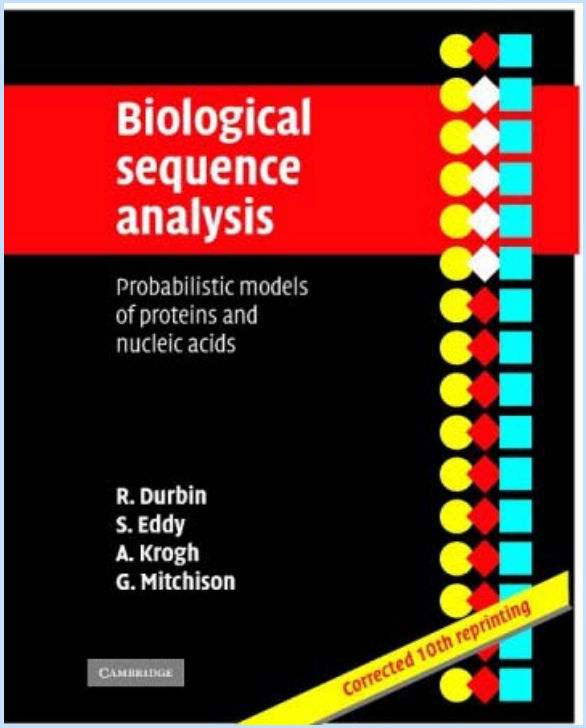
Computational Genomics
Prof. Ron Shamir & Prof. Roded Sharan
School of Computer Science, Tel Aviv University

גנומיקה חישובית
פרופ' רון שמיר ופרופ' רודד שרן
ביה"ס למדעי המחשב,אוניברסיטת תל אביב

Lecture 11: Stochastic context free grammar

1,3/1/13

RNA Structure & Stochastic Context Free Grammars



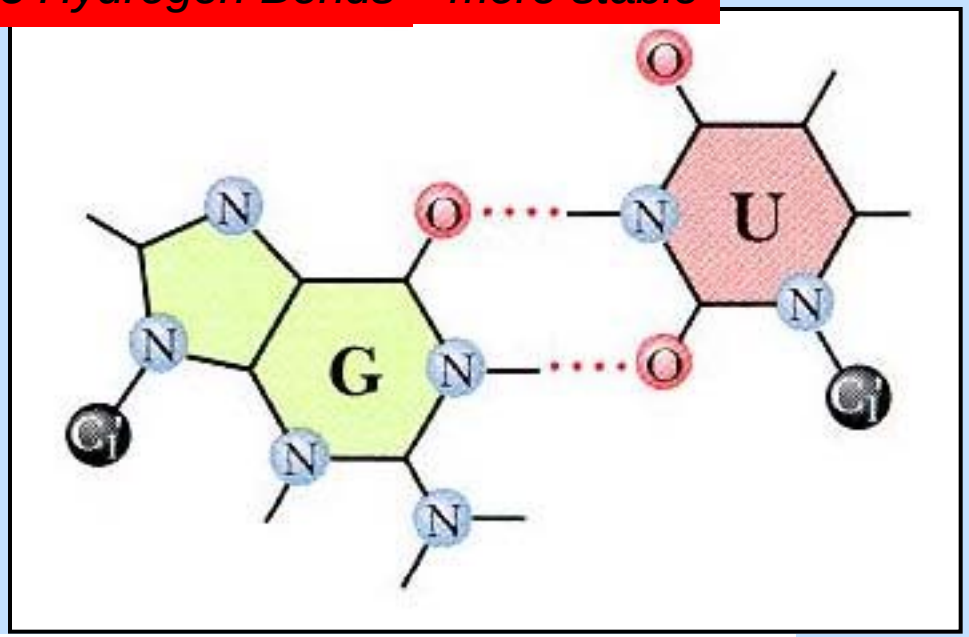
Main source: Durbin et al.,
"Biological Sequence Alignment"
(Cambridge, '98)

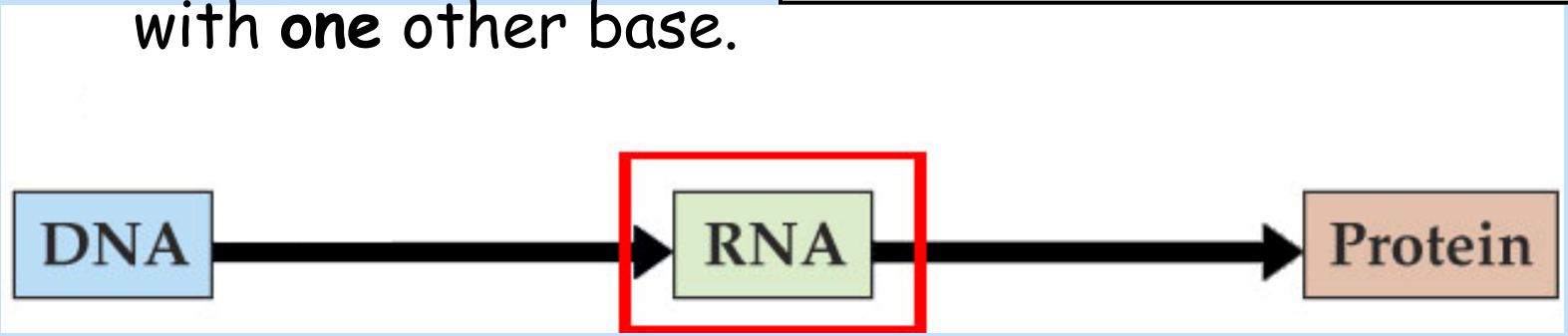
RNA Basics

- RNA bases A,C,G,U
- Canonical Base Pairs
  - A-U
  - G-C
  - G-U

    "wobble" pairing
  - Bases can only pair with **one** other base.

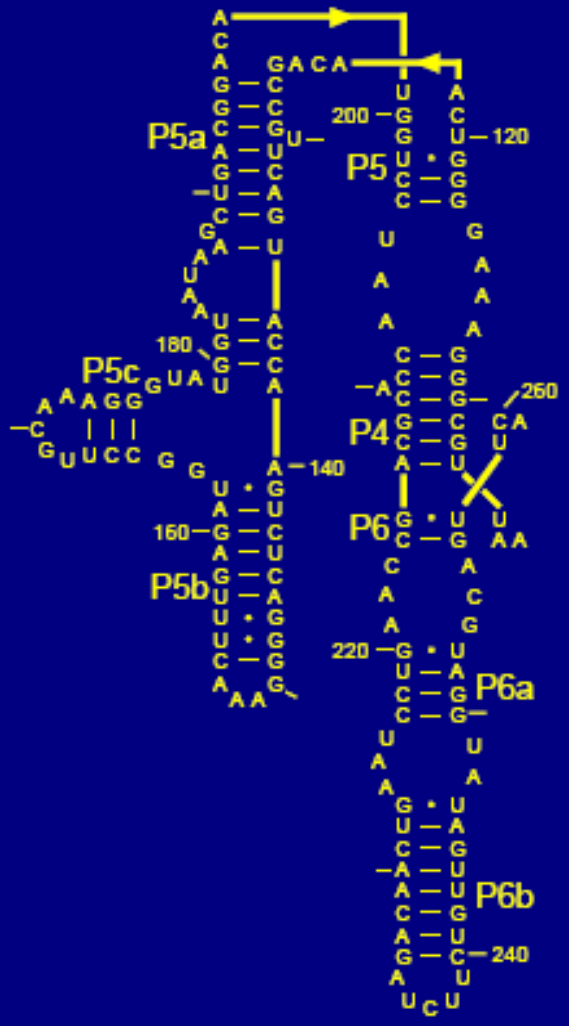
3 Hydrogen Bonds – more stable

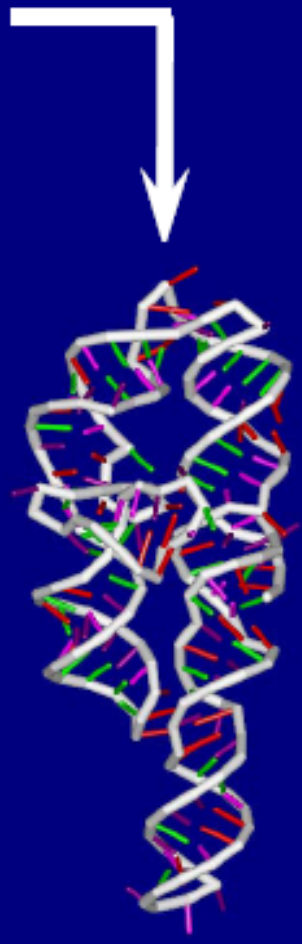




RNA Secondary and Tertiary Structure:

AAUUGCGGGAAAGGGGUCAA
CAGCCGUUCAGUACCAAGUC
UCAGGGGAAACUUUGAGAUG
GCCUUGCAAAGGGUAUGGUA
AUAAGCUGACGGACAUGGUC
CUAACCACGCAGCCAAGUCC
UAAGUCAACAGAUCUUCUGU
UGAUAUGGAUGCAGUUCA

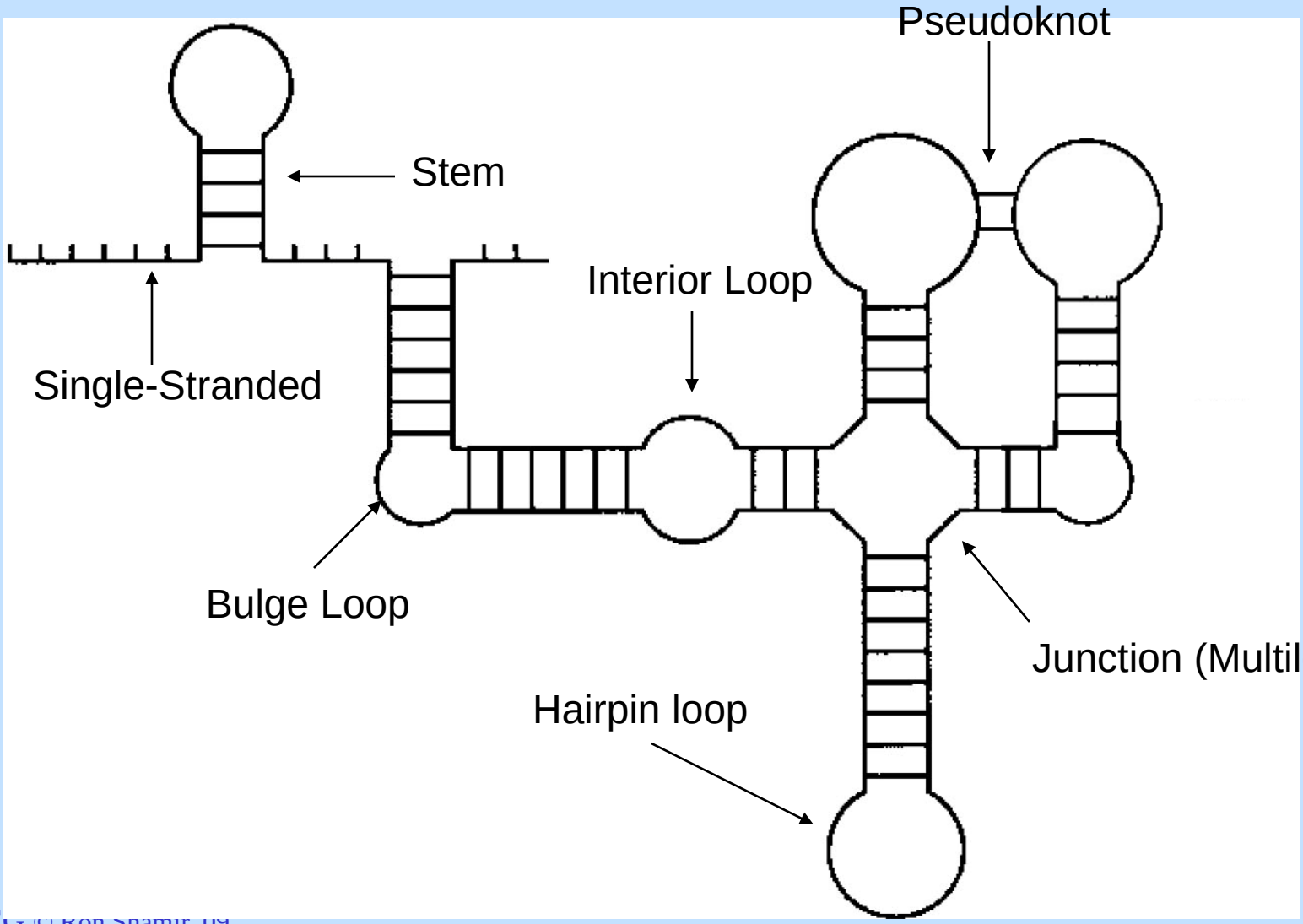




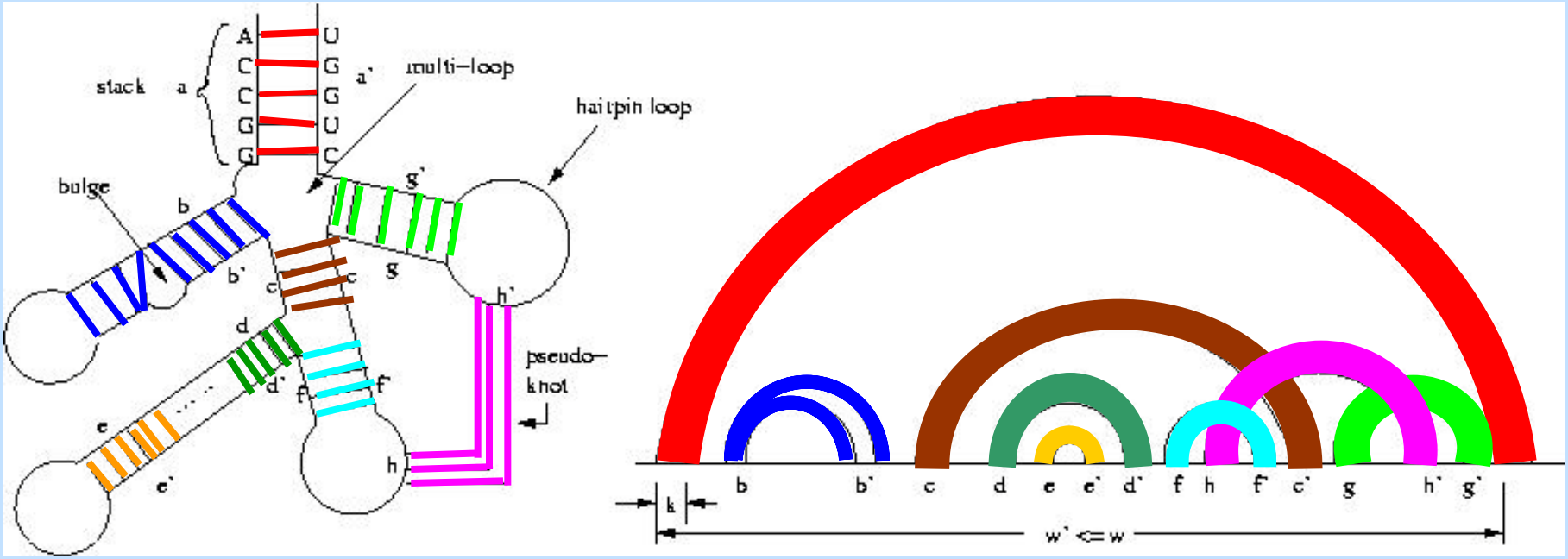
Waring & Davies.
(1984) *Gene 28*: 277.

Cate, *et al.* (Cech & Doudna).
(1996) *Science 273*:1678.

