Transformational Priors Over Grammars."

It should become clear what I mean by a prior over grammars, and where the transformations come in.

But here's the big concept:

The Big Concept

- Want to parse (or build a syntactic language model).
- Must estimate rule probabilities.
- **Problem:** Too many possible rules!
  - Especially with lexicalization and flattening (which help).
  - So it's hard to estimate probabilities.

Suppose we want to estimate probabilities of parse trees, either to pick the best one or to do language modeling.

Then we have to estimate the probabilities of context free rules.

But the problem, as usual, is sparse data – since there are too many rules, too many probabilities to estimate.

This is especially true if we use lexicalized rules, especially "flat" ones where all the dependents attach at one go.

It does help to use such rules, as we'll see, but it also increases the number of parameters.

The Big Concept

- **Problem:** Too many rules!
  - Especially with lexicalization and flattening (which help).
  - So it's hard to estimate probabilities.
- **Solution:** Related rules tend to have related probs
  - *POSSIBLE* relationships are given a priori
  - *LEARN* which relationships are strong in this language

    (just like feature selection)
- Method has connections to:
  - Parameterized finite-state machines (Monday's talk)
  - Bayesian networks (inference, abduction, explaining away)
  - Linguistic theory (transformations, metarules, etc.)

Solution, I think, is to realize that related rules tend to have related probabilities.

Then if you don't have enough data to observe a rule's probability directly, you can estimate it by looking at other, related rules.

It's a form of smoothing. Sort of like reducing the number of parameters, although actually I'm going to keep all the parameters in case the data aren't sparse, and use a prior to bias their values in case the data are sparse.

OLD: This is like reducing the number of parameters, since it lets you predict a rule's probability instead of learning it.

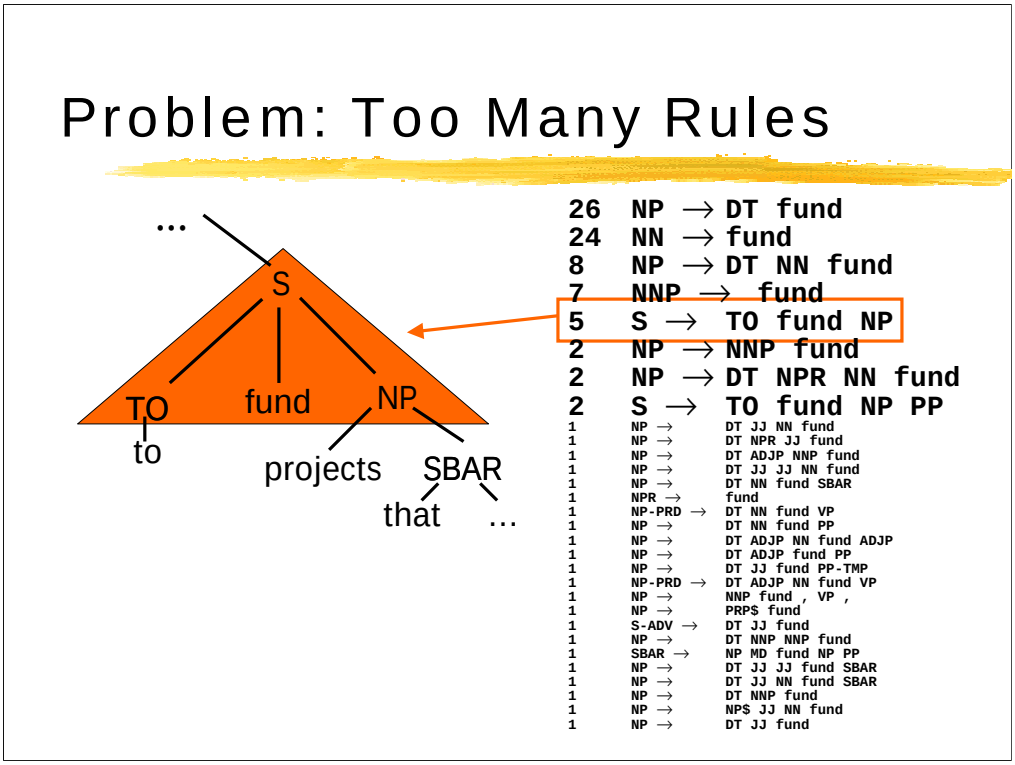
OLD: (More precisely, you have a prior expectation of that rule probability, which can be overridden by data, but which you can fall back on in the absence of data.)

What do I mean by "related rules"? I mean something like active and passive, but it varies from language to language. So you give the model a grab bag of *possible* relationships, which is language independent, and it learns which ones are predictive.

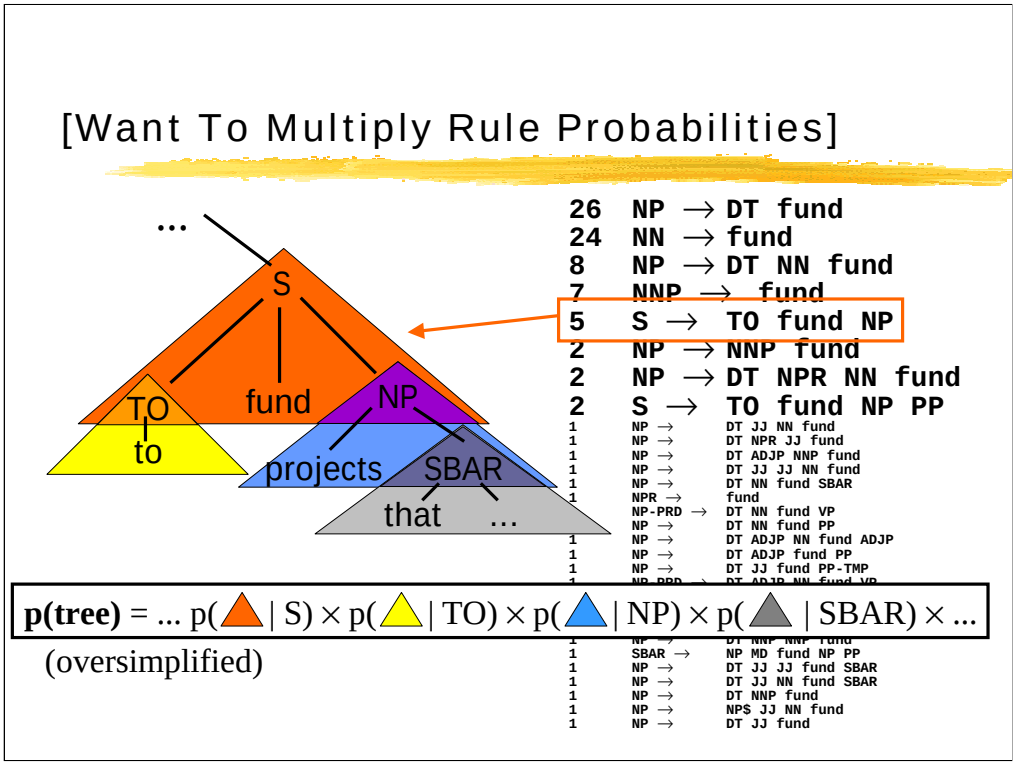
That's akin to feature selection, in TBL or maxent modeling. You have maybe 70000 features generated by filling in templates, but only a few hundred or a few thousand of them turn out to be useful.

The statistical method I'll use is a new one, but it has connections to other things.

First of all, I'm giving a very general talk first thing Monday morning about PFSMs, and these models are a special case.



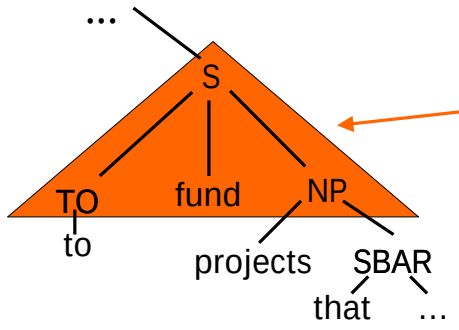
Here's a parse, or a fragment of one; the whole sentence might be "I want to fund projects that are worthy." To see whether it's a likely parse, we see whether its individual CF rules are likely. For instance, the rule we need here for "fund" was used 5 times in training data.



The other rules in the parse have their own counts. And to get the probability of the parse, basically you convert the counts to probabilities and multiply them.

I'm oversimplifying, but you already know how PCFGs work and it doesn't matter to this talk. What matters is how to convert the counts to probabilities.

Too Many Rules … But Luckily …



All these rules for fund – & other, still unobserved rules – are **connected** by the deep structure of English.

```
26   NP → DT fund
24   NN → fund
8    NP → DT NN fund
7    NNP → fund
5    S → TO fund NP
2    NP → NNP fund
2    NP → DT NPR NN fund
2    S → TO fund NP PP
1    NP →        DT JJ NN fund
1    NP →        DT NPR JJ fund
1    NP →        DT ADJP NNP fund
1    NP →        DT JJ JJ NN fund
1    NP →        DT NN fund SBAR
1    NPR →       fund
1    NP-PRD →    DT NN fund VP
1    NP →        DT NN fund PP
1    NP →        DT ADJP NN fund ADJP
1    NP →        DT ADJP fund PP
1    NP →        DT JJ fund PP-TMP
1    NP-PRD →    DT ADJP NN fund VP
1    NP →        NNP fund , VP ,
1    NP →        PRP$ fund
1    S-ADV →     DT JJ fund
1    NP →        DT NNP NNP fund
1    SBAR →      NP MD fund NP PP
1    NP →        DT JJ JJ fund SBAR
1    NP →        DT JJ NN fund SBAR
1    NP →        DT NNP fund
1    NP →        NP$ JJ NN fund
1    NP →        DT JJ fund
```

Notice that I'm using lexicalized rules. Every rule I pull out of training data contains a word, so words can be idiosyncratic: the list of rules for "fund" might be different than the list of rules for another noun, or at least have different counts.

That's important for parsing.

Now, I didn't pick "fund" for any reason– in fact, this is an old slide. But it's instructive to look at this list of rules for fund, which is from the Penn Treebank.

It's a long list, is the first thing to notice, and we haven't seen them all – there's a long tail of singletons.

But there's order here. All of these rules are connected in ways that are common in English.

