CS681: Advanced Topics in Computational Biology

Week 10 Lecture 1

Can Alkan
EA224
calkan@cs.bilkent.edu.tr

http://www.cs.bilkent.edu.tr/~calkan/teaching/cs681/

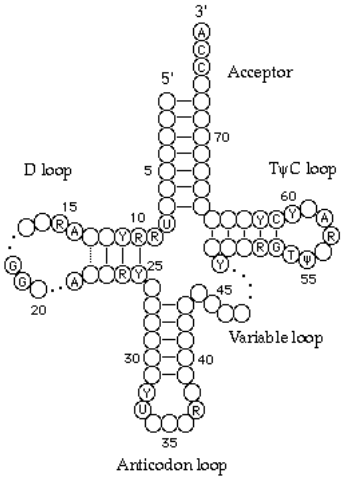
RNA folding

- Prediction of secondary structure of an RNA given its sequence
- General problem is NP-hard due to “difficult” substructures, like pseudoknots
- Most existing algorithms require too much memory ($\geq O(n^2)$), and run time ($\geq O(n^3)$) thus limited to smaller RNA sequences

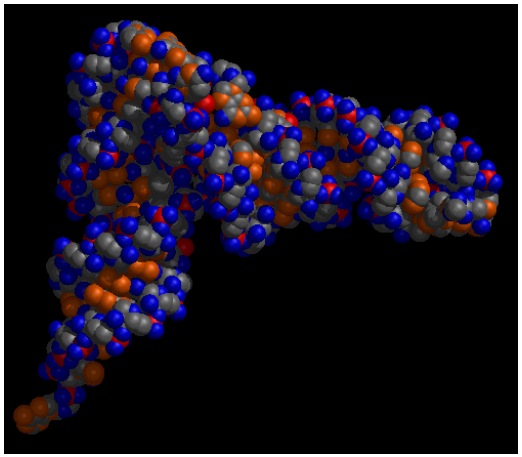
RNA Structural Levels

AAUCG...CUUCUUCCA

Primary

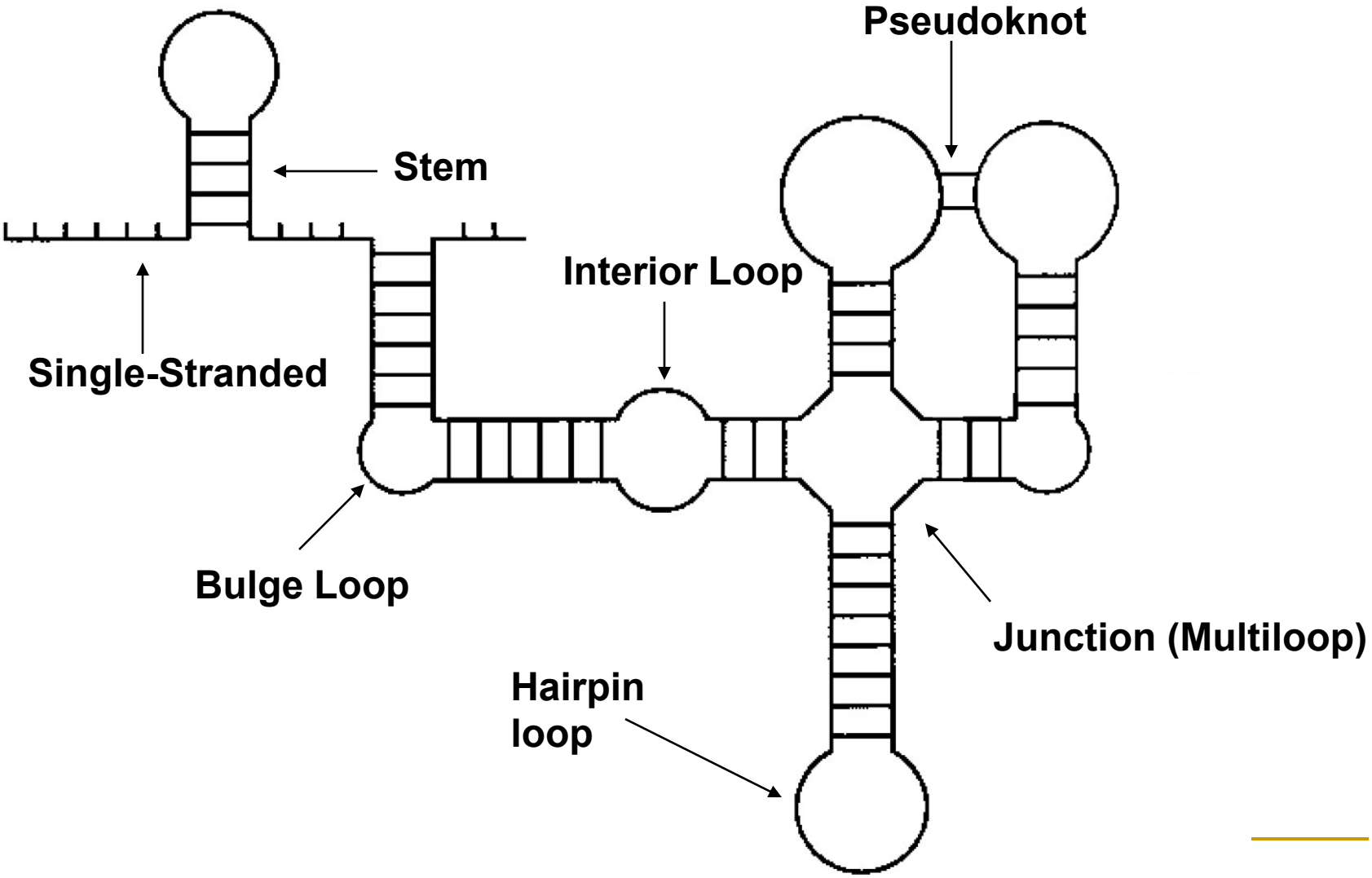


Secondary



Tertiary

RNA Secondary Structure



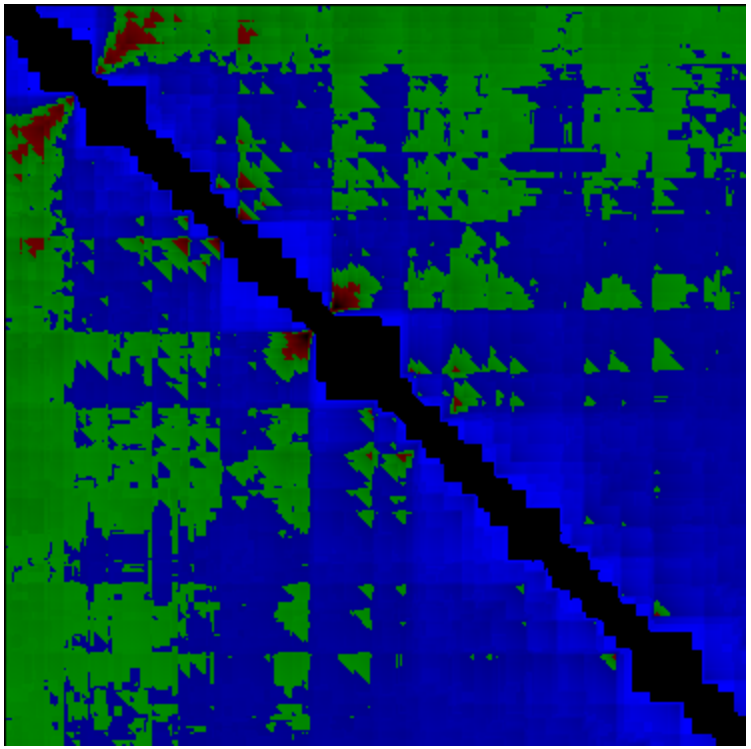