RNA Secondary Structure

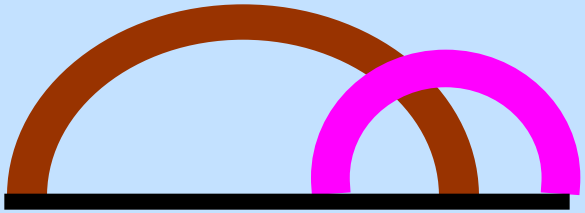


RNA Structure: Details



Pseudoknots

Intersecting base pairs



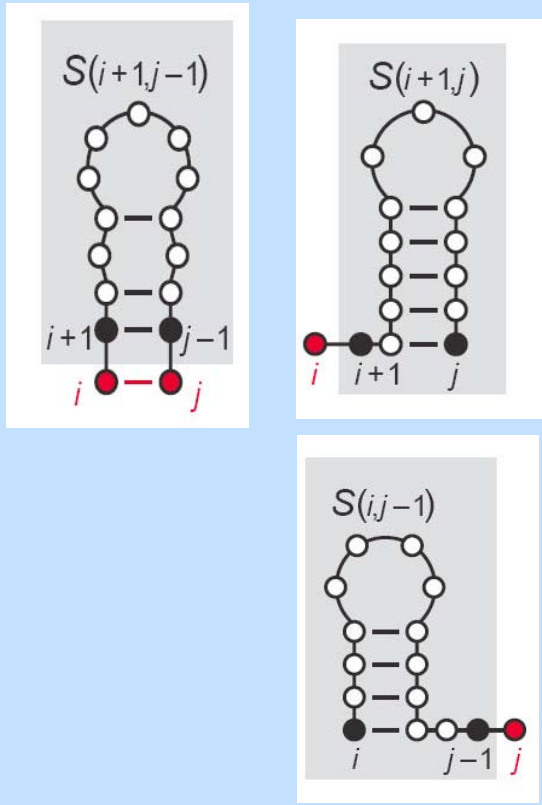
We shall assume no pseudoknots in the structure

Base Pair Maximization – Dynamic Programming Algorithm

(Nussinov-Jacobson 1978)

S(i,j) = max no. of base pairs in a folding of the subsequence from index i to index j

$$S(i,j) = \max \begin{cases} S(i+1,j-1)+1 & [\text{if } i,j \text{ base pair}] \\ S(i+1,j) \\ S(i,j-1) \\ \max_{i<k<j} S(i,k) + S(k+1,j) \end{cases}$$



Grammars

Grammars express languages

Example: the English language

$$\langle sentence \rangle \rightarrow \langle noun\_phrase \rangle \ \langle predicate \rangle$$

$$\langle noun\_phrase \rangle \rightarrow \langle article \rangle \ \langle noun \rangle$$

$$\langle predicate \rangle \rightarrow \langle verb \rangle$$

$$\langle article \rangle \rightarrow a$$

$$\langle article \rangle \rightarrow the$$

$$\langle noun \rangle \rightarrow boy$$

$$\langle noun \rangle \rightarrow dog$$

$$\langle verb \rangle \rightarrow runs$$

$$\langle verb \rangle \rightarrow walks$$

A derivation of "the boy walks":

$$\langle sentence \rangle \Rightarrow \langle noun\_phrase \rangle \ \langle predicate \rangle$$
$$\Rightarrow \langle noun\_phrase \rangle \ \langle verb \rangle$$
$$\Rightarrow \langle article \rangle \ \langle noun \rangle \ \langle verb \rangle$$
$$\Rightarrow the \ \langle noun \rangle \ \langle verb \rangle$$
$$\Rightarrow the \ boy \ \langle verb \rangle$$
$$\Rightarrow the \ boy \ walks$$

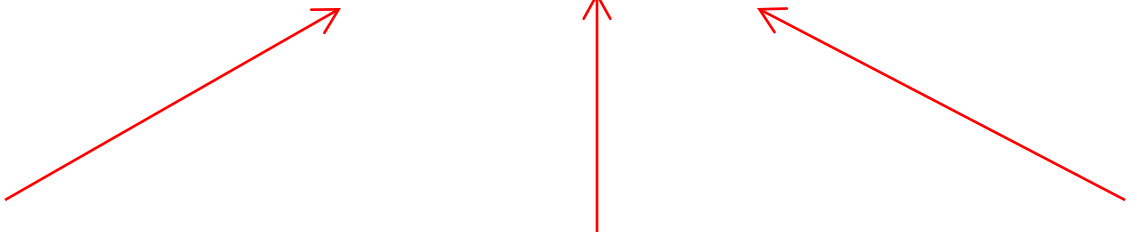
Language of the grammar:

L = { "a boy runs",
"a boy walks",
"the boy runs",
"the boy walks",
"a dog runs",
"a dog walks",
"the dog runs",
"the dog walks" }

Notation

$$\langle noun \rangle \rightarrow boy$$

$$\langle noun \rangle \rightarrow dog$$



Variable or Non-terminal

Production rule

Terminal

Another Example

Grammar:

$$S \to aSb$$

$$S \to \varepsilon$$

Derivation of sentence ab :

$$S \Rightarrow aSb \Rightarrow ab$$

$S \to aSb$ $\qquad$ $S \to \varepsilon$

Another derivation:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

Language of the grammar

$$L = \{a^n b^n : n \geq 0\}$$

The Chomsky Hierarchy

(αWβ-> α γ β; recognized by linear bounded automata)

(αWβ->γ)

unrestricted grammars

context-sensitive grammars

context-free grammars

regular grammars

(W->aW or W->a; recognized by finite automata; HMMs are equiv. to stochastic RGs)

(W->β; recognized by pushdown automata)

Context-free Grammars (CFG's)

A *context-free grammar* is a generative model

$G = (V, \alpha, S, R)$ where:

V is a *nonterminal alphabet*, (e.g., $\{A, B, C, D, E, \ldots\}$

α is a *terminal alphabet*, (e.g., $\{a, c, g, t\}$)

$S \in V$ is a special *start symbol*

R is a set of rewriting rules called *productions*.

Productions in R are rules of the form: $X \rightarrow \lambda$

where $X \in V$, $\lambda \in (V \cup \alpha)^*$

Context-freeness

The "*context-freeness*" is imposed by the requirement that the l.h.s of each production rule may contain only a **single** symbol, and that symbol must be a **nonterminal**:

$$X \rightarrow \lambda$$

Thus, a CFG cannot specify *context-sensitive* rules such as:

$$wXz \rightarrow w\lambda z$$

Derivations

Suppose a CFG G has generated a *terminal string* $x \in \alpha^*$. A *derivation* $S \Rightarrow^* x$ denotes a possible way for generating x.

A *derivation* (or *parse*) consists of a series of applications of productions from R, beginning with the *start symbol* S and ending with the *terminal string* x:

$$S \Rightarrow s_1 \Rightarrow s_2 \Rightarrow s_3 \Rightarrow \cdots \Rightarrow x$$

where $s_i \in (V \cup \alpha)^*$.

We will concentrate on leftmost derivations, where the leftmost nonterminal is always replaced first.

Context-free Versus Regular

The advantage of CFGs over RGs/HMMs lies in their ability to model arbitrary runs of matching pairs of elements, such as matching pairs of parentheses:

$$\cdots(((((((\cdots)))))))\cdots$$

When the number of matching pairs is unbounded, a finite-state model such as an HMM is inadequate to enforce the constraint that all left elements must have a matching right element.

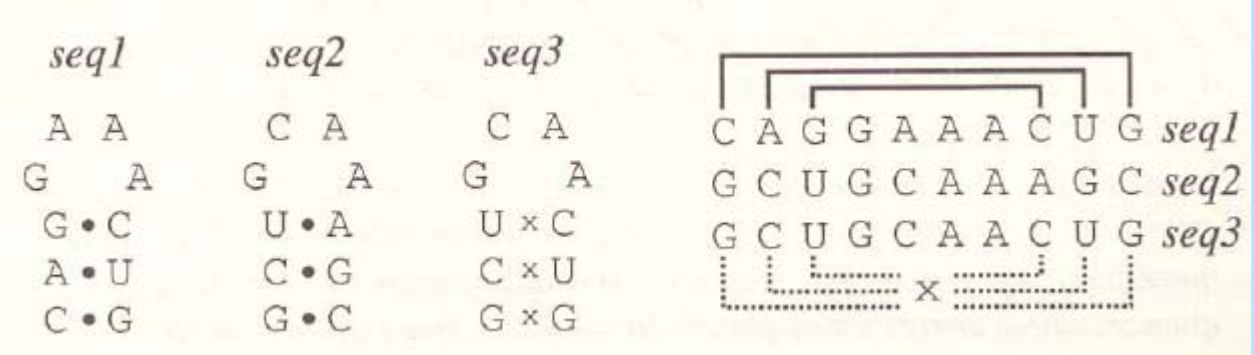
In contrast, in a CFG we can use rules such as $X \rightarrow (X)$. A sample derivation using such a rule is:

$$X \Rightarrow (X) \Rightarrow ((X)) \Rightarrow (((X))) \Rightarrow ((((X)))) \Rightarrow (((((X)))))$$

An additional rule such as $X \rightarrow \varepsilon$ is necessary to terminate the recursion.

A CFG for an RNA stem loop

- Wish a CFG that models RNA stem loops with 3 bps and a GCAA or GAAA loop.