Rules Are Related

- fund behaves like a typical singular noun ...

```
26   NP  → DT fund
24   NN  → fund
8    NP  → DT NN fund
7    NNP →  fund
5    S   → TO fund NP
```

one fact!
though PCFG represents it as many apparently unrelated rules.

```
1    NP →       DT NPR JJ fund
1    NP →       DT ADJP NNP fund
1    NP →       DT JJ JJ NN fund
1    NP →       DT NN fund SBAR
1    NPR →      fund
1    NP-PRD →   DT NN fund VP
1    NP →       DT NN fund PP
1    NP →       DT ADJP NN fund ADJP
1    NP →       DT ADJP fund PP
1    NP →       DT JJ fund PP-TMP
1    NP-PRD →   DT ADJP NN fund VP
1    NP →       NNP fund , VP ,
1    NP →       PRP$ fund
1    S-ADV →    DT JJ fund
1    NP →       DT NNP NNP fund
1    SBAR →     NP MD fund NP PP
1    NP →       DT JJ JJ fund SBAR
1    NP →       DT JJ NN fund SBAR
1    NP →       DT NNP fund
1    NP →       NP$ JJ NN fund
1    NP →       DT JJ fund
```

We could summarize them by saying that **fund** behaves like a typical singular noun.

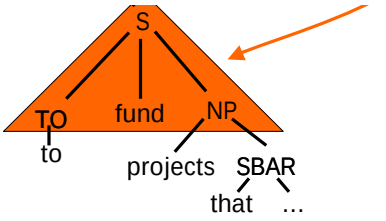
That's just one fact to learn – we don't have to learn the rules individually. So in a sense there's only one parameter here.

.

Rules Are Related

- fund behaves like a typical singular noun ...
- ... **or** transitive verb ...

one more fact!
even if several more rules.
Verb rules are **RELATED**.
Should be able to **PREDICT** the ones we haven't seen.



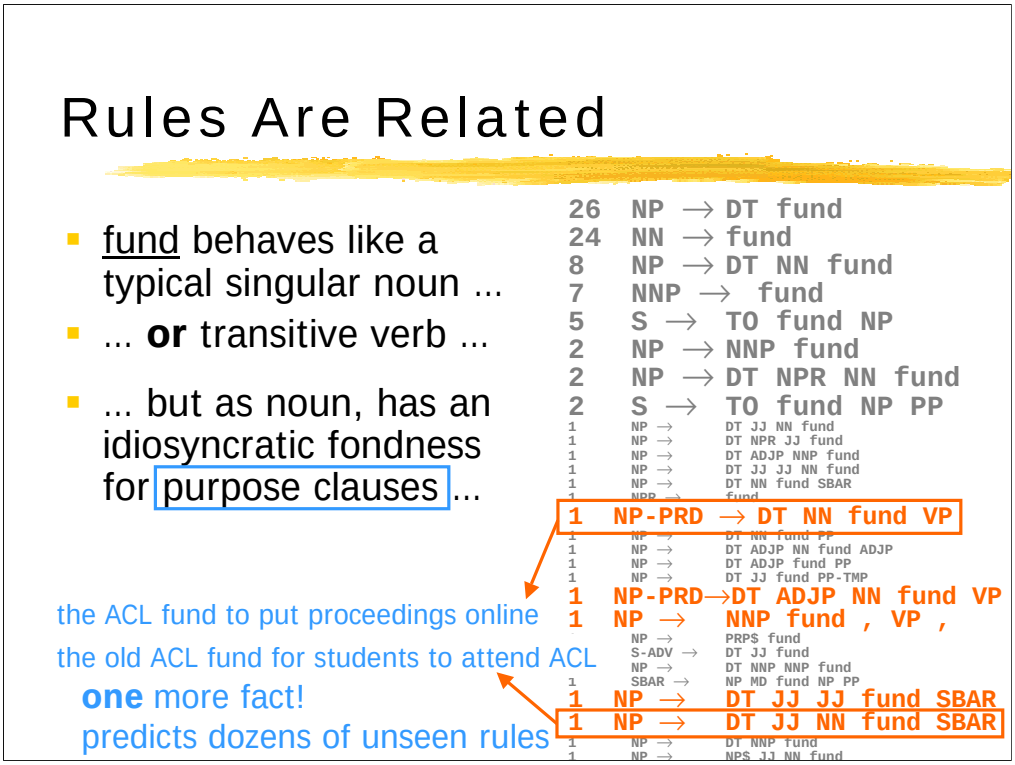

```
26   NP → DT fund
24   NN → fund
8    NP → DT NN fund
7    NNP → fund
5    S → TO fund NP
2    NP → NNP fund
2    NP → DT NPR NN fund
2    S → TO fund NP PP
1    NP → DT JJ NN fund
1    NP → DT NPR JJ fund
1    NP → DT ADJP NNP fund
1    NP → DT JJ JJ NN fund
1    NP → DT NN fund SBAR
1    NP → DT ADJP fund PP
1    NP → DT JJ fund PP-TMP
1    NP-PRD → DT ADJP NN fund VP
1    NP → NNP fund , VP ,
1    NP → PRP$ fund
1    S-ADV → DT JJ fund
1    NP → DT NNP NNP fund
1    SBAR → NP MD fund NP PP
1    NP → DT JJ JJ fund SBAR
1    NP → DT JJ NN fund SBAR
1    NP → DT NNP fund
1    NP → NP$ JJ NN fund
1    NP → DT JJ fund
```

Of course, it's not quite right, because we just saw it used as a transitive verb, *to fund projects that are worthy.*

There are a *few* verb rules in the list.

But that's just a second fact.

These verb rules are related. We've only seen a few, but that should be enough to predict the rest of the transitive verb paradigm.



I said it could act as a typical noun or verb, but that's still not quite right, because as a noun it's not quite typical.

Look at these rules in orange – for noun phrases like … They describe what the fund does.

Typical nouns *can* take these purpose clauses, but **fund** takes them more often than typical, probably for semantic reasons.

Well, that's fact #3 about **fund**. It explains these 5 rules and predicts dozens more.

Rules Are Related

- fund behaves like a typical singular noun ...
- ... **or** transitive verb ...
- ... but as noun, has an idiosyncratic fondness for purpose clauses ...
- ... and maybe other idiosyncrasies to be discovered, like unaccusativity ...

```
26  NP → DT fund
24  NN → fund
8   NP → DT NN fund
7   NNP →  fund
5   S →  TO fund NP
2   NP → NNP fund
2   NP → DT NPR NN fund
```

NSF issued the grant
The grant issued today

NSF funded the grant
The grant funded today

unlikely sentence, but if we do see it,
is unaccusativity **plausible**? (vs. other parse)

And I'll mention one more potential fact.

There's no rule in training data suggesting that **fund** might be unaccusative.

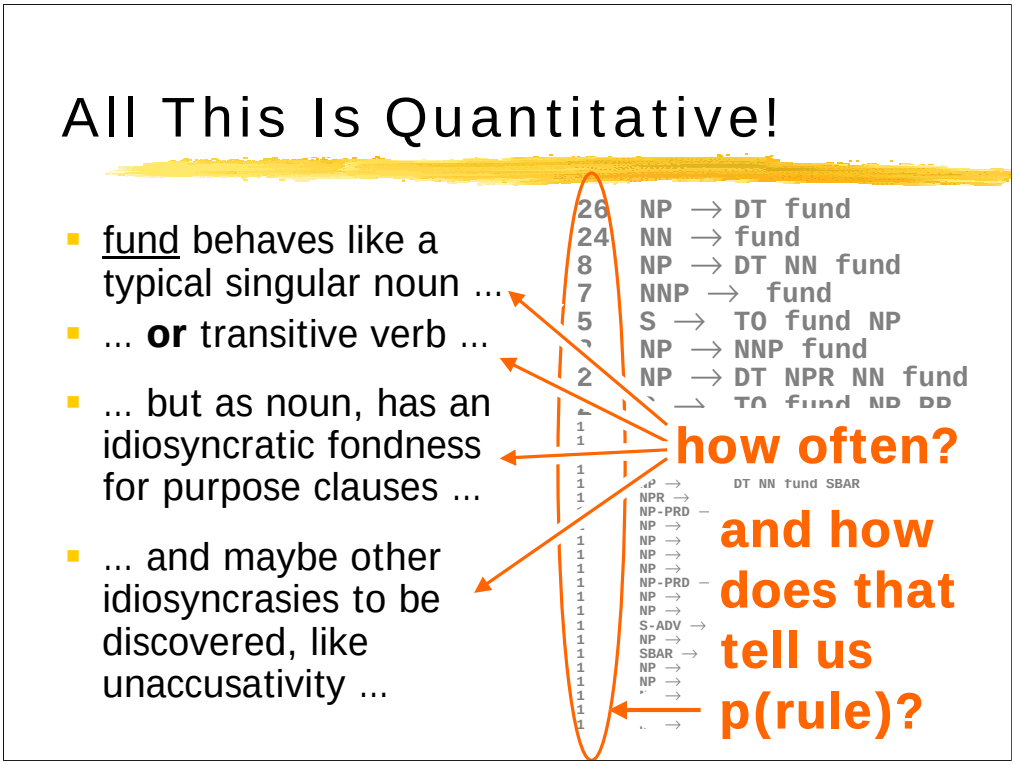
What's unaccusative? It's kind of a sneak passive, like this …

We'd like to say that since some verbs like **issue** can do this, maybe **fund** can too: NSF funded the grant, the grant funded today.

Based on the evidence we've seen so far, that's low-probability. But we don't want it to be too low, since if the system were to see this sentence,

it would have to decide between the unaccusative parse and treating today as a direct object: today was funded by the grant.

We'd want it to admit that the unaccusative parse is syntactically reasonable. That's how it can learn new constructions – it does EM. "Oh, that's the best parse of this weird sentence, I guess I'll count it as new training data."

