Stochastic Context Free Grammars for RNA Structure Modeling

BMI/CS 776

www.biostat.wisc.edu/bmi776/

Mark Craven

craven@biostat.wisc.edu

Spring 2009

Modeling RNA with Stochastic Context Free Grammars

- consider tRNA genes
  - 274 in yeast genome, ~1500 in human genome
  - get transcribed, like protein-coding genes
  - don't get translated, therefore base statistics much different than protein-coding genes
  - but secondary structure is conserved
- to recognize new tRNA genes, model known ones using stochastic context free grammars [Eddy & Durbin, 1994; Sakakibara et al. 1994]
- but what is a grammar?

Transformational Grammars

- a transformational grammar characterizes a set of legal strings
- the grammar consists of
  - a set of abstract *nonterminal* symbols

    $$\{s,\ c_1,\ c_2,\ c_3,\ c_4\}$$

  - a set of *terminal* symbols (those that actually appear in strings)

    $$\{\mathrm{A},\ \mathrm{C},\ \mathrm{G},\ \mathrm{U}\}$$

  - a set of *productions*

    $$c_1 \rightarrow \mathrm{U}c_2 \qquad c_2 \rightarrow \mathrm{A}c_3 \qquad c_3 \rightarrow \mathrm{A}$$

    $$c_2 \rightarrow \mathrm{G}c_4 \qquad c_3 \rightarrow \mathrm{G}$$

    $$c_4 \rightarrow \mathrm{A}$$

A Grammar for Stop Codons

$$s \rightarrow c_1 \quad c_1 \rightarrow \mathrm{U}c_2 \quad c_2 \rightarrow \mathrm{A}c_3 \quad c_3 \rightarrow \mathrm{A} \quad c_4 \rightarrow \mathrm{A}$$

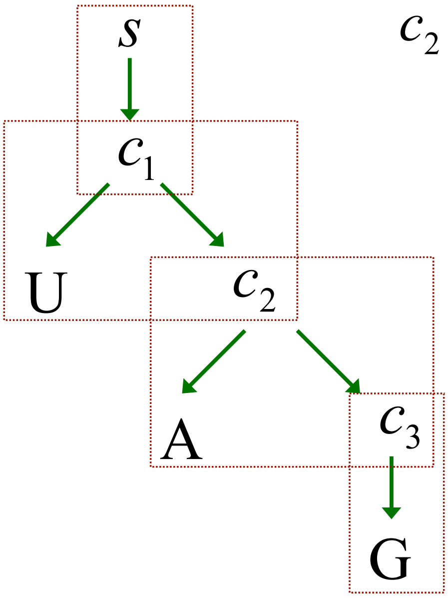
$$c_2 \rightarrow \mathrm{G}c_4 \quad c_3 \rightarrow \mathrm{G}$$

- this grammar can generate the 3 stop codons: UAA, UAG, UGA
- with a grammar we can ask questions like
  - what strings are derivable from the grammar?
  - can a particular string be derived from the grammar?

The Parse Tree for UAG

$s \to c_1 \quad c_1 \to \mathrm{U}c_2 \quad c_2 \to \mathrm{A}c_3 \quad c_3 \to \mathrm{A} \quad c_4 \to \mathrm{A}$

$c_2 \to \mathrm{G}c_4 \quad c_3 \to \mathrm{G}$



The Derivation for UAG

$s \to c_1 \quad c_1 \to \mathrm{U}c_2 \quad c_2 \to \mathrm{A}c_3 \quad c_3 \to \mathrm{A} \quad c_4 \to \mathrm{A}$

$c_2 \to \mathrm{G}c_4 \quad c_3 \to \mathrm{G}$

$$s \Rightarrow c_1 \Rightarrow \mathrm{U}c_2 \Rightarrow \mathrm{UA}c_3 \Rightarrow \mathrm{UAG}$$

Some Shorthand

$$c_2 \rightarrow \mathrm{A}c_3$$
$$c_2 \rightarrow \mathrm{G}c_4$$

$$\Longleftrightarrow$$

$$c_2 \rightarrow \mathrm{A}c_3 \mid \mathrm{G}c_4$$

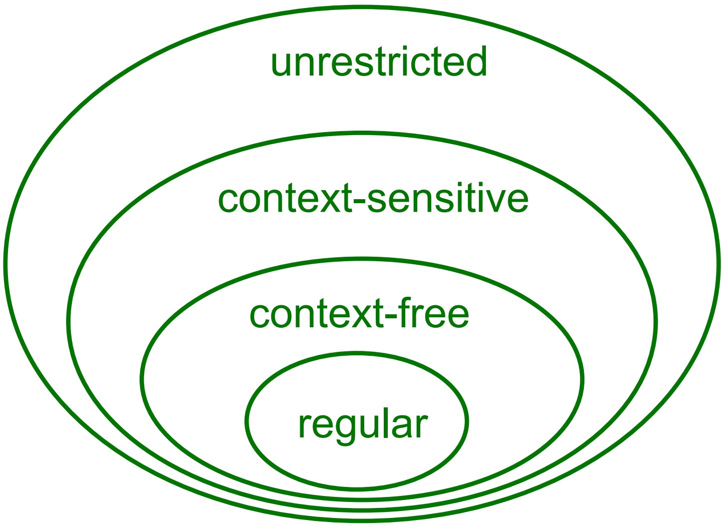
A Probabilistic Version of the Grammar

$$s \xrightarrow{1.0} c_1 \qquad c_1 \xrightarrow{1.0} \mathrm{U}c_2 \qquad c_2 \xrightarrow{0.7} \mathrm{A}c_3 \qquad c_3 \xrightarrow{0.2} \mathrm{A} \qquad c_4 \xrightarrow{1.0} \mathrm{A}$$