S -> aXu | cXg | gXc | uXa

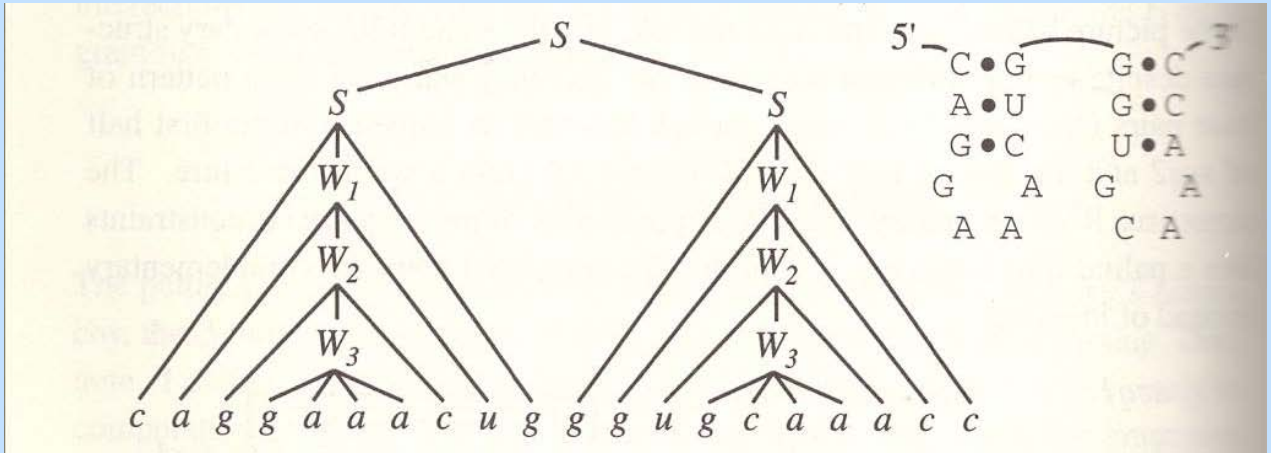
X -> aYu | cYg | gYc | uYa

Y -> aZu | cZg | gZc | uZa

Z -> gaaa | gcaa

Parse trees

- A representation of a parse of a string by a CFG (in 1-1 correspondence with left derivations)
- **Root –** start nonterminal S
- **Leaves –** terminal symbols in the given string
- **Internal nodes –** non-terminals
- The children of an internal node are the productions of that nonterminal (left-to-right order)
- A subtree spans a contiguous sequence segment



Stochastic CFG

A *stochastic context-free grammar* (*SCFG*) is a CFG plus a probability distribution on productions:

$$G = (V, \alpha, S, R, P_p)$$

where $P_p : R \mapsto [0,1]$, and probabilities are normalized at the level of each nonterminal X:

$$\forall \left[\sum_{X \in V} \sum_{X \to \lambda} P_p(X \to \lambda) = 1 \right]$$

The probability of a derivation $S \Rightarrow^* x$ is the product of the probabilities for all its productions: $\prod_i P(X_i \to \lambda_i)$

We can sum over all possible (leftmost) derivations of a given string x to get the probability that G will generate x at random: $P(x \mid G) = \sum_j P(S \Rightarrow_j^* x \mid G)$.

A Simple Example

As an example, consider $\mathcal{G}=(V_{\mathcal{G}}, \alpha, S, R_{\mathcal{G}}, P_{\mathcal{G}})$, for $V_{\mathcal{G}}=\{S, L, N\}$, $\alpha=\{\mathtt{a},\mathtt{c},\mathtt{g},\mathtt{t}\}$, and $R_{\mathcal{G}}$ the set consisting of:

$$S \rightarrow \mathtt{a}\,S\,\mathtt{t} \mid \mathtt{t}\,S\,\mathtt{a} \mid \mathtt{c}\,S\,\mathtt{g} \mid \mathtt{g}\,S\,\mathtt{c} \mid L \qquad (P=0.2)$$

$$L \rightarrow NNNN \qquad (P=1.0)$$

$$N \rightarrow \mathtt{a} \mid \mathtt{c} \mid \mathtt{g} \mid \mathtt{t} \qquad (P=0.25)$$

The probability of the sequence $\mathtt{acgtacgtacgt}$ is given by:

$$P(\mathtt{acgtacgtacgt}) =$$

$$P(\, S \Rightarrow \mathtt{a}S\mathtt{t} \Rightarrow \mathtt{ac}S\mathtt{gt} \Rightarrow \mathtt{acg}S\mathtt{cgt} \Rightarrow \mathtt{acgt}S\mathtt{acgt} \Rightarrow \mathtt{acgt}L\mathtt{acgt} \Rightarrow \mathtt{acgt}NNNN\mathtt{acgt} \Rightarrow \mathtt{acgta}NNN\mathtt{acgt} \Rightarrow \mathtt{acgtac}NN\mathtt{acgt} \Rightarrow \mathtt{acgtacg}N\mathtt{acgt} \Rightarrow \mathtt{acgtacgtacgt}) =$$

$$0.2 \times 0.2 \times 0.2 \times 0.2 \times 0.2 \times 1 \times 0.25 \times 0.25 \times 0.25 \times 0.25 = 1.25\times10^{-6}$$

because this sequence has only one possible (leftmost) derivation under grammar $\mathcal{G}$.

Chomsky Normal Form (CNF)

Any CFG which does not derive the empty string (i.e., $\varepsilon \notin L(G)$) can be converted into an equivalent grammar in *Chomsky Normal Form* (*CNF*). A CNF grammar is one in which all productions are of the form:

$$X \rightarrow YZ \quad \text{or} \quad X \rightarrow \mathtt{a}$$

for nonterminals X, Y, Z, and terminal $\mathtt{a}$.

Transforming a CFG into CNF can be accomplished by appropriately-ordered application of the following operations (ex.):

- Eliminating *useless symbols* (nonterminals that only derive ε)
- Eliminating *null productions* ($X \rightarrow \varepsilon$)
- Eliminating *unit productions* ($X \rightarrow Y$)
- Factoring long rhs expressions ($A \rightarrow \mathtt{abc}$ factored into $A \rightarrow \mathtt{a}B$, $B \rightarrow \mathtt{b}C$, $C \rightarrow \mathtt{c}$)
- Factoring terminals ($A \rightarrow \mathtt{c}B$ is factored into $A \rightarrow CB$, $C \rightarrow \mathtt{c}$)

CNF - Example

Non-CNF:

$$S \rightarrow \mathtt{a}\ S\ \mathtt{t} \mid \mathtt{t}\ S\ \mathtt{a} \mid \mathtt{c}\ S\ \mathtt{g} \mid \mathtt{g}\ S\ \mathtt{c} \mid L$$
$$L \rightarrow NNNN$$
$$N \rightarrow \mathtt{a} \mid \mathtt{c} \mid \mathtt{g} \mid \mathtt{t}$$

CNF:

$$S \rightarrow A\,S_T \mid T\,S_A \mid C\,S_G \mid G\,S_C \mid N\,L_1$$
$$S_A \rightarrow S\,A$$
$$S_T \rightarrow S\,T$$
$$S_C \rightarrow S\,C$$
$$S_G \rightarrow S\,G$$
$$L_1 \rightarrow N\,L_2$$
$$L_2 \rightarrow NN$$
$$N \rightarrow \mathtt{a} \mid \mathtt{c} \mid \mathtt{g} \mid \mathtt{t}$$
$$A \rightarrow \mathtt{a}$$
$$C \rightarrow \mathtt{c}$$
$$G \rightarrow \mathtt{g}$$
$$T \rightarrow \mathtt{t}$$

Disadvantages of CNF: (1) more nonterminals & productions (up to quadratic blowup in size), (2) less obvious relation to problem domain

Advantage: easy implementation of parsers

The Parsing Problem

Two questions for a CFG:

1) Can a grammar *G* derive string *x*?

2) If so, what series of productions would be used during the derivation? *(there may be multiple answers!)*

Additional questions for an SCFG:

1) What is the *probability* that *G* derives string *x*?

2) What is the *most probable* derivation of *x* via *G*?

The CYK Parsing Algorithm

Given a grammar $G = (V, \alpha, S, R)$ in CNF, we build a DP matrix D s.t. $D_{i,j}$ holds the set of nonterminals that could derive the subsequence $x_i \ldots x_j$

Initialization: $\forall_{1 \le i \le n} D_{i,i} = \{A \mid A \rightarrow x_i \in R\}$

The remainder of the DP matrix is then computed left-to-right, top-to-bottom:

$D_{i,j} = \{A \mid A \rightarrow BC \in R, \text{ for some } B \in D_{i,k} \text{ and } C \in D_{k+1,j},\ i \le k < j\}$

Termination: $S \Rightarrow^* x$ iff $S \in D_{1,n}$.