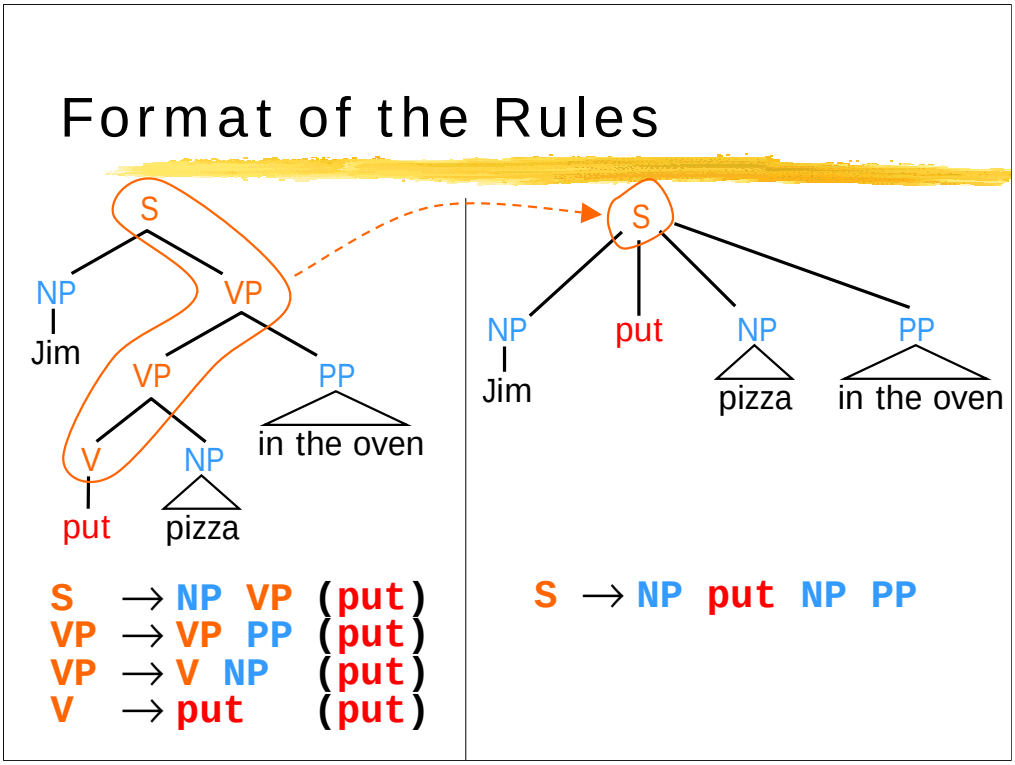


So we have 4 "facts" about **fund** – there might be more. And really, they're all quantitative.

How **OFTEN** is it a noun, and how often a verb?

How **MUCH** more frequent are purpose clauses than for the typical noun?

How **OFTEN** is **fund** used unaccusatively?



Here's a traditional structure for "Jim put pizza in the oven."

Going from the top down, it expands S by a sequence of 4 rules.

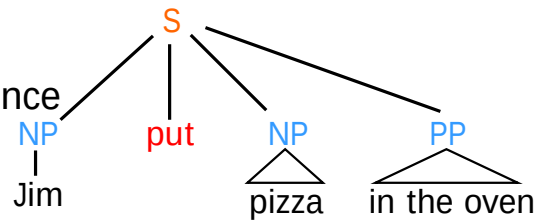
Nowadays we condition those expansions on the head word, put – note that put is the head of all these projections.

But I'm going to argue in favor of this structure on the right, which collapses the spine of put into a single level. Put takes all its dependents at once.

Format of the Rules

Why use flat rules?

- Avoids silly independence assumptions: a win
  - Johnson 1998 →
  - New experiments
- Our method likes them
  - Traditional rules aren't systematically related
  - But relationships exist among wide, flat rules that express different ways of filling same roles



S → **NP** **put** **NP** **PP**

It's a way of avoiding independence assumptions. Adjuncts are not really independent of one another.

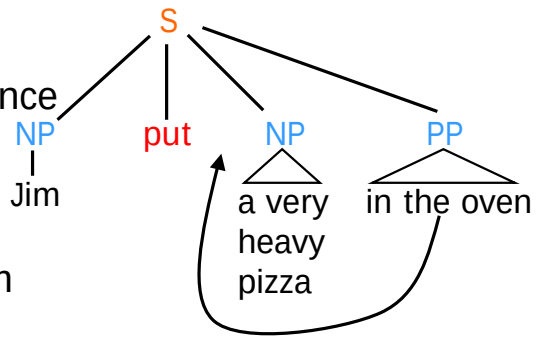
And even traditional methods do better when estimating the whole flat rule at one go. Mark Johnson showed that for some special cases, and my experiments bear it out in spades.

But the new method especially wants to work with wide, flat rules, because it looks at relationships among rules.

Format of the Rules

Why use flat rules?

- Avoids silly independence assumptions: a win
  - Johnson 1998 →
  - New experiments
- Our method likes them
  - Traditional rules aren't systematically related
  - But relationships exist among wide, flat rules that express different ways of filling same roles

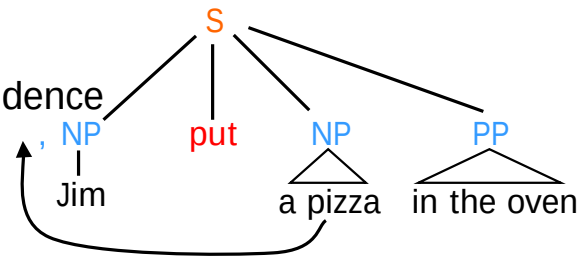


S → **NP** **put** **PP** **NP**

Format of the Rules

Why use flat rules?

- Avoids silly independence assumptions: a win
  - Johnson 1998 →
  - New experiments
- Our method likes them
  - Traditional rules aren't systematically related
  - But relationships exist among wide, flat rules that express different ways of filling same roles



S → NP , NP put PP

A *pepperoni pizza* he put in the oven. What's next, shrimp fricassee?

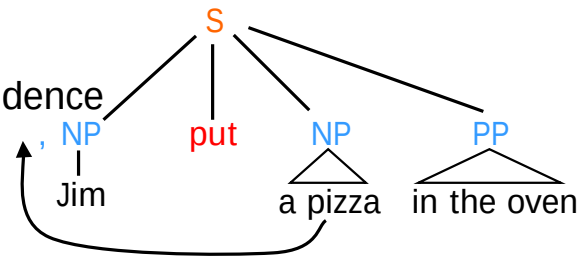
Jim put a pizza in the oven last week

A pizza was put in the oven last week

Format of the Rules

Why use flat rules?

- Avoids silly independence assumptions: a win
  - Johnson 1998 →
  - New experiments
- Our method likes them
  - Traditional rules aren't systematically related
  - But relationships exist among wide, flat rules that express different ways of filling same roles



in short, flat rules are the **locus of transformations**

What a transformation does is to take a flat rule – a word together with all its roles – and rearrange the way those roles are expressed syntactically.

Format of the Rules

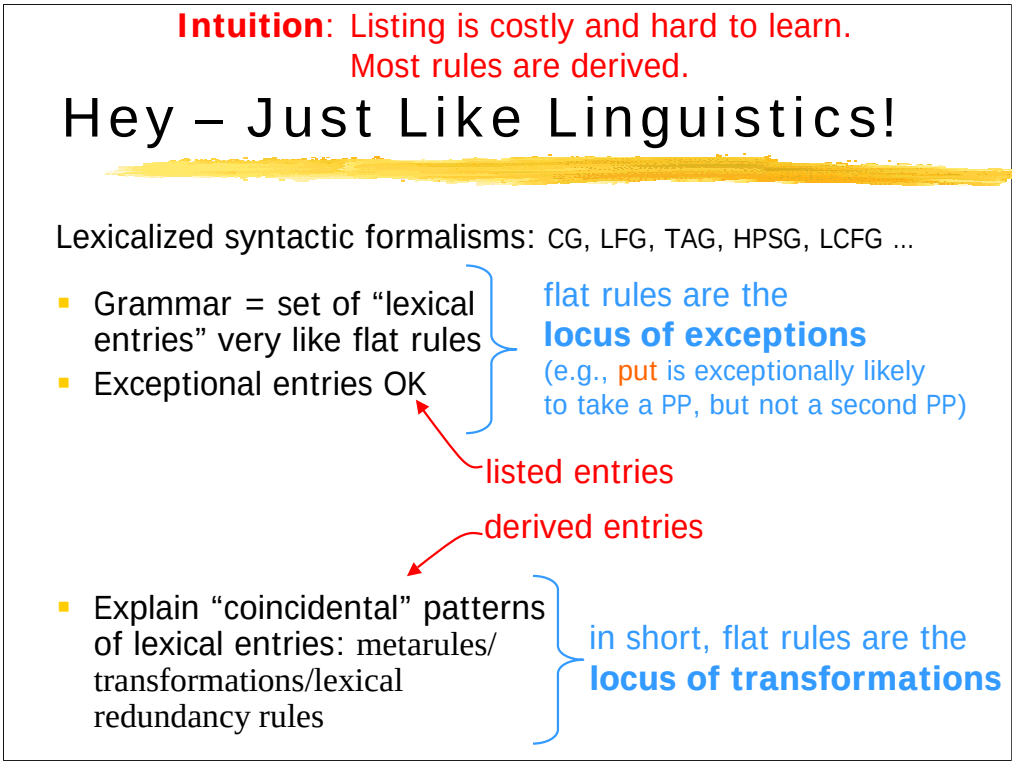
Why use flat rules?

- Avoids silly indep. assumptions: a win
  - Johnson 1998 →
  - New experiments
- Our method likes them
  - Traditional rules aren't systematically related
  - But relationships exist among wide, flat rules that express different ways of filling same roles

flat rules are the **locus of exceptions**
(e.g., put is exceptionally likely to take a PP, but not a second PP)

in short, flat rules are the **locus of transformations**

And some ways of expressing those roles may be particularly favored.



Hey, just like ling. Think about the lexicon …

What these formalisms have in common is that they all end in "G" – no, just kidding.

In all of them, the grammar is just a set of lexical entries that can be combined in various ways. And a lexical entry is always basically like one of our flat rule.

It's ok to have some weird entries in the lexicon, but there's also a lot of redundancy, as we saw for **FUND**. And linguists have mechanisms for **deriving** the redundant entries.

So you could also see this talk as being about "how to stochasticize these approaches to syntax – including the lexical redundancy rules."

The Rule Smoothing Task

- **Input:** Rule counts (from parses or putative parses)
- **Output:** Probability distribution over rules
- **Evaluation:** Perplexity of held-out rule counts
  - That is, did we assign high probability to the rules needed to correctly parse test data?

Now that we know what a rule is, let's talk about rule smoothing.

We look at some parses and count up the rules. In EM, they'd be fractional counts. Then we have to figure out the real probabilities of the rules.

To evaluate, we look at some more parses and see whether they perplex our model. Our model is good if it assigns high probability to the rules needed to parse test sentences correctly.