$$c_2 \xrightarrow{0.3} \mathrm{G}c_4 \qquad c_3 \xrightarrow{0.8} \mathrm{G}$$

- each production has an associated probability
- the probabilities for productions with the same left-hand side sum to 1
- *this* grammar has a corresponding Markov chain model

The Chomsky Hierarchy



- a hierarchy of grammars defined by restrictions on productions

The Chomsky Hierarchy

- regular grammars $u \rightarrow \mathrm{X}v \qquad u \rightarrow \mathrm{X}$
- context-free grammars $u \rightarrow \beta$
- context-sensitive grammars $\alpha_1 u \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$
- unrestricted grammars $\alpha_1 u \alpha_2 \rightarrow \gamma$

u, v are nonterminals

X is a terminal

α, γ are any sequence of terminals/nonterminals

β is any non-null sequence of terminals/nonterminals

CFGs and RNA

- context free grammars are well suited to modeling RNA secondary structure because they can represent base pairing preferences
- a grammar for a 3-base stem with and a loop of either GAAA or GCAA

$$s \rightarrow \mathrm{A}w_1\mathrm{U} \mid \mathrm{C}w_1\mathrm{G} \mid \mathrm{G}w_1\mathrm{C} \mid \mathrm{U}w_1\mathrm{A}$$

$$w_1 \rightarrow \mathrm{A}w_2\mathrm{U} \mid \mathrm{C}w_2\mathrm{G} \mid \mathrm{G}w_2\mathrm{C} \mid \mathrm{U}w_2\mathrm{A}$$

$$w_2 \rightarrow \mathrm{A}w_3\mathrm{U} \mid \mathrm{C}w_3\mathrm{G} \mid \mathrm{G}w_3\mathrm{C} \mid \mathrm{U}w_3\mathrm{A}$$

$$w_3 \rightarrow \mathrm{GAAA} \mid \mathrm{GCAA}$$

CFGs and RNA

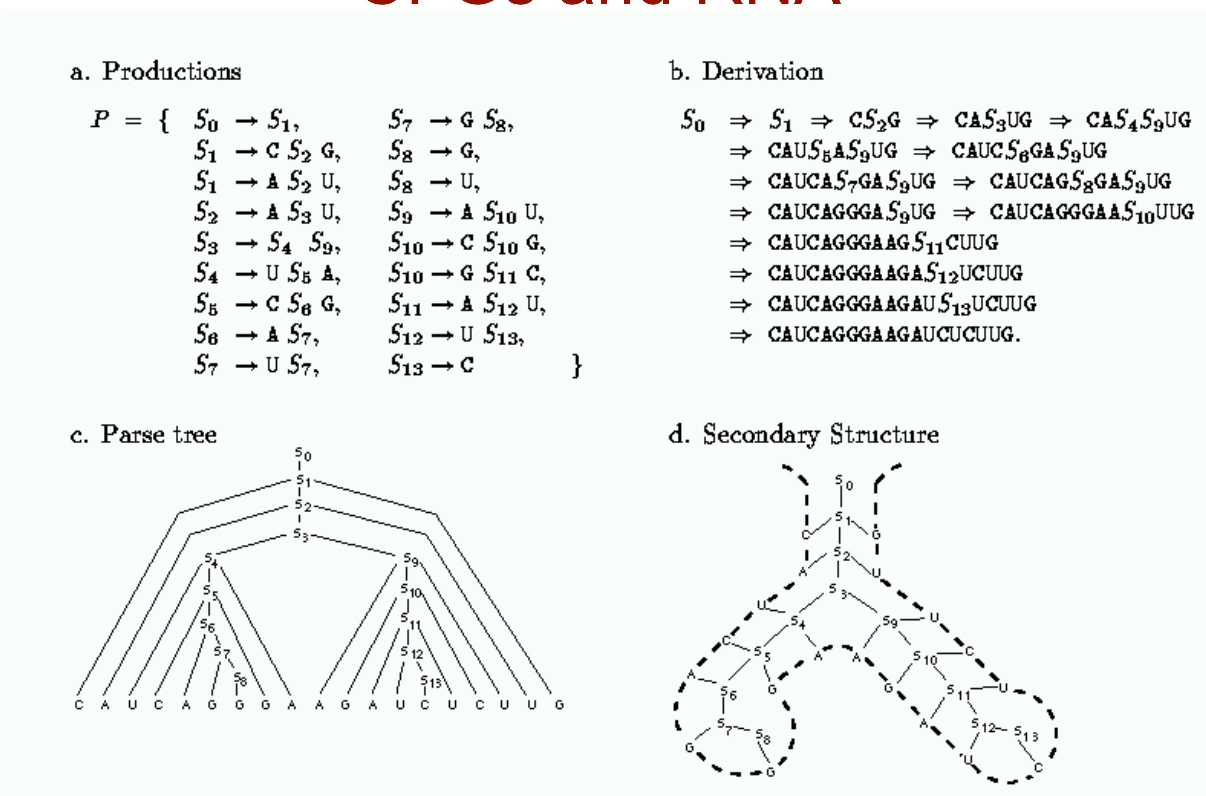
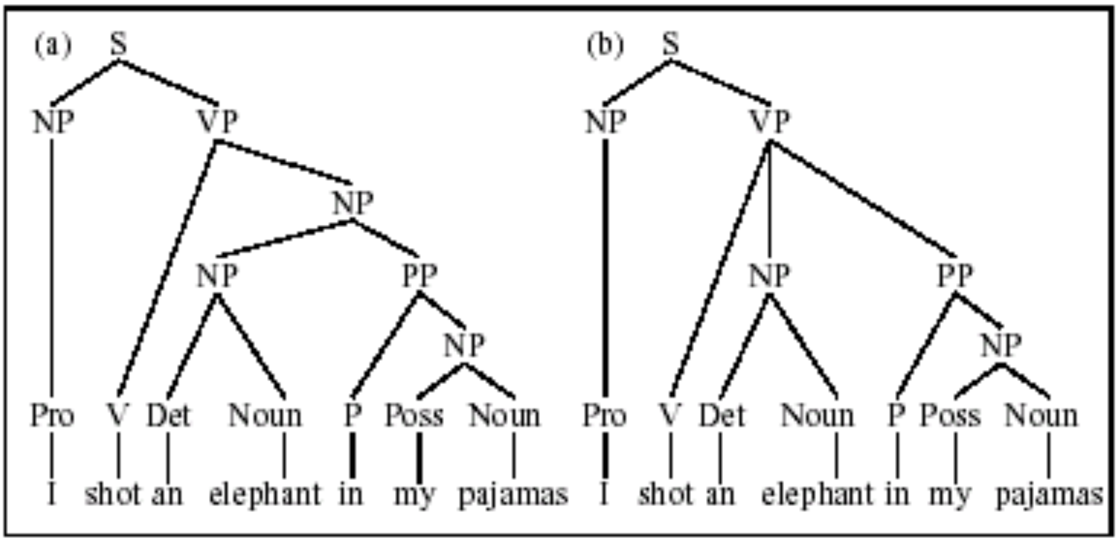