(Cocke and Schwartz, 1970; Younger, 1967; Kasami, 1965)

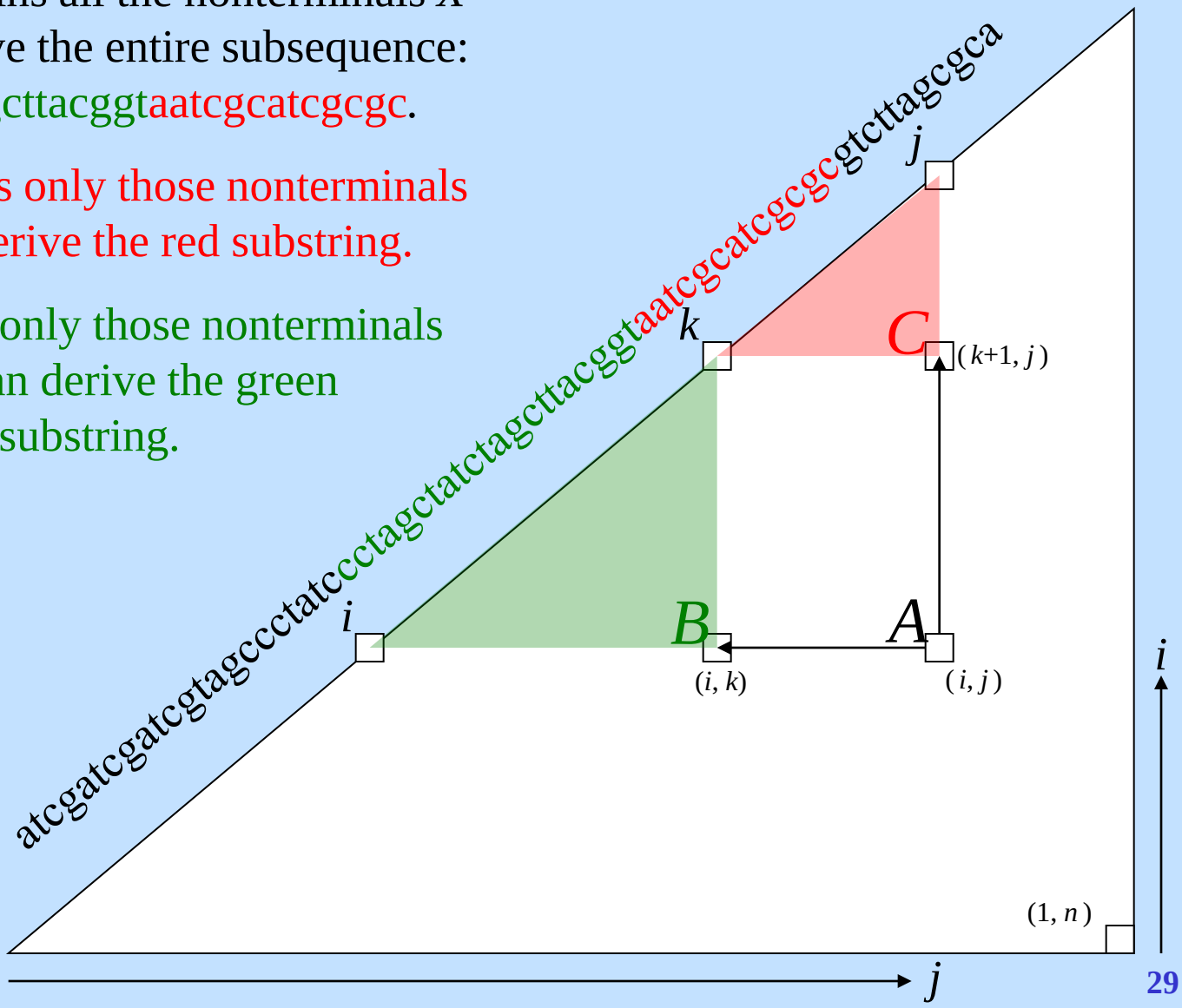
The CYK Parsing Algorithm

Cell (i, j) contains all the nonterminals X which can derive the entire subsequence: actagctatctagcttacggtaatcgcatcgcgc.

$(k+1, j)$ contains only those nonterminals which can derive the red substring.

(i, k) contains only those nonterminals which can derive the green substring.

$A \rightarrow BC$



CYK for SCFG

D(i,j,v) – (log) probability of optimal parse tree with root v of $x_i \ldots x_j$

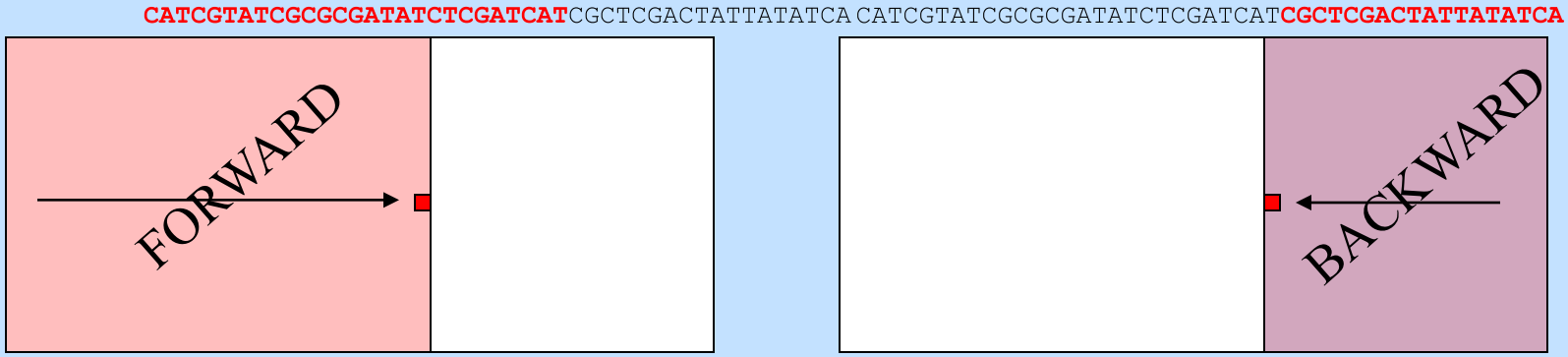
Initialization: $\forall_{1 \le i \le n, v}\ D(i,i,v) = \log P(v \rightarrow x_i)$

Iteration: $D(i,j,v) = \max_{\{y,z,k\}} [D(i,k,y) + D(k+1,j,z) + \log P(v \rightarrow yz)]$

Termination: $\log P(x,\Pi^*) = D(1,L,S)$

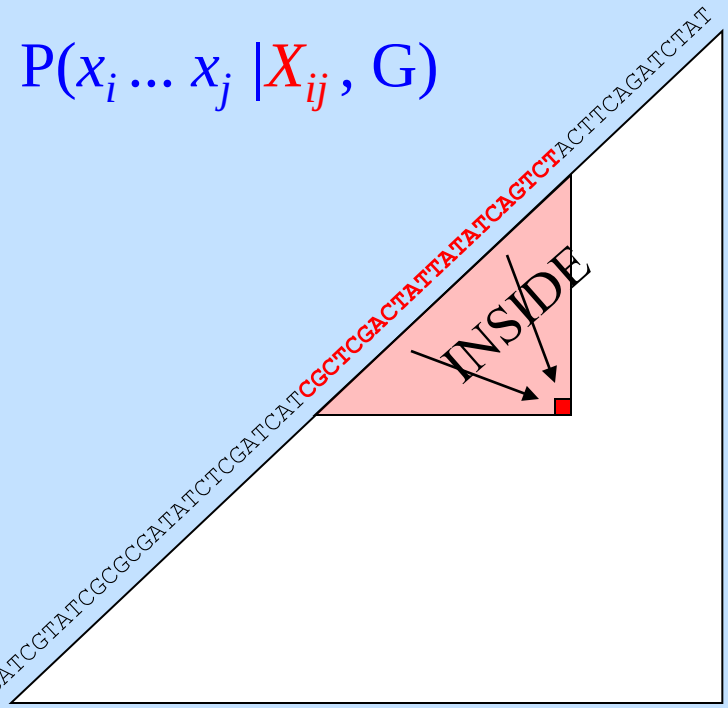
Complexity: $O(L^3M^3)$ time; $O(L^2M)$ memory (L-length; M-#non-terminals)
[compare to HMM: $O(LM^2)$ time, $O(LM)$ memory]