The Rule Smoothing Task

- **Input:** Rule counts (from parses or putative parses)
- **Output:** Probability distribution over rules
- **Evaluation:** Perplexity of held-out rule counts

Rule probabilities: p(S → NP put NP PP | S, put)

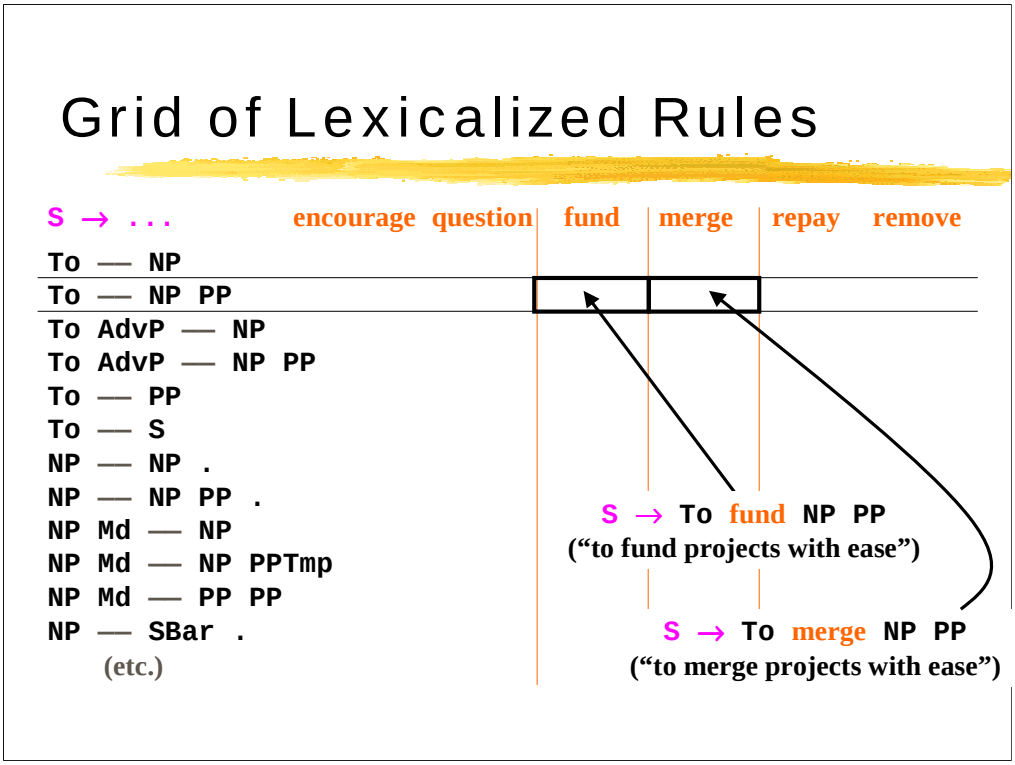
Infinite set of possible rules; so we will estimate

p(S → NP Adv PP put PP PP NP AdjP S | S, put)
= a very tiny number > 0

Now that we know what a rule is, let's talk about rule smoothing.

We look at some parses and count up the rules. In EM, they'd be fractional counts. Then we have to figure out the real probabilities of the rules.

To evaluate, we look at some more parses and see whether they perplex our model. Our model is good if it assigns high probability to the rules needed to parse test sentences correctly.



We saw this rule before ...

I've pulled out the **head word**, **fund**, and used it as a column label. The rest of the rule w/o the word is the row label, and I'll call it a **frame**.

Here's a similar atom - only the word is different, the frame is the same - so it goes in a different column of the same row.

Training Counts

S → ...	encourage	question	fund	merge	repay	remove
To — NP	1	1	5	1	3	2
To — NP PP	1	1	2	2	1	1
To AdvP — NP						1
To AdvP — NP PP						1
NP — NP .		2				
NP — NP PP .	1					
NP Md — NP	1					
NP Md — NP PPTmp					1	
NP Md — PP PP						1
To — PP				1		
To — S	1					
NP — SBar .		2				
(other)						

Count of (word, frame)

And in training data, we saw each of those frames twice. These are real counts, by the way.

So this is our training data ...

Naive prob. estimates (MLE model)

S → . . .	encourage	question	fund	merge	repay	remove
To — NP	200	167	714	250	600	333
To — NP PP	200	167	286	500	200	167
To AdvP — NP	0	0	0	0	0	167
To AdvP — NP PP	0	0	0	0	0	167
NP — NP .	0	333	0	0	0	0
NP — NP PP .	200	0	0	0	0	0
NP Md — NP	200	0	0	0	0	0
NP Md — NP PPTmp	0	0	0	0	200	0
NP Md — PP PP	0	0	0	0	0	167
To — PP	0	0	0	250	0	0
To — S	200	0	0	0	0	0
NP — SBar .	0	333	0	0	0	0
(other)	0	0	0	0	0	0

Estimate of p(frame | word) * 1000

First column, “encourage” - we have seen “encourage” once with each of these frames, for a total of 5 - so we give each frame the probability “1 out of 5” or 0.2. And every other

But there are more things in language and speech, MLE model, than are dreamt of in your philosophy! In other words, all these zeroes are a problem. There *are* new things under the sun, and they *will* show up in test data. In fact, they’ll show up quite often!

When the parser needs a particular atom, chances are 21% it never saw that atom during training - so it’s got prob=0 in this table. You might point out, well, that’s because your atoms are so specific - they’re specified down to the level of the particular word. But chances are 5% it never even saw the frame before - so the whole row is all 0’s. We have to generalize from the other rows.

TASK: counts → probs ("smoothing")

S → ...	encourage	question	fund	merge	repay	remove
To — NP	142	117	397	210	329	222
To — NP PP	77	64	120	181	88	80
To AdvP — NP	0.55	0.47	1.1	0.82	0.91	79
To AdvP — NP PP	0.18	0.15	0.33	0.37	0.26	50
NP — NP .	22	161	7.8	7.5	7.9	7.5
NP — NP PP .	79	8.5	2.6	2.7	2.6	2.6
NP Md — NP	90	2.1	2.4	2.0	24	2.6
NP Md — NP PPTmp	1.8	0.16	0.17	0.16	69	0.19
NP Md — PP PP	0.1	0.027	0.027	0.038	0.078	59
To — PP	9.2	6.5	12	126	10	9.1
To — S	98	1.6	4.3	3.9	3.6	2.7
NP — SBar .	3.4	190	3.2	3.2	3.2	3.2
(other)	478	449	449	461	461	482

Estimate of p(frame | word) * 1000

... but for parsing what we really need is the true probabilities of the atoms, not the counts.

(These are probs * 1000, to 2 signif. figures, so the ones that jut left are big, the ones that jut right are small.)

[flip back and forth to counts] So that's our task - to turn counts into probabilities. This is traditionally called smoothing: we've sort of smeared the black counts down the column so that we get some positive probability on each row.

In fact, as the legend at the bottom says, these are p(frame | word), so each column is a distribution over possible frames for a word. It ranges over ALL possible frames, and it sums to 1. A possible frame is anything of the form "blah blah blah --- blah blah," so there are infinitely many of them.

You'll note that the counts involved are very small. These are real data, and they're 0's and 1's and 2's. The difference between seeing something 0 times and 1 time may just be a matter of luck - or it may be significant. That' swhy this problem is hard, and why I'm using a Bayesian approach, which weighs evidence carefully against expectations.

Smooth Matrix via LSA / SVD, or SBS?

S → . . .	encourage	question	fund	merge	repay	remove
To — NP	1	1	5	1	3	2
To — NP PP	1	1	2	2	1	1
To AdvP — NP						1
To AdvP — NP PP						1
NP — NP .		2				
NP — NP PP .	1					
NP Md — NP	1					
NP Md — NP PPTmp					1	
NP Md — PP PP						1
To — PP				1		
To — S	1					
NP — SBar .		2				
(other)						

Count of (word, frame)

No – then each column would be approximated by a linear combination of standard columns.

Also true for similarity-based smoothing (Lee et al.)

That's not a bad idea, since the column for **fund** would be a mixture of noun behavior and verb behavior.

But lots of rows of the training matrix are all zeros. That is, lots of frames in test data never showed up in training data at all.

SVD can't handle that.

And SVD doesn't know anything about the internal structure of frames – it sees each column as a vector over interchangeable dimensions.

But it's clear from looking at this table that the internal structure is really predictive.

If a frame appears, then it generally appears with PP added at the right edge.

These frames for **remove** are just split-infinitive versions of these: to completely remove, to surgically remove.

Smoothing via a Bayesian Prior

- Choose grammar to maximize p(observed rule counts | grammar)*p(grammar)
- grammar = probability distribution over rules
- **Our job:** Define p(grammar)
- **Question:** What makes a grammar likely, a priori?
- **This paper's answer:** Systematicity. Rules are mainly derivable from other rules. Relatively few stipulations ("deep facts").

We'd like a grammar that explains the observed data, and is also a priori a good grammar.

So we use Bayes' Rule and maximize this product.

By a grammar I mean a probability distribution over rules.

Only a Few Deep Facts

- fund behaves like a transitive verb 10% of time ...
- and noun 90% of time ...
- ... takes purpose clauses 5 times as often as typical noun.

```
26  NP → DT fund
24  NN → fund
8   NP → DT NN fund
7   NNP → fund
5   S → TO fund NP
2   NP → NNP fund
2   NP → DT NPR NN fund
2   S → TO fund NP PP
1   NP → DT JJ NN fund
1   NP → DT NPR JJ fund
1   NP → DT ADJP NNP fund
1   NP → DT JJ JJ NN fund
1   NP → DT NN fund SBAR
1   NPR → fund
1   NP-PRD → DT NN fund VP
1   NP → DT NN fund PP
1   NP → DT ADJP NN fund ADJP
1   NP → DT ADJP fund PP
1   NP → DT JJ fund PP-TMP
1   NP-PRD → DT ADJP NN fund VP
1   NP → NNP fund , VP ,
1   NP → PRP$ fund
1   S-ADV → DT JJ fund
1   NP → DT NNP NNP fund
1   SBAR → NP MD fund NP PP
1   NP → DT JJ JJ fund SBAR
1   NP → DT JJ NN fund SBAR
1   NP → DT NNP fund
1   NP → NP$ JJ NN fund
1   NP → DT JJ fund
```

These are the key facts.

If fund has other little idiosyncrasies, we can add those facts, but small idiosyncrasies don't hurt the prior probability much – the prior cares more about big ones.