Figure from: Sakakibara et al. *Nucleic Acids Research*, 1994

Ambiguity in Parsing

"I shot an elephant in my pajamas. How he got in my pajamas, I'll never know." – Groucho Marx



Ambiguity in Parsing

$$s \to Aw_1U \mid Cw_1G \mid Gw_1C \mid Uw_1A$$

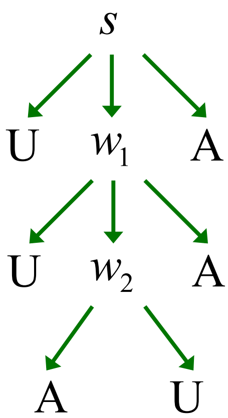
$$w_1 \to Aw_2U \mid Cw_2G \mid Gw_2C \mid Uw_2A$$

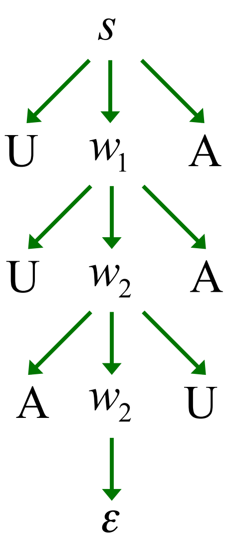
$$w_2 \to Aw_2U \mid Cw_2G \mid Gw_2C \mid Uw_2A$$

$$w_2 \to AA \mid AU$$

$$w_2 \to \varepsilon$$

denotes the empty string





Stochastic Context Free Grammars

$$s \rightarrow \overset{0.25}{\mathrm{A}w_1\mathrm{U}} \mid \overset{0.25}{\mathrm{C}w_1\mathrm{G}} \mid \overset{0.25}{\mathrm{G}w_1\mathrm{C}} \mid \overset{0.25}{\mathrm{U}w_1\mathrm{A}}$$

$$w_1 \rightarrow \overset{0.1}{\mathrm{A}w_2\mathrm{U}} \mid \overset{0.4}{\mathrm{C}w_2\mathrm{G}} \mid \overset{0.4}{\mathrm{G}w_2\mathrm{C}} \mid \overset{0.1}{\mathrm{U}w_2\mathrm{A}}$$

$$w_2 \rightarrow \overset{0.25}{\mathrm{A}w_3\mathrm{U}} \mid \overset{0.25}{\mathrm{C}w_3\mathrm{G}} \mid \overset{0.25}{\mathrm{G}w_3\mathrm{C}} \mid \overset{0.25}{\mathrm{U}w_3\mathrm{A}}$$

$$w_3 \rightarrow \overset{0.8}{\mathrm{GAAA}} \mid \overset{0.2}{\mathrm{GCAA}}$$

Stochastic Grammars?