Inside-outside vs. forward-backward

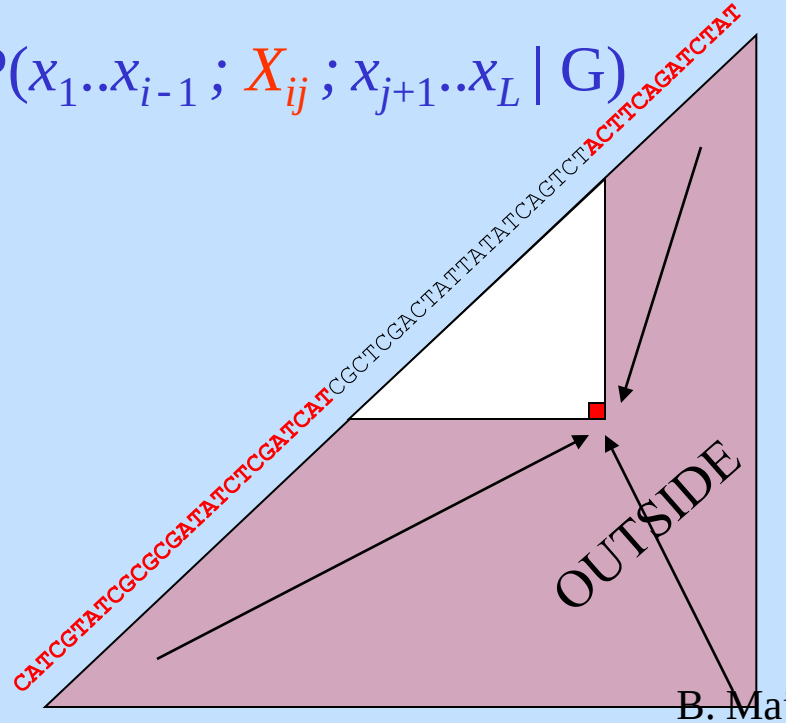


Inside-Outside is the equivalent for trees

$P(x_i \ldots x_j \mid X_{ij}, G)$



$P(x_1..x_{i-1}; X_{ij}; x_{j+1}..x_L \mid G)$



The Inside Algorithm

The "equivalent" of the forward alg. for evaluating the probability of a parse subtree rooted at nonterminal X for the subsequence $x_i \ldots x_j$

$$\alpha(i, j, X) = P(X \Rightarrow^* x_i \ldots x_j) = P(x_i \ldots x_j \mid X_{ij}, G)$$

Initialization:

$\alpha(i,i,X) = P(X \rightarrow x_i)$

Recursion:

$$\alpha(i,j,X) = \sum_{Y,Z} \sum_{k=i..j-1} P(X \rightarrow YZ)\,\alpha(i,k,Y)\,\alpha(k+1,j,Z);$$

The probability $P(x|G)$ of the full input sequence x of length L can then be found in the final cell of the matrix: $\alpha(1, L, S)$.

Complexity: $time = O(L^3 M^3)$ $memory = O(L^2 M)$

Training an SCFG

Two common methods for training an SCFG:

1) If parses are known for the training sequences, we can simply count the number of times each production occurs in the training parses and normalize these counts into probabilities (analogous to HMMs).

2) If parses are NOT known for the training sequences, we can use an EM algorithm similar to the *Forward-Backward* ("Baum-Welch") algorithm for HMMs. The EM algorithm for SCFGs is called *Inside-Outside*.

Covariance models (Eddy & Durbin 1994)

- A general modeling scheme for RNA families
- Based on “profile” SCFG:
  - Model base pairs and single-stranded positions in an RNA secondary structure consensus
  - Allow for insertions and deletions w.r.t. the consensus

The basic model

State type	Description	Production	Emission	Transition
P	(pair emitting)	$P \rightarrow aYb$	$e_v(a,b)$	$t_v(Y)$
L	(left emitting)	$L \rightarrow aY$	$e_v(a)$	$t_v(Y)$
R	(right emitting)	$R \rightarrow Ya$	$e_v(a)$	$t_v(Y)$
B	(bifurcation)	$B \rightarrow SS$	1	1
S	(start)	$S \rightarrow Y$	1	$t_v(Y)$
E	(end)	$E \rightarrow \varepsilon$	1	1

A Toy RNA Family

input multiple alignment:

```
[structure]  . x x > > > x x x x < x < < x > > x > . x x x . < < < .
      human  . A A G A C U U C G G A U C U G G C G . A C A . C C C .
      mouse  a U A C A C U U C G G A U G - C A C C . A A A . G U G a
        orc  . A G G U C U U C - G C A C G G G C A g C C A c U U C .
             1       5         10        15        20        25    28
```

example structure:

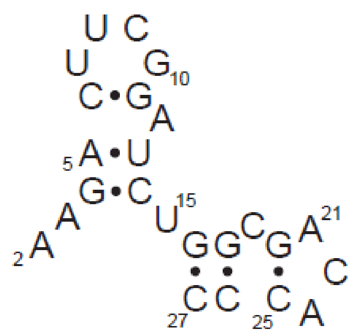
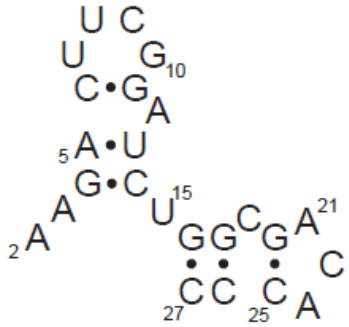


Figure 1

An example RNA sequence family. Top: a toy multiple alignment of three sequences, with 28 total columns, 24 of which will be modeled as consensus positions. The [structure] line annotates the consensus secondary structure: > and < symbols mark base pairs, x's mark consensus single stranded positions, and .'s mark "insert" columns that will not be considered part of the consensus model. Bottom: the secondary structure of the "human" sequence.

Parsing according to structure

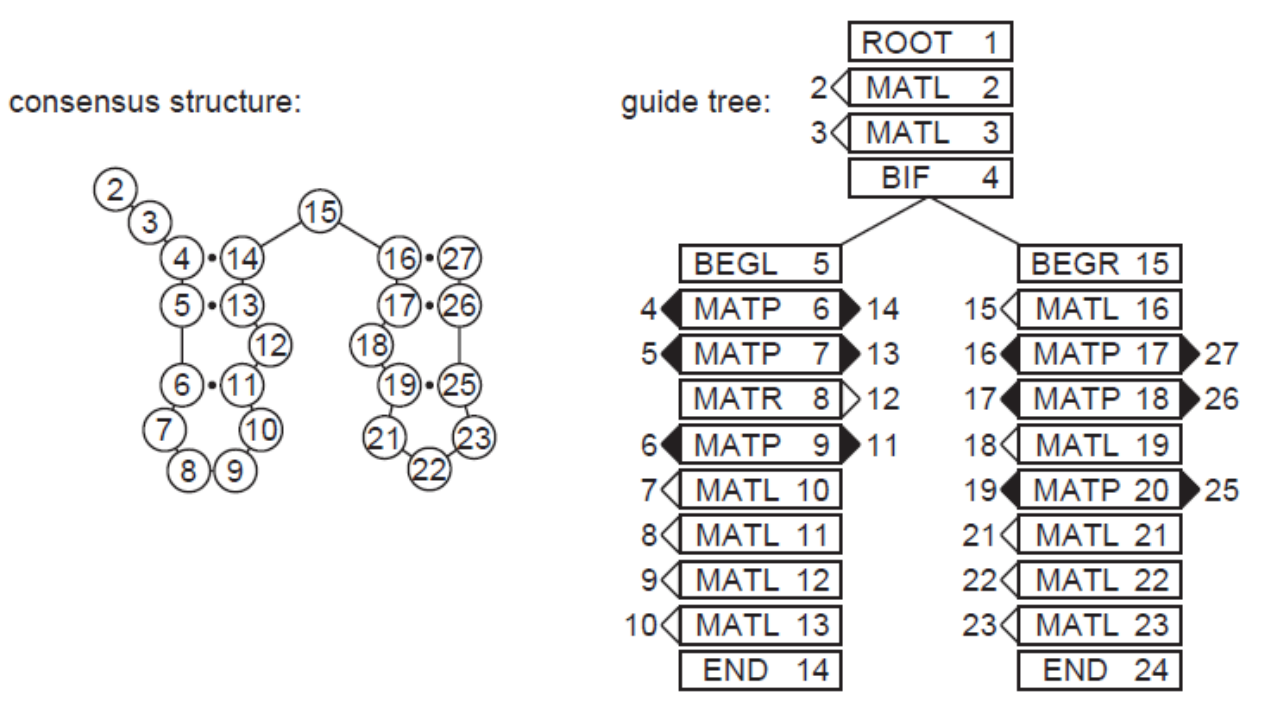
example structure:



			Stem 1			Stem 2		
S_1	$\rightarrow$	$L_2\ldots$	S_5	$\rightarrow$	P_6	S_{15}	$\rightarrow$	L_{16}
L_2	$\rightarrow$	$aL_3\ldots$	P_6	$\rightarrow$	$gP_7c\ldots$	L_{16}	$\rightarrow$	$uP_{17}\ldots$
L_3	$\rightarrow$	$aB_4\ldots$	P_7	$\rightarrow$	$aR_8u\ldots$	P_{17}	$\rightarrow$	$gP_{18}c\ldots$
B_4	$\rightarrow$	S_5S_{15}	R_8	$\rightarrow$	$P_9a\ldots$	P_{18}	$\rightarrow$	$gL_{19}c\ldots$
			P_9	$\rightarrow$	$cL_{10}g\ldots$	L_{19}	$\rightarrow$	$cP_{20}\ldots$
			L_{10}	$\rightarrow$	$uL_{11}\ldots$	P_{20}	$\rightarrow$	$gL_{21}c\ldots$
			L_{11}	$\rightarrow$	$uL_{12}\ldots$	L_{21}	$\rightarrow$	$aL_{22}\ldots$
			L_{12}	$\rightarrow$	$cL_{13}\ldots$	L_{22}	$\rightarrow$	$cL_{23}\ldots$
			L_{13}	$\rightarrow$	$gE_{14}\ldots$	L_{23}	$\rightarrow$	$aE_{24}\ldots$
			E_{14}	$\rightarrow$	ϵ	E_{24}	$\rightarrow$	ϵ

- The positional nonterminals are connected by a tree!

From consensus structure to a guide tree model

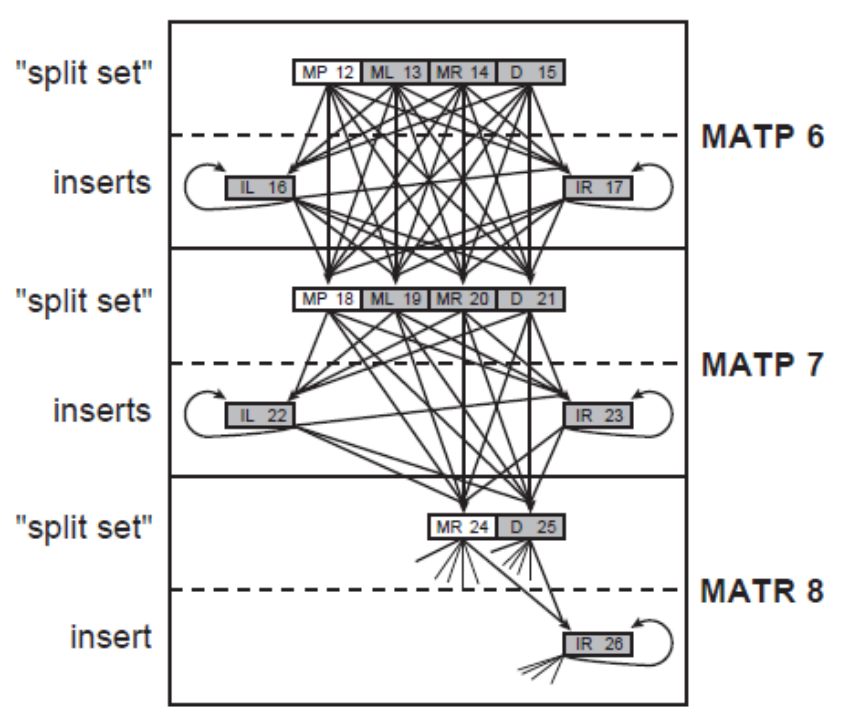


Node	Description
MATP	(pair)
MATL	(single strand, left)
MATR	(single strand, right)
BIF	(bifurcation)
ROOT	(root)
BEGL	(begin, left)
BEGR	(begin, right)
END	(end)

Prefer MATL over MATR

- Non-terminal → state.
- State transition prob. = 1
- Emission probs per state: MATP (16), MATL/R (4)

From guide tree to CM

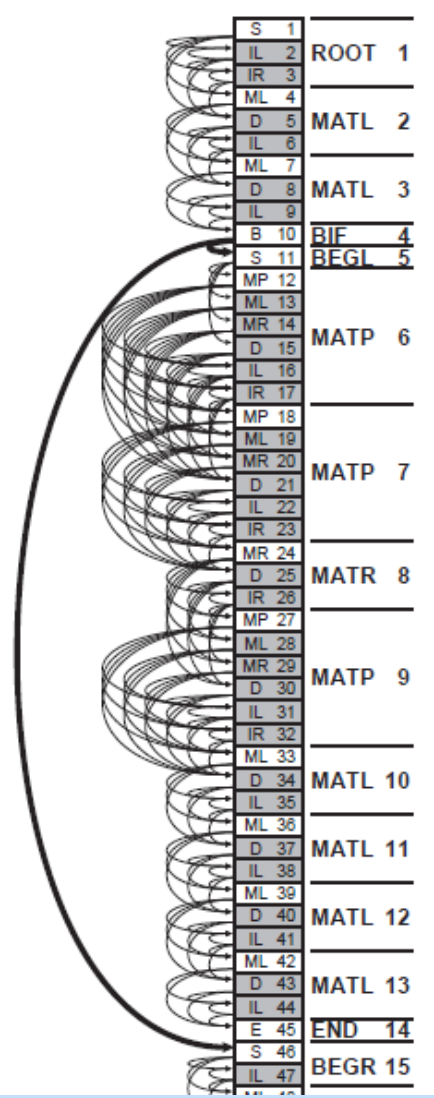


Node	States
MATP	[MP ML MR D] IL IR
MATL	[ML D] IL
MATR	[MR D] IR
BIF	[B]
ROOT	[S] IL IR
BEGL	[S]
BEGR	[S] IL
END	[E]

- Each tree node is expanded into several states:
  - Split set – the main consensus state; one of which must be visited
  - Insert set – visited 0 or more times.

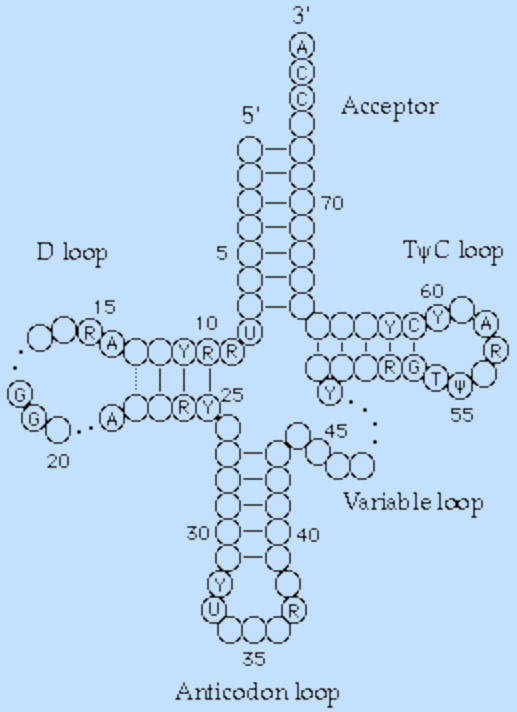
The final CM

1. Start from consensus structure, or use a variant of Nussinov-Jacobson on some multiple alignment if (*) no such structure is known.
2. Learn parameters using EM.
3. If (*) use the current model to create a new multiple alignment (via CYK) and feed to (1) till convergence.



Application to tRNA modeling

- The largest gene family in most genomes.
- Ideal test case – primary sequence varies while structure is conserved.
- Started from an alignment of 1415 tRNAs; avg. 40% sequence identity;100 held out for testing.