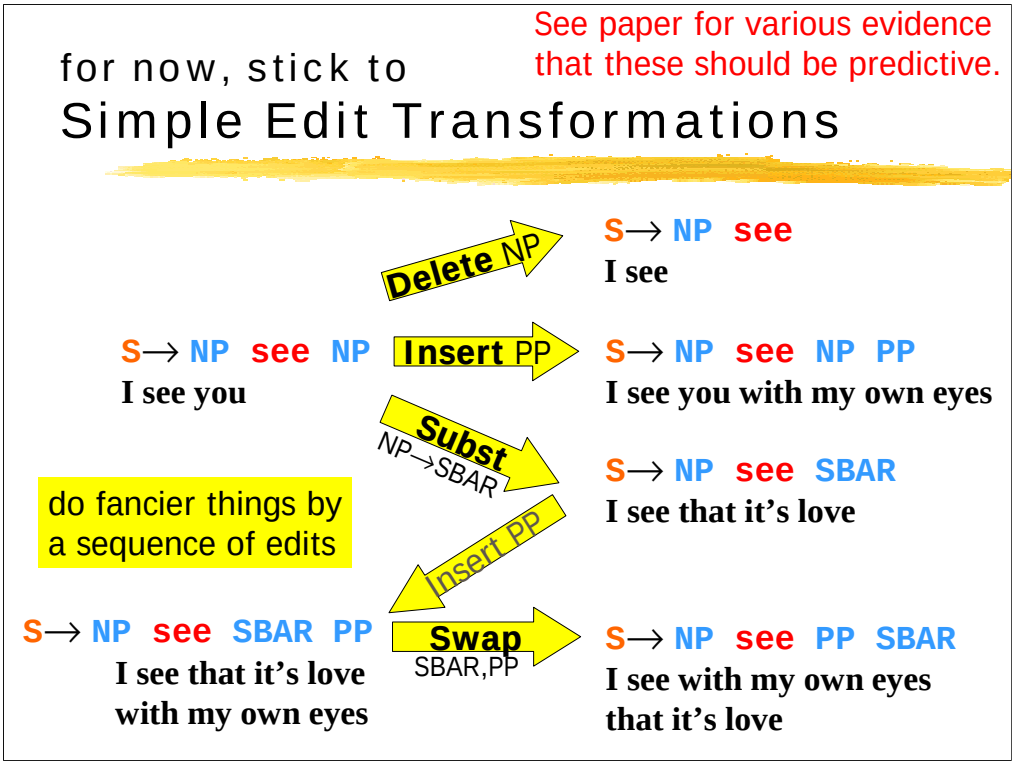
.

Smoothing via a Bayesian Prior

- Previous work (several papers in past decade):
  - Rules should be few, short, and approx. equiprobable
  - These priors try to keep rules **out of** grammar
  - Bad idea for lexicalized grammars ...

- This work:
  - Prior tries to get related rules **into** grammar
  - transitive ➔ passive at ≈1/20 the probability
    - NSF spraggles the project ➔ The project is spraggled by NSF
    - Would be weird for the passive to be missing, and prior knows it!
    - In fact, weird if p(passive) is too far from 1/20 * p(active)
  - Few facts, not few rules!

If you can say NSF funds the project,
it would be really weird if you couldn't say The project is funded by NSF.
This prior is going to be much happier if the passive rule for fund is in there.
Or really, all rules are "in there," the question is what the probability is.



We won't do passives in these experiments, though. Stick to simple edit transformations.

Start with a simple transitive verb rule.

What are the related rules?

We could suppress the direct object – let the hearer infer it.

We could make the instrument explicit, in case the hearer can't infer it. - I see you in the park with a telescope

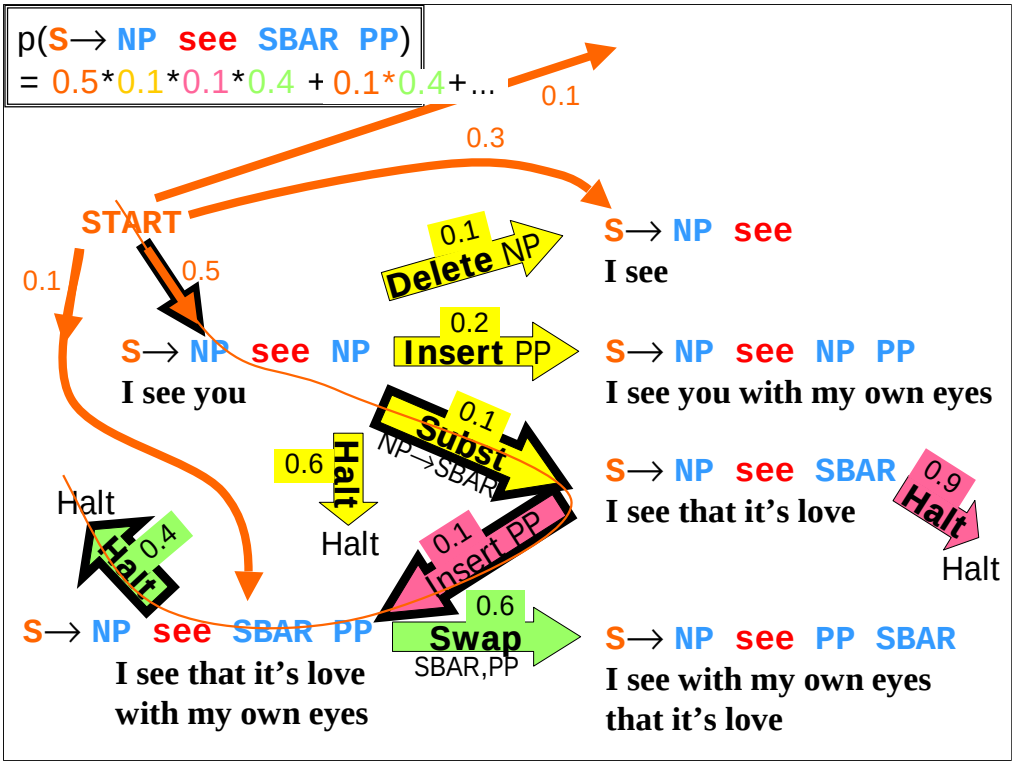
We could change the type of the direct object and see not an object, you, but a proposition, that it's love.

And we could do heavy-shift.

These have different probabilities. Remember, we're treating this as feature selection. We'll tell the statistical model about all kinds of transformations,

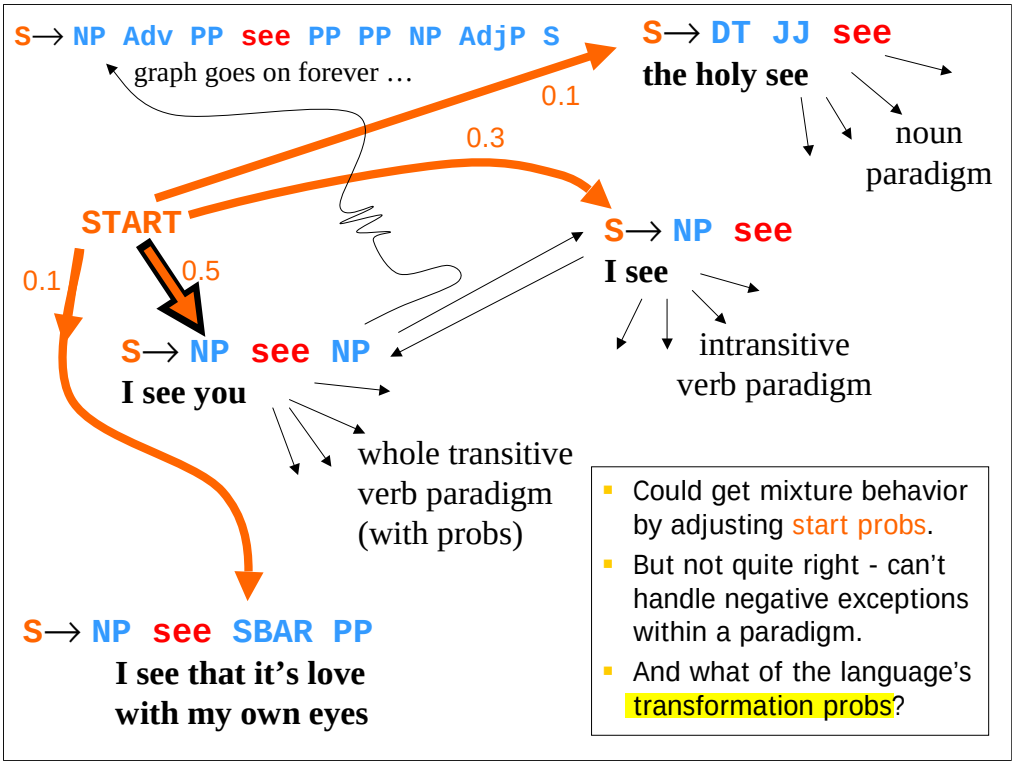
including insertion of weird constituents in weird places, and let it figure out which ones are good. But I did some preliminary

experiments to verify that **in general,** edit transformations do tend to be predictive. You can read about those in the paper.



These transformations have probabilities.

We can use the transformation probabilities to calculate the rule probabilities, basically in the obvious way.



So if we increase this from 0.5, then all the transitive verb rules increase.

If we increase this from 0.3, then all the intransitive verb rules increase.

And there's some crosstalk between them, so if we suddenly learn that **see** is a transitive verb, we also raise the probability that it **could** be used intransitively.

Just as fund can be used as a verb or noun, so can see: …

So one way to get a simple mixture behavior would be to adjust the start weights. That would be sort of like SVD, where a word is approximated as a linear combination of basis vectors. But here the "basis vectors" or paradigms are infinite – they're distributions over an infinite set of possible rules. And we're doing something else that SVD can't do, which is to use information about the dimensions – some of these rules are related to each other in the sense of having low edit distance.

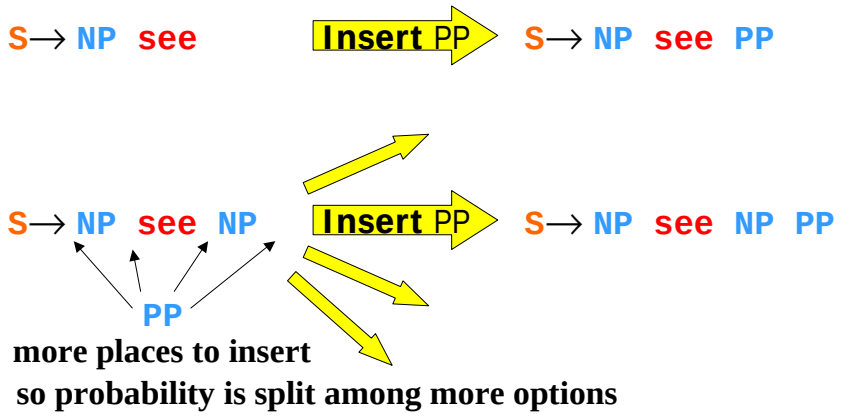
If every word's distribution over frames is a mixture of standard distributions (which is not quite what we'll end up doing),

then maybe we should use LSA or SVD to find those standard distributions and the mixture coefficients.

But that would just model the observed distribution vector as a sum of standard vectors. It wouldn't constrain what those standard vectors looked like,

for example by paying attention to edit distance. And those standard vectors would have finite support, so

Infinitely Many Arc Probabilities: Derive From Finite Parameter Set



Why not just give any two PP-insertion arcs the same probability?

In second one, we are inserting PP into a slightly different context.

In second one, more places to insert PP, so each has lower probability.