…the notion "probability of a sentence" is an entirely useless one, under any known interpretation of this term.

— Noam Chomsky
(famed linguist)

Every time I fire a linguist, the performance of the recognizer improves.

— Fred Jelinek
(former head of IBM speech recognition group)

Credit for pairing these quotes goes to Dan Jurafsky and James Martin, *Speech and Language Processing*

Three Key Questions

- How likely is a given sequence?

  the Inside algorithm
- What is the most probable parse for a given sequence?

  the Cocke-Younger-Kasami (CYK) algorithm
- How can we learn the SCFG parameters given a grammar and a set of sequences?

  the Inside-Outside algorithm

Chomsky Normal Form

- it is convenient to assume that our grammar is in *Chomsky Normal Form*; i.e all productions are of the form:

  $v \rightarrow yz$ right hand side consists of two nonterminals

  $v \rightarrow A$ right hand side consists of a single terminal

- any CFG can be put into Chomsky Normal Form

Parameter Notation

- for productions of the form $v \rightarrow yz$, we'll denote the associated probability parameters

$$t_v(y,z)$$ transition

- for productions of the form $v \rightarrow A$, we'll denote the associated probability parameters

$$e_v(A)$$ emission

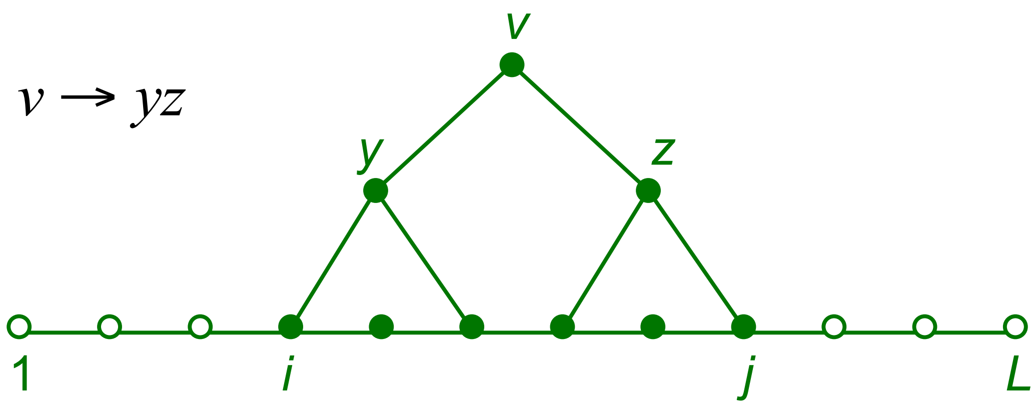
Determining the Likelihood of a Sequence: The Inside Algorithm

- a dynamic programming method, analogous to the Forward algorithm
- involves filling in a 3D matrix

$$\alpha(i, j, v)$$

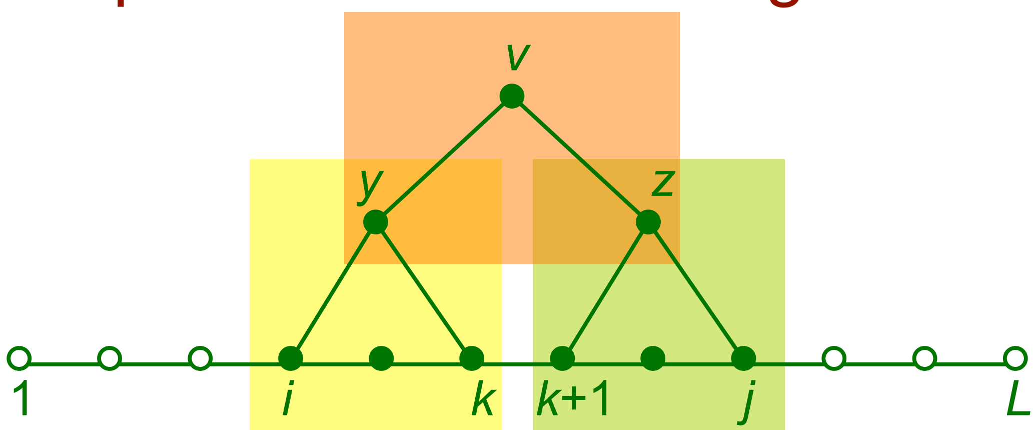
representing the probability of the all parse subtrees rooted at nonterminal v for the subsequence from i to j

Determining the Likelihood of a Sequence: The Inside Algorithm



- $\alpha(i, j, v)$: the probability of all parse subtrees rooted at nonterminal v for the subsequence from i to j

Determining the Likelihood of a Sequence: The Inside Algorithm



$$\alpha(i, j, v) = \sum_{y=1}^{M} \sum_{z=1}^{M} \sum_{k=i}^{j-1} \alpha(i, k, y) \; \alpha(k+1, j, z) \; t_v(y, z)$$