Performance Evaluation I

- Two training modes: (A) start from known alignment; (U) start from unaligned sequences
- Bit log-odds score averaged over test set
- Alignment accuracy – % truly aligned symbol pairs that are also aligned in the inferred alignment

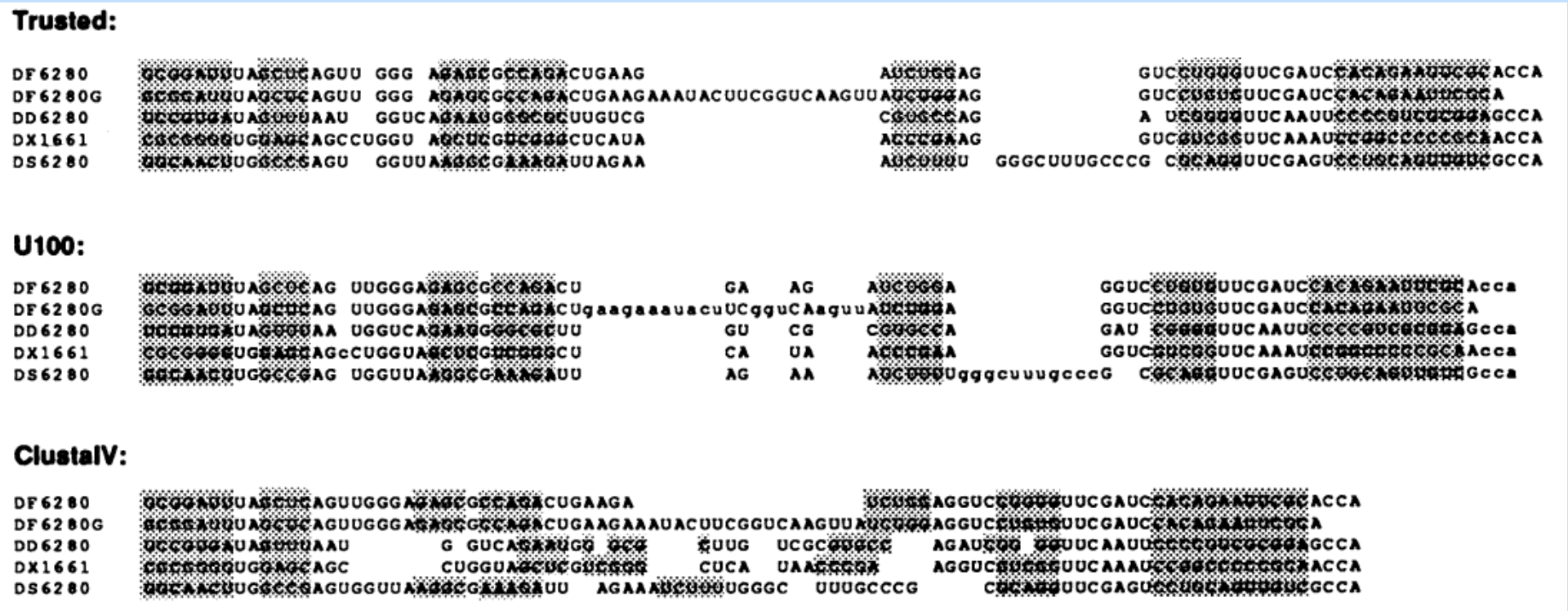
Mode	#Iterations	Bit score	Accuracy
A100	3	57.3	94%
U100	23	56.7	90%

30 for HMM

30% for degapped alignment

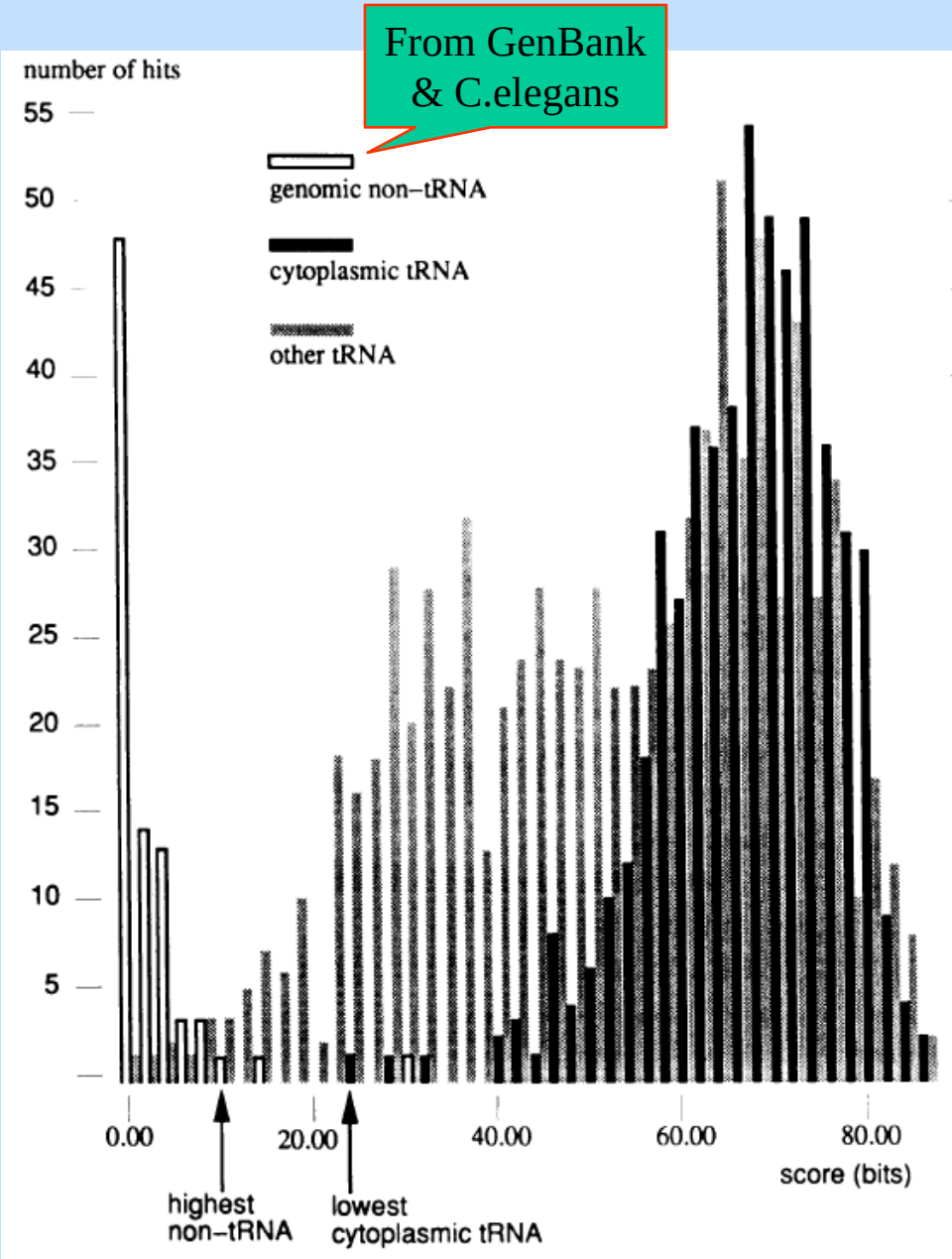
Multiple sequence alignment

- Example of yeast tRNAs whose 3D structure is known from crystallography.



Performance Evaluation II

- Tested A1415 in searching 6Mb sequence from GenBank structural RNA db and C. elegans genome.
- Perfect separation for a wide threshold range of the more conserved c-tRNAs
- In C. elegans all 14 tRNAs were detected (score>31), no false positives
- In GenBank, 26/522 (5%) annotated tRNAs were missed; 22/26 lack the D-stem loop



HOME | SEARCH | BROWSE | FTP | BLOG | HELP

Rfam 9.1 (January 2009, 1372 families)

The Rfam database is a collection of RNA families, each represented by **multiple sequence alignments, consensus secondary structures** and **covariance models (CMs)**. **More...**

QUICK LINKS	YOU CAN FIND DATA IN RFAM IN VARIOUS WAYS...
SEQUENCE SEARCH	Analyze your RNA sequence for Rfam matches
VIEW AN RFAM FAMILY	View Rfam family annotation and alignments
KEYWORD SEARCH	Query Rfam by keywords
TAXONOMY SEARCH	Fetch families or sequences by NCBI taxonomy
JUMP TO	enter any accession or ID [Go] [Example]

Enter any type of accession or ID to jump to the page for a Rfam family, sequence or genome

Or view the help pages for more information