Arc Probabilities: A Conditional Log-Linear Model

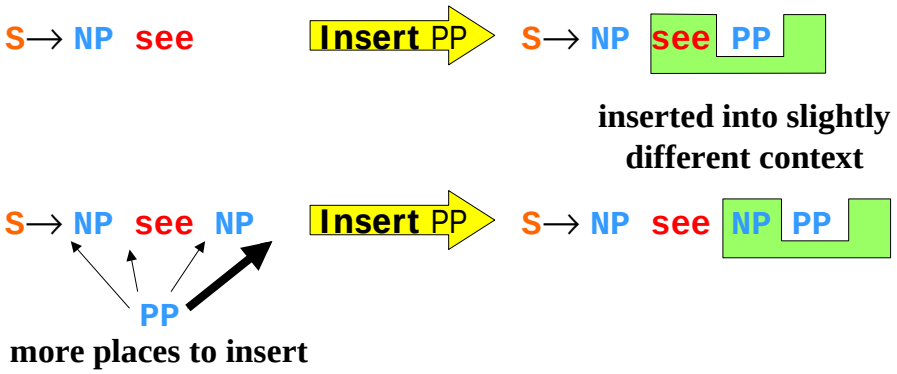
To make sure outgoing arcs sum to 1, introduce a normalizing factor Z (at each vertex).

S→ NP see NP — **Insert PP** → **S→ NP see NP PP**

$\frac{1}{Z} \exp \theta_3 + \theta_5 + \theta_6$

Models p(arc | vertex)

Arc Probabilities: A Conditional Log-Linear Model



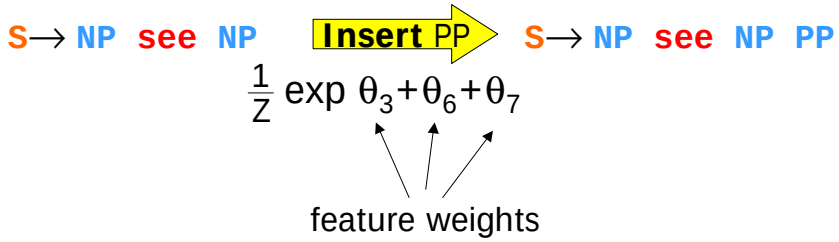
Both are PP-adjunction arcs. Same probability?

Almost but not quite ...

Arc Probabilities: A Conditional Log-Linear Model

Not enough just to say "Insert PP."

Each arc bears several features, whose weights determine its probability.



a feature of weight 0 has no effect

raising a feature's weight strengthens all arcs with that feature

Every arc bears **several** features describing what it does, which together determine its probability.

… It's like turning a knob that adjusts a whole class of arcs. There are as many knobs as there are features. But only finitely many.

Arc Probabilities: A Conditional Log-Linear Model

S → NP see **Insert** PP S → NP see PP

$$\frac{1}{Z'} \exp \theta_3 + \theta_5 + \theta_7$$

S → NP see NP **Insert** PP S → NP see NP PP

$$\frac{1}{Z} \exp \theta_3 + \theta_6 + \theta_7$$

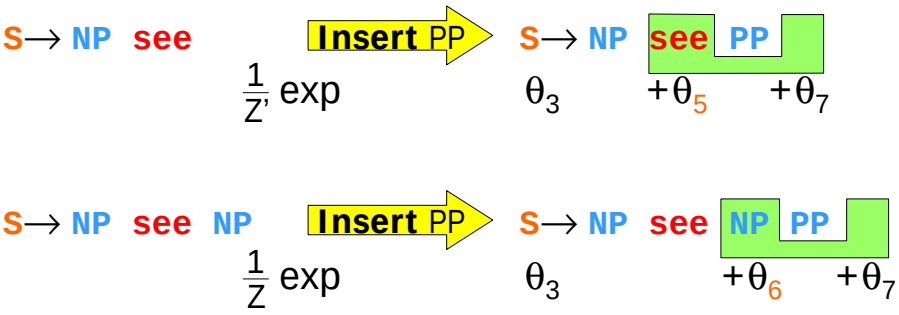
θ_3 : appears on arcs that insert **PP** into **S**

θ_5 : appears on arcs that insert **PP** just after head

θ_6 : appears on arcs that insert **PP** just after **NP**

θ_7 : appears on arcs that insert **PP** just before edge

Arc Probabilities: A Conditional Log-Linear Model



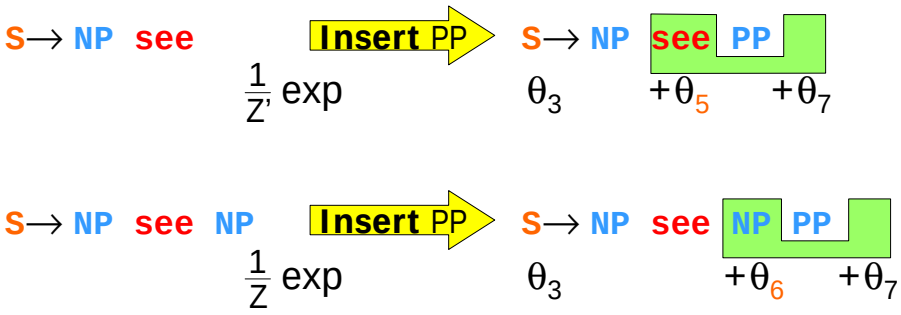
θ_3 : appears on arcs that insert **PP** into **S**

θ_5 : appears on arcs that insert **PP** just after head

θ_6 : appears on arcs that insert **PP** just after **NP**

θ_7 : appears on arcs that insert **PP** just before edge

Arc Probabilities: A Conditional Log-Linear Model

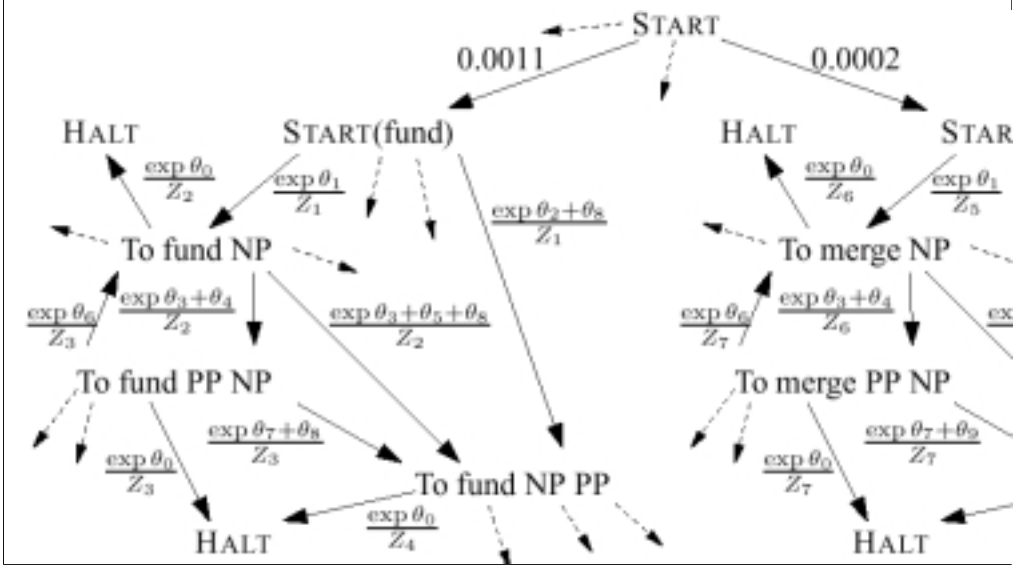


These arcs share most features.
So their probabilities tend to rise and fall together.
To fit data, could manipulate them independently (via θ_5,θ_6).

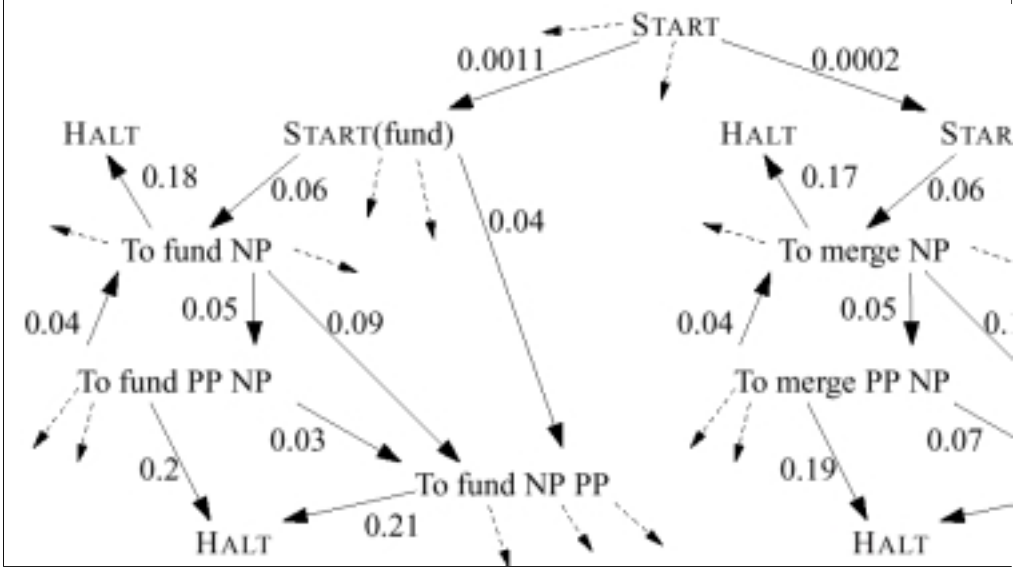
Prior Distribution

- PCFG grammar is **determined** by θ_0, θ_1, θ_2, …

Universal Grammar



Instantiated Grammar

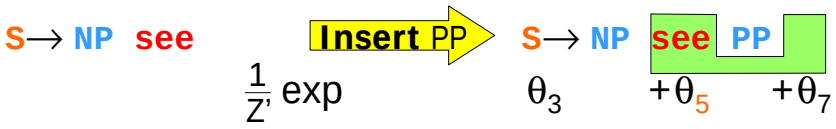


Prior Distribution

- Grammar is **determined** by $\theta_0, \theta_1, \theta_2, \ldots$
- Our prior: $\theta_i \sim N(0, \sigma^2)$, IID
- Thus: -log p(grammar) = c+ $(\theta_0^2+\theta_1^2+\theta_2^2+\ldots)/\sigma^2$

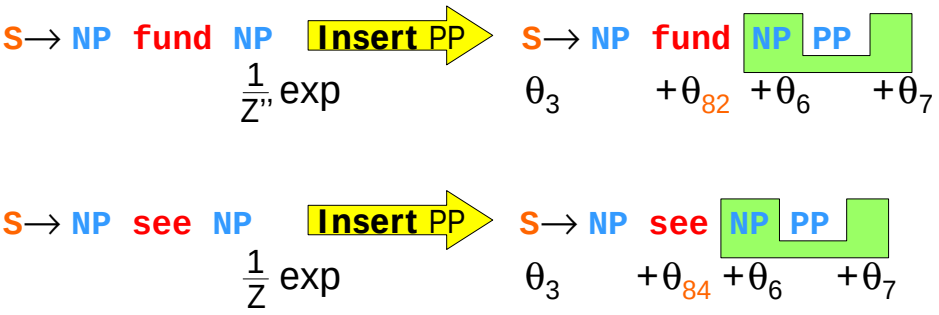
- So good grammars have few large weights.
- Prior prefers one generalization to many exceptions.