Predicting RNA secondary structure

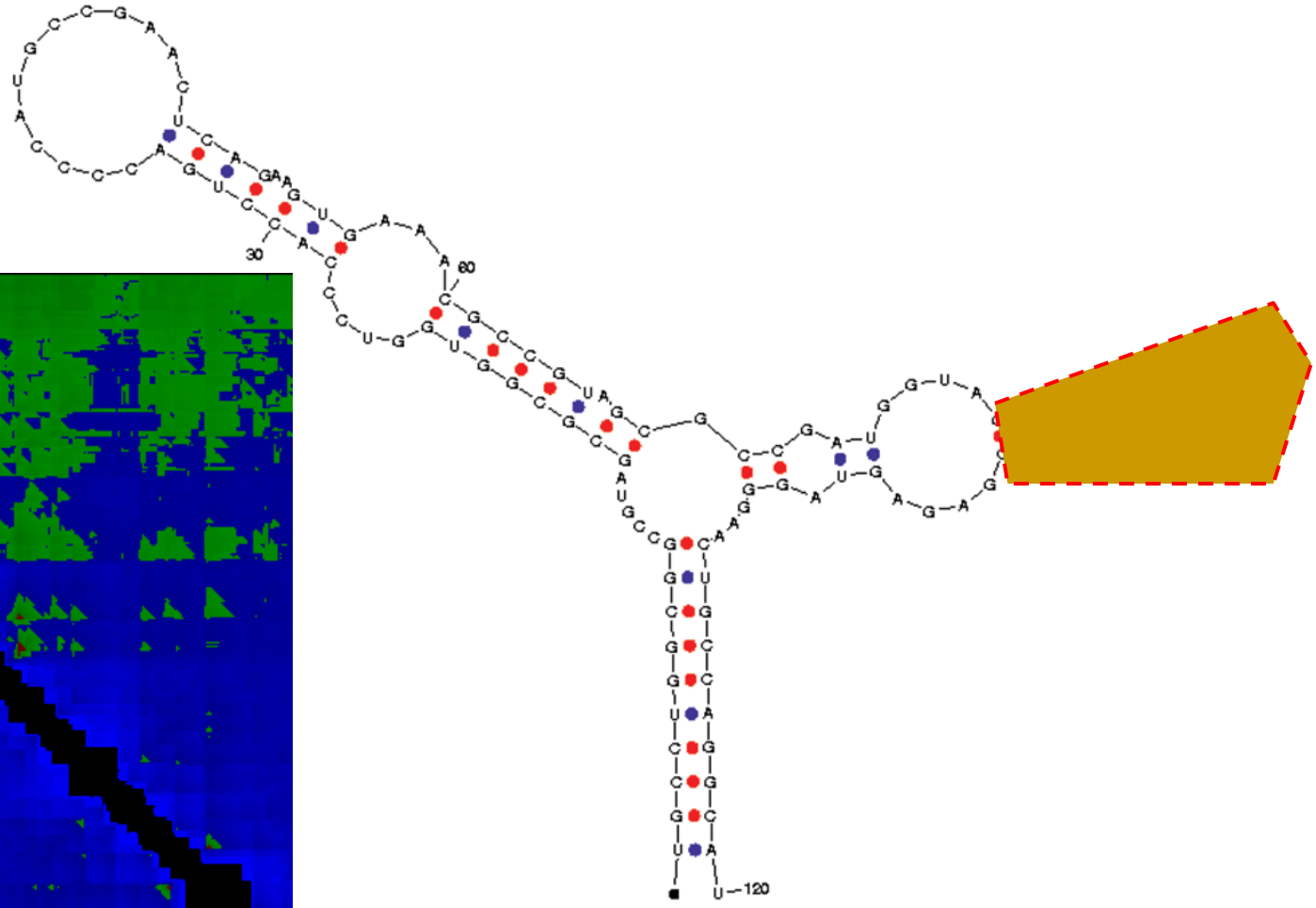
- Base pair maximization
- Minimum free energy (most common)
  - Fold, Mfold (Zuker & Stiegler)
  - RNAfold (Hofacker)
- Multiple sequence alignment
  - Use known structure of RNA with similar sequence
- Covariance
- Stochastic Context-Free Grammars

Alkan, Karakoç et al, RECOMB 2006

DENSITYFOLD