M is the number of nonterminals in the grammar

The Inside Algorithm

- initialization (for $i = 1$ to L, $v = 1$ to M)

$$\alpha(i,i,v) = e_v(x_i)$$

- iteration (for $i = 1$ to $L - 1$, $j = i+1$ to L, $v = 1$ to M)

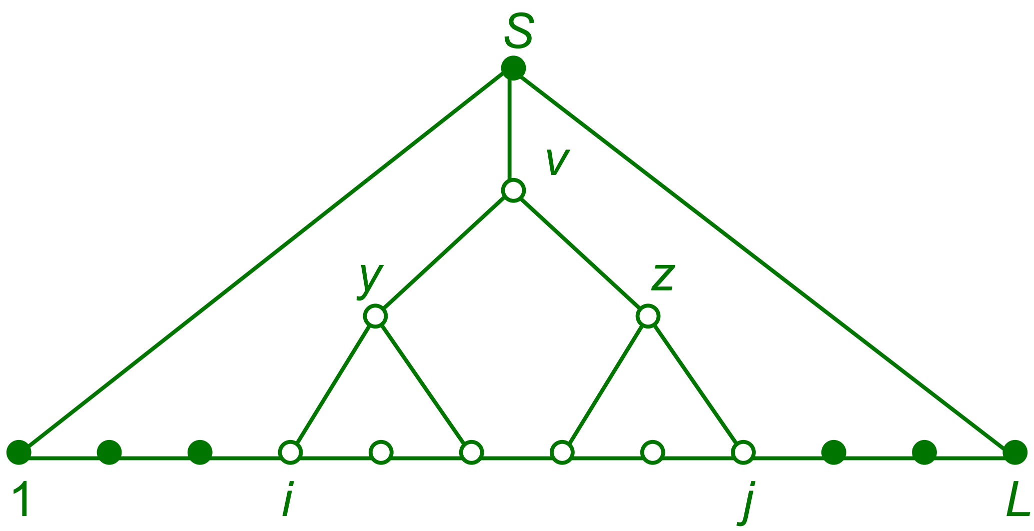
$$\alpha(i,j,v) = \sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=i}^{j-1} \alpha(i,k,y)\alpha(k+1,j,z)\, t_v(y,z)$$

- termination

$$\Pr(x) = \alpha(1,L,1)$$

start nonterminal

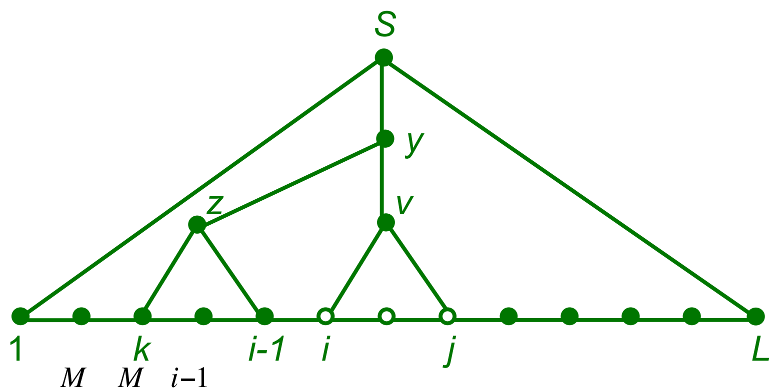
The Outside Algorithm



- $\beta(i,j,v)$: the probability of parse trees rooted at the start nonterminal, exluding the probability of all subtrees rooted at nonterminal v covering the subsequence from i to j

The Outside Algorithm

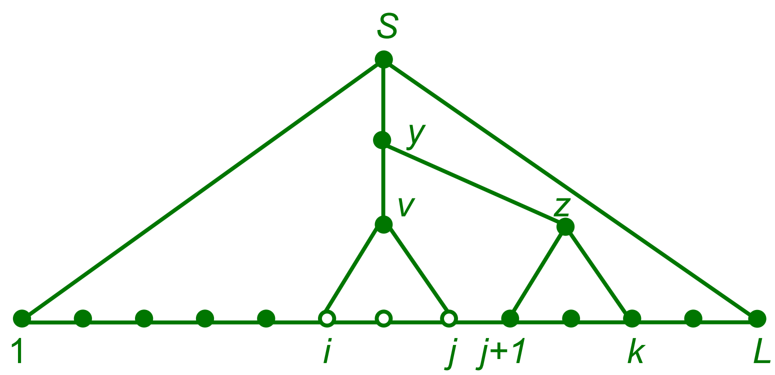
- we can recursively calculate $\beta(i, j, v)$ from β values we've calculated for y
- the first case we consider is where v is used in productions of the form: $y \rightarrow zv$



$$\sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=1}^{i-1}\alpha(k,i-1,z)\beta(k,j,y)\,t_y(z,v)$$

The Outside Algorithm

- the second case we consider is where v is used in productions of the form: $y \rightarrow vz$



$$\sum_{y=1}^{M}\sum_{z=1}^{M}\sum_{k=j+1}^{L}\alpha(j+1,k,z)\beta(i,k,y)\,t_y(v,z)$$