Arc Probabilities: A Conditional Log-Linear Model



To raise both rules' probs, cheaper to use θ_3 than both θ_5 & θ_6.
This generalizes – also raises other cases of PP-insertion!

Arc Probabilities: A Conditional Log-Linear Model



To raise both probs, cheaper to use θ_3 than both θ_{82} & θ_{84}.
This generalizes – also raises other cases of PP-insertion!

Reparameterization

- Grammar is determined by $\theta_0, \theta_1, \theta_2, \ldots$
- A priori, the θ_i are normally distributed

- We've reparameterized!
- The parameters are feature weights θ_i, not rule probabilities
- Important tendencies captured in big weights
  - Similarly: Fourier transform – find the formants
  - Similarly: SVD – find the principal components
  - It's on this deep level that we want to compare events, impose priors, etc.

```
( (S
    (NP-SBJ (JJ Big) (NN indexer) (NNP Bankers) (NNP Trust) (NNP Co.) )
    (ADVP (RB also) )
    (VP (VBZ uses)
      (NP (NNS futures) )
      (PP-LOC (IN in)
        (NP
          (NP (DT a) (NN strategy) )
          (SBAR
            (WHNP-1 (IN that) )
            (S
              (PP (IN on)
                (NP (NN average) ))
              (NP-SBJ (-NONE- *T*-1) )
              (VP (VBZ has)
                (VP (VBN added)
                  (NP (CD one) (NN percentage) (NN point) )
                  (PP-CLR (TO to)
                    (NP
                      (NP (PRP$ its) (JJ enhanced) (NN fund) (POS 's) )
                      (NNS returns) ))))))))
    (. .) ))
```

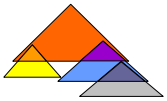
```
( (S
    (NP (JJ big) indexer (NPR (NNP Bankers) (NNP Trust) Co.) )
    (ADVP also)
    uses
    (NP futures)
    (PP-LOC in
      (NP
        (NP (DT a)
            strategy
            (SBAR that
              (S
                (PP on (NP average) )
                (VBZ has)
                added
                (NP (CD one) (NN percentage) point)
                (PP to
                  (NP
                    (NP$ (NP (PRP$ its) (JJ enhanced) fund) 's)
                    returns) ))))))
    (. .) ))
```

	frame		
headword	LHS		RHS
big	JJ	→	——
indexer	NP	→	JJ —— NPR
Bankers	NNP	→	——
Trust	NNP	→	——
Co.	NPR	→	NNP NNP ——
also	ADVP	→	——
uses	S	→	NP ADVP —— NP PP-LOC .
futures	NP	→	——
in	PP-LOC	→	—— NP
a	DT	→	——
strategy	NP	→	DT —— SBAR
that	SBAR	→	—— S
on	PP	→	—— NP
average	NP	→	——
has	VBZ	→	——
added	S	→	PP VBZ —— NP PP
one	QP	→	——
percentage	NN	→	——
point	NP	→	QP NN ——
to	PP	→	—— NP
its	PRP$	→	——
enhanced		→	JJ ——
fund	NP	→	PRP$ JJ ——
's	NP$	→	NP ——
returns	NP	→	NP$ ——
.	.	→	——

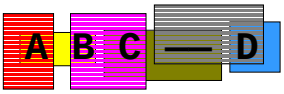
Other models of this string:
- max-likelihood
- n-gram
- Collins arg/adj
- hybrids

Simple Bigram Model (Eisner 1996)

- A parser assumes tree is probable if its component rules are:



- Try assuming rule is probable if its component bigrams are:



p(A | start) × p(B | A)
× p(C | B) × p(—— | C)
× p(D | ——) × p(stop | D)

- Markov process, 1 symbol of memory; conditioned on L, w, side of ——
- One-count backoff to handle sparse data (Chen & Goodman 1996)

p(L → A B C — D | w) = p(L | w)• p(A B C — D | L,w)

Ok, here's a simple model that does assign non-zero probability to every atom. Remember we've assumed a tree is probable if its component atoms are. We might make the same independence assumption at a finer grain, and assume that an atom is probable if its subatomic particles are.

I've taken a subatomic particle to be a sequence of 2 consecutive nonterminals in the frame, known as a bigram. The bigrams here are start A, A B, etc. To get the prob of the frame, we do like before - multiply together the bigrams' probabilities and divide by the overlap. And that's just saying that the frame is generated by a Markov process with one symbol of memory. With some other tricks of the trade, this does respectably at parsing if you have $250K worth of data.

headword	frame LHS		RHS
big	JJ	→	——
indexer	NP	→	JJ [$_{NN}$ ——] NPR
Bankers	NNP	→	——
Trust	NNP	→	——
Co.	NPR	→	NNP NNP [$_{NNP}$ ——]
also	ADVP	→	[$_{RB}$ ——]
uses	S	→	NP ADVP [$_{VP}$ [$_{VBZ}$ ——] NP PP-LOC] .
futures	NP	→	[$_{NNS}$ ——]
in	PP-LOC	→	[$_{IN}$ ——] NP
a	DT	→	——
strategy	NP	→	[$_{NP}$ DT [$_{NN}$ ——]] SBAR
that	SBAR	→	[$_{WHNP}$ [$_{IN}$ ——]] S
on	PP	→	[$_{IN}$ ——] NP
average	NP	→	[$_{NN}$ ——]
has	VBZ	→	——
added	S	→	PP [$_{VP}$ VBZ [$_{VP}$ [$_{VBN}$ ——] NP PP]]
one	QP	→	[$_{CD}$ ——]
percentage	NN	→	——
point	NP	→	QP NN [$_{NN}$ ——]
to	PP	→	[$_{TO}$ ——] NP
its	PRP$	→	——
enhanced		→	JJ ——
fund	NP	→	PRP$ JJ [$_{NN}$ ——]
's	NP$	→	NP [$_{POS}$ ——]
returns	NP	→	NP$ [$_{NNS}$ ——]
.	.	→	——

Use “non-flat” frames?
Extra training info.
For test, sum over all bracketings.

Figure 6.7: A version of Fig. 6.3 if frame-internal brackets are retained at step 8 of data preparation (§6.2).

Perplexity: Predicting test frames

	basic	
	flat	non-flat[b]
Treebank	∞	∞
1-gram	1774.9	86435.1
2-gram	**135.2**	199.3
3-gram	136.5	177.4
Collins[c]	363.0	494.5
transformation	**108.6**	

20% further reduction

from previous lit.

Can get big perplexity reduction just by flattening.

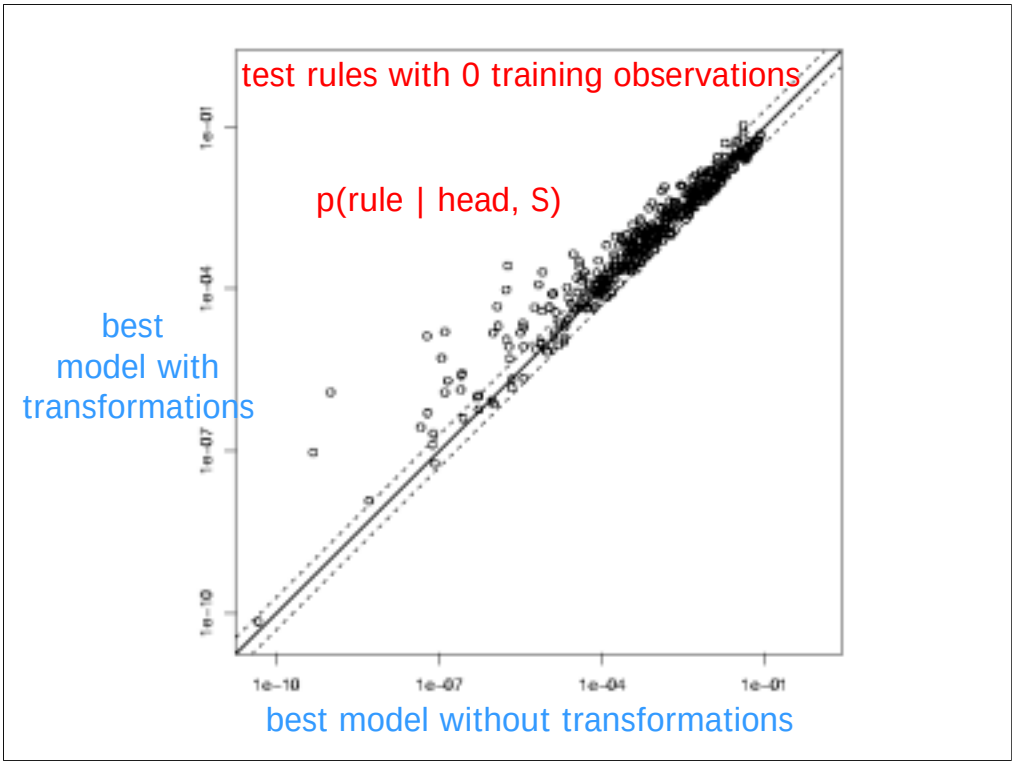
Perplexity: Predicting test frames

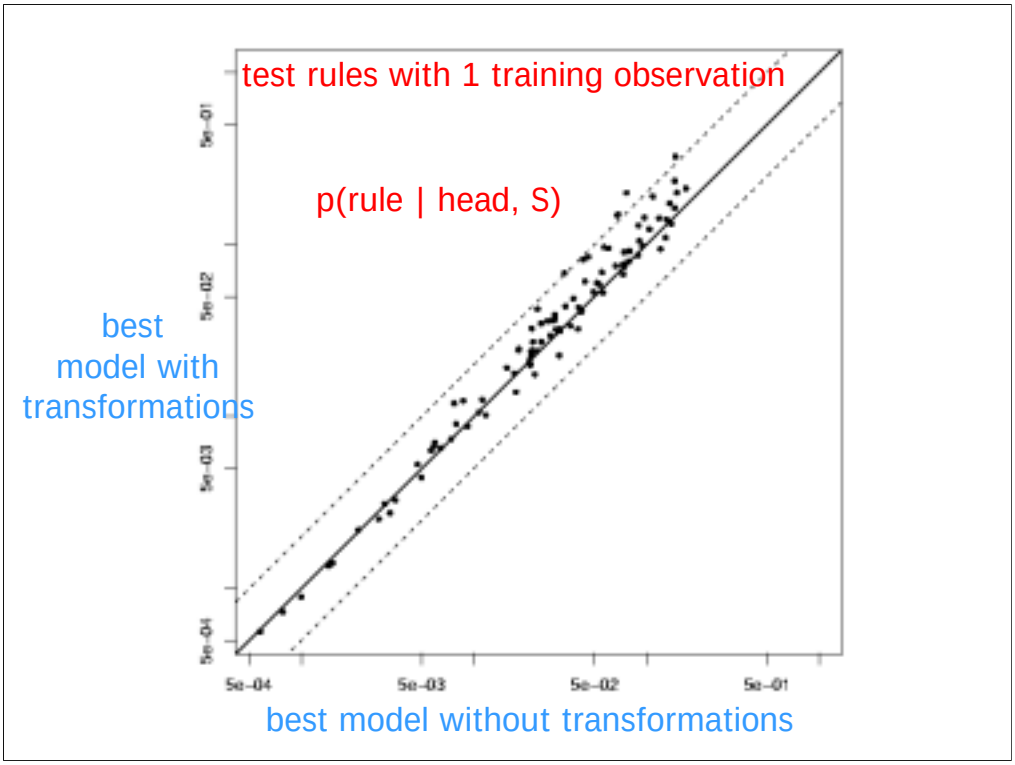
	basic		Treebank/Markov		
			Katz	one-count[a]	
	flat	non-flat[b]	flat	flat	non-flat
Treebank	∞	∞			
1-gram	1774.9	86435.1	340.9	160.0	193.2
2-gram	**135.2**	199.3	127.2	**116.2**	174.7
3-gram	136.5	177.4	132.7	123.3	174.8
Collins[c]	363.0	494.5	197.9		
transformation	**108.6**				
averaged[d]	**102.3**				

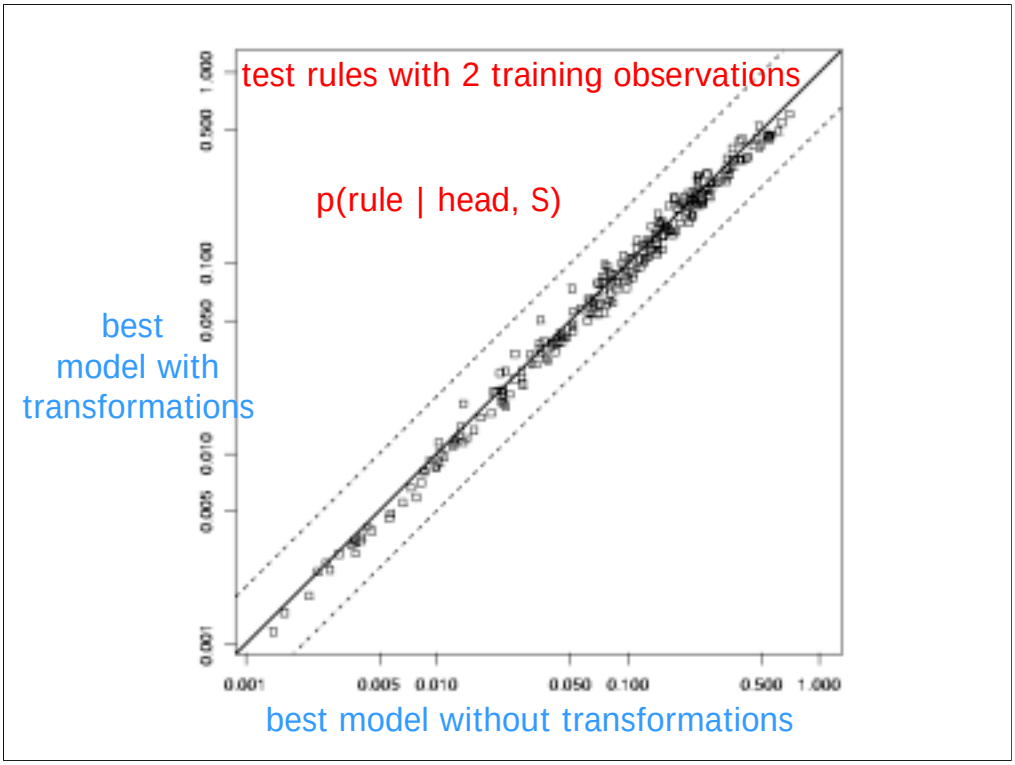
best model without transformations

from previous lit

best model with transformations


test rules with 0 training observations
p(rule | head, S)
best
model with
transformations
1e-01
1e-04
1e-07
1e-10
1e-10
1e-07
1e-04
1e-01
best model without transformations


test rules with 1 training observation
p(rule | head, S)
best
model with
transformations
5e-01
5e-02
5e-03
5e-04
5e-04
5e-03
5e-02
5e-01
best model without transformations


test rules with 2 training observations
p(rule | head, S)
best
model with
transformations
0.001
0.005
0.010
0.050
0.100
0.500
1.000
best model without transformations

Forced matching task

- Test model's ability to extrapolate novel frames for a word
- Randomly select **two** (word, frame) pairs from test data
  - ... ensuring that neither frame was ever seen in training
- Ask model to choose a matching:

i.e., does frame A look more like word 1's known frames or word 2's?

- 20% fewer errors than bigram model

Graceful degradation

	basic		memorization+backoff Katz		
				one-count	
	flat	non-flat	flat	flat	non-flat
1-gram	1991.2	96318.8	455.1	194.3	233.1
2-gram	**162.2**	236.6	153.2	**138.8**	205.6
3-gram	161.9	211.0	156.8	145.7	208.1
subcat	414.5	589.4	242.0		
transf	**124.8**				
combined	**118.0**				

Twice as much data
But no transformations

	basic		memorization+backoff Katz		
				one-count	
	flat	non-flat	flat	flat	non-flat
1-gram	1774.9	86435.1	340.9	160.0	193.2
2-gram	**135.2**	199.3	127.2	**116.2**	174.7
3-gram	136.5	177.4	132.7	123.3	174.8
subcat	363.0	494.5	197.9		

Even when you take away half of the transformation model's data, it still wins.

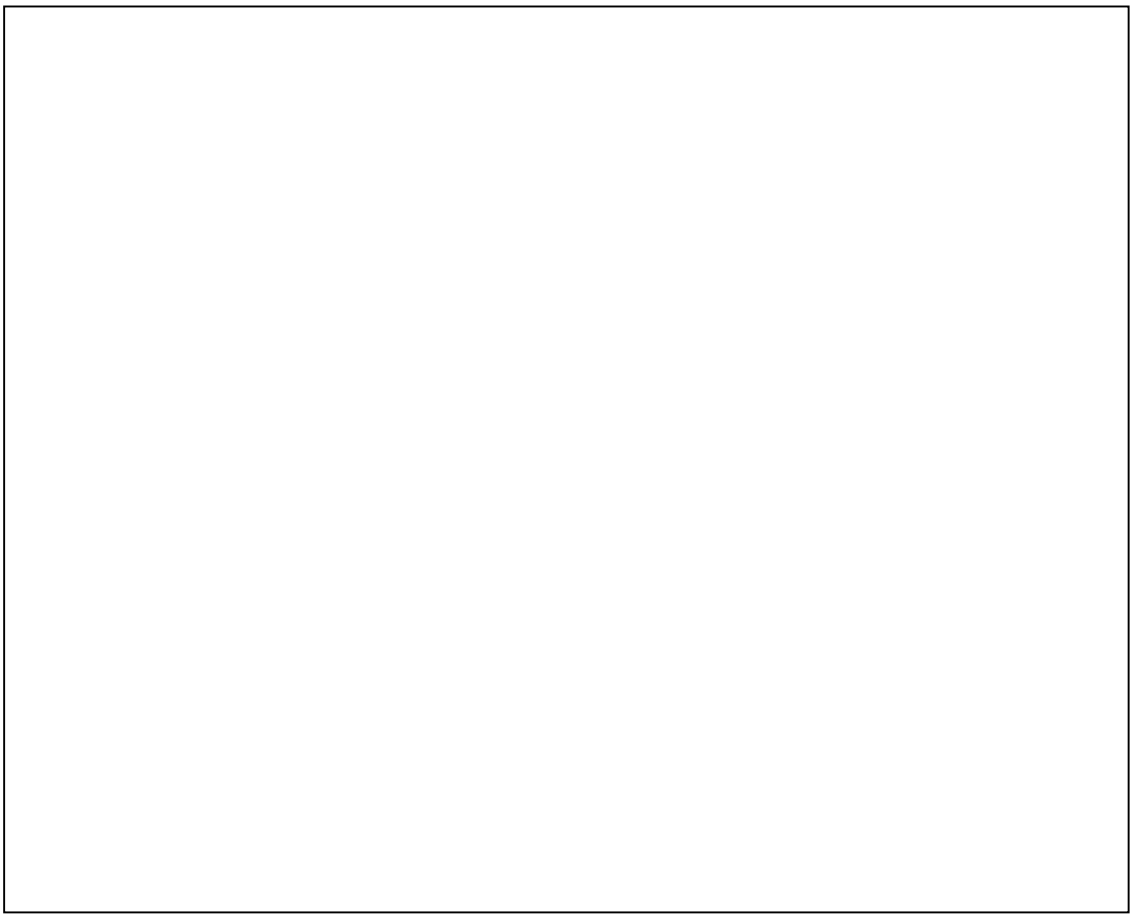
Or in the case of hybrid models, it's about a tie.

So these kinds of perplexity reductions were comparable to a twofold increase in the amount of training data.

Summary: Reparameterize PCFG in terms of deep transformation weights, to be learned under a simple prior.

- **Problem:** Too many rules!
  - Especially with lexicalization and flattening (which help).
  - So it's hard to estimate probabilities.
- **Solution:** Related rules tend to have related probs
  - *POSSIBLE* relationships are given a priori
  - *LEARN* which relationships are strong in this language

    (just like feature selection)
- Method has connections to:
  - Parameterized finite-state machines (Monday's talk)
  - Bayesian networks (inference, abduction, explaining away)
  - Linguistic theory (transformations, metarules, etc.)




FIN