The Outside Algorithm

- initialization

$$\beta(1, L, 1) = 1 \qquad \text{(the } \textit{start} \text{ nonterminal)}$$

$$\beta(1, L, v) = 0 \qquad \text{for } v = 2 \text{ to } M$$

- iteration (for i = 1 to L, j = L to i, v = 1 to M)

$$\beta(i, j, v) = \sum_{y=1}^{M} \sum_{z=1}^{M} \sum_{k=1}^{i-1} \alpha(k, i-1, z) \beta(k, j, y)\, t_y(z, v) +$$

$$\sum_{y=1}^{M} \sum_{z=1}^{M} \sum_{k=j+1}^{L} \alpha(j+1, k, z) \beta(i, k, y)\, t_y(v, z)$$

Learning SCFG Parameters

- if we know the parse tree for each training sequence, learning the SCFG parameters is simple
  - no hidden state during training
  - count how often each parameter (i.e. production) is used
  - normalize/smooth to get probabilities

- more commonly, there are many possible parse trees per sequence – we don't know which one is correct
  - thus, use an EM approach (Inside-Outside)
  - iteratively
    - determine expected # times each production is used
      - consider all parses
      - weight each by it's probability
    - set parameters to maximize these counts

The Inside-Outside Algorithm

- we can learn the parameters of an SCFG from training sequences using an EM approach called Inside-Outside
- in the E-step, we determine
  - the expected number of times each *nonterminal* is used in parses $c(v)$
  - the expected number of times each *production* is used in parses $c(v \rightarrow yz)$

    $c(v \rightarrow A)$
- in the M-step, we update our production probabilities

The Inside-Outside Algorithm

- the EM re-estimation equations (for 1 sequence) are:

$$\hat{e}_v(A) = \frac{c(v \rightarrow A)}{c(v)} = \frac{\sum_{i \mid x_i = A} \beta(i,i,v) e_v(A)}{\sum_{i=1}^{L} \sum_{j=i}^{L} \beta(i,j,v) \alpha(i,j,v)}$$

cases where v used to generate A

cases where v used to generate any subsequence

$$\hat{t}_v(y,z) = \frac{c(v \rightarrow yz)}{c(v)}$$

$$= \frac{\sum_{i=1}^{L-1} \sum_{j=i+1}^{L} \sum_{k=i}^{j-1} \beta(i,j,v)\ t_v(y,z)\ \alpha(i,k,y)\ \alpha(k+1,j,z)}{\sum_{i=1}^{L} \sum_{j=i}^{L} \beta(i,j,v)\ \alpha(i,j,v)}$$

The CYK Algorithm

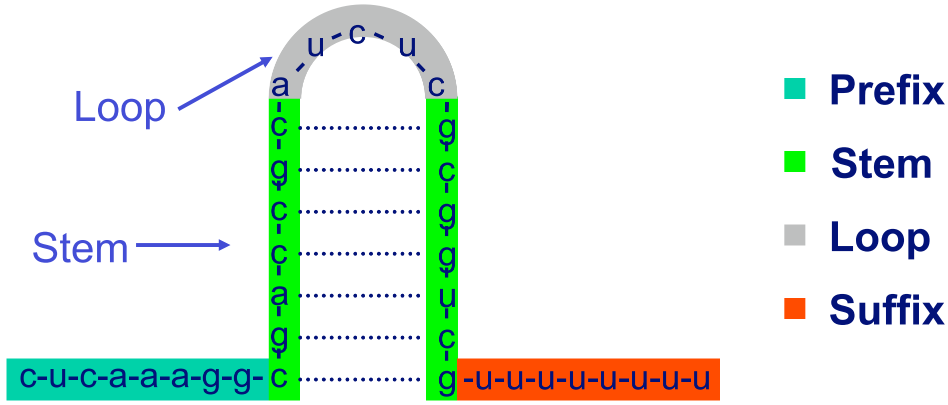
- analogous to Viterbi algorithm
- like Inside algorithm but
  - max operations instead of sums
  - retain traceback pointers
- traceback is a little more involved than Viterbi
  - need to reconstruct parse tree instead of recovering simple path

Comparison of SCFG Algorithms to HMM Algorithms

	HMM algorithm	SCFG algorithm
optimal alignment	Viterbi	CYK
probability of sequence	forward	inside
EM parameter estimation	forward-backward	inside-outside
memory complexity	$O(LM)$	$O(L^2M)$
time complexity	$O(LM^2)$	$O(L^3M^3)$

Recognizing Terminators with SCFGs

- [Bockhorst & Craven, *IJCAI* 2001]



- a prototypical terminator has the structure above
- the lengths and base compositions of the elements can vary a fair amount

Our Initial Terminator Grammar

START → PREFIX STEM_BOT1 SUFFIX
PREFIX → B B B B B B B B B
STEM_BOT1 → t_l STEM_BOT2 t_r
STEM_BOT2 → t_l^* STEM_MID t_r^* | t_l^* STEM_TOP2 t_r^*
STEM_MID → t_l^* STEM_MID t_r^* | t_l^* STEM_TOP2 t_r^*
STEM_TOP2 → t_l^* STEM_TOP1 t_r^*
STEM_TOP1 → t_l LOOP t_r
LOOP → B B LOOP_MID B B
LOOP_MID → B LOOP_MID | ε
SUFFIX → B B B B B B B B B
B → a | c | g | u

Nonterminals are uppercase, terminals are lowercase

t = {a,c,g,u}, t^* = {a,c,g,u,ε}

Our Initial Terminator Grammar

START → PREFIX STEM_BOT1 SUFFIX

PREFIX → B B B B B B B B B

STEM_BOT1 → t_l STEM_BOT2 t_r

* | t_l^* STEM_TOP2 t_r^*

STEM_TOP2 t_r^*

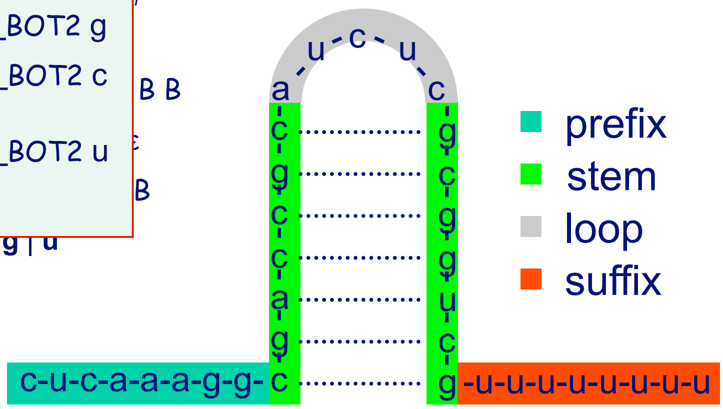
B B

B

B → **a** | **c** | **g** | **u**

STEM_BOT1	0.2 →	a STEM_BOT2 u	
STEM_BOT1	0.2 →	u STEM_BOT2 a	
STEM_BOT1	0.2 →	c STEM_BOT2 g	
STEM_BOT1	0.2 →	g STEM_BOT2 c	
STEM_BOT1	0.05 →	g STEM_BOT2 u	
	⋮		

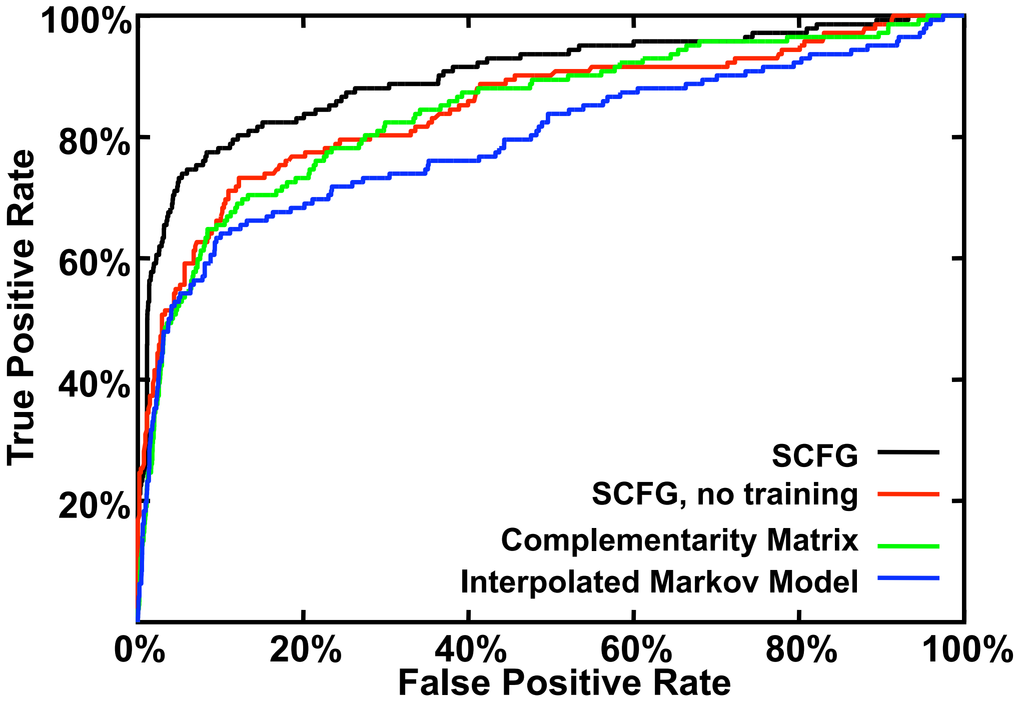
t = {**a**,**c**,**g**,**u**},
t^* = {**a**,**c**,**g**,**u**, ε}



Terminator SCFG Experiments

- compare predictive accuracy of
  - SCFG with learned parameters
  - SCFG without learning (but parameters initialized using domain knowledge)
  - interpolated Markov models (IMMs)
    - can represent distribution of bases at each position
    - * cannot easily encode base pair dependencies
  - complementarity matrices
    - Brendel et al., *J Biom Struct and Dyn* 1986
    - ad hoc way of considering base pairings
    - * cannot favor specific base pairs by position

SCFGs vs. Related Methods



Learning SCFG Structure

- given the productions of a grammar, can learn the probabilities using the Inside-Outside algorithm
- we developed an algorithm that can add new nonterminals & productions to a grammar during learning [Bockhorst & Craven, *IJCAI* 01]

- basic idea:
  - identify nonterminals that seem to be “overloaded”
  - split these nonterminals into two; allow each to specialize

Refinement Algorithm Overview

Given:

- set of sequences
- initial grammar structure hypothesis

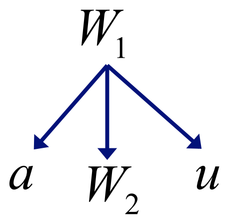
Do:

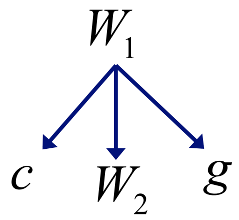
repeat

1) find MAP estimates for probabilities
2) refine grammar structure
   - 2.1) diagnostically identify overloaded nonterminal
   - 2.2) apply **EXPAND** operator

Refining the Grammar in a SCFG

- there are various “contexts” in which each grammar nonterminal may be used
- consider two contexts for the nonterminal W_2





Context $a\,W_2\,u$		Context $c\,W_2\,g$	
$W_2 \rightarrow \mathrm{a}W_3\mathrm{u}$ |	0.4	$W_2 \rightarrow \mathrm{a}W_3\mathrm{u}$ |	0.1
$\mathrm{c}W_3\mathrm{g}$ |	0.1	$\mathrm{c}W_3\mathrm{g}$ |	0.4
$\mathrm{g}W_3\mathrm{c}$ |	0.1	$\mathrm{g}W_3\mathrm{c}$ |	0.4
$\mathrm{u}W_3\mathrm{a}$	0.4	$\mathrm{u}W_3\mathrm{a}$	0.1

- if the probabilities for W_2 look very different, depending on its context, we add a new nonterminal and specialize

Refining the Grammar in a SCFG

- we can compare two probability distributions *P* and *Q* using Kullback-Leibler divergence

$$H(P \parallel Q) = \sum_i P(x_i) \frac{P(x_i)}{Q(x_i)}$$

	P		*Q*
$W_2 \rightarrow aW_3u \mid$	0.4	$W_2 \rightarrow aW_3u \mid$	0.1
$cW_3g \mid$	0.1	$cW_3g \mid$	0.4
$gW_3c \mid$	0.1	$gW_3c \mid$	0.4
uW_3a	0.4	uW_3a	0.1

- or we can compare expected number of times each production is used (over training set) using χ^2

EXPAND Operator

EXPAND(P,N) /* N on RHS of production P */

1) create new nonterminal N'

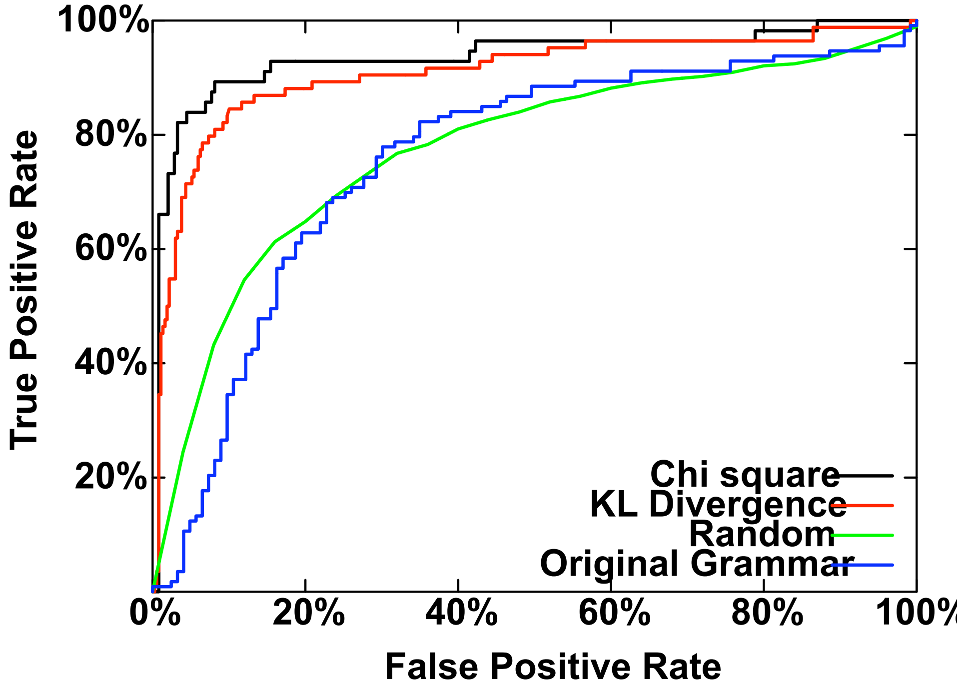
2) replace N with N' in P.

3) for each production N → •, create a production N' → •

Learning Terminator SCFGs

- extracted grammar from the literature (~ 120 productions)
- data set consists of 142 known *E. coli* terminators, 125 sequences that do not contain terminators
- learn parameters using Inside-Outside algorithm (an EM algorithm)
- consider adding nonterminals guided by three heuristics
  - KL divergence
  - chi-squared
  - random

SCFG Accuracy After Adding 25 New Nonterminals



SCFG Accuracy vs. Nonterminals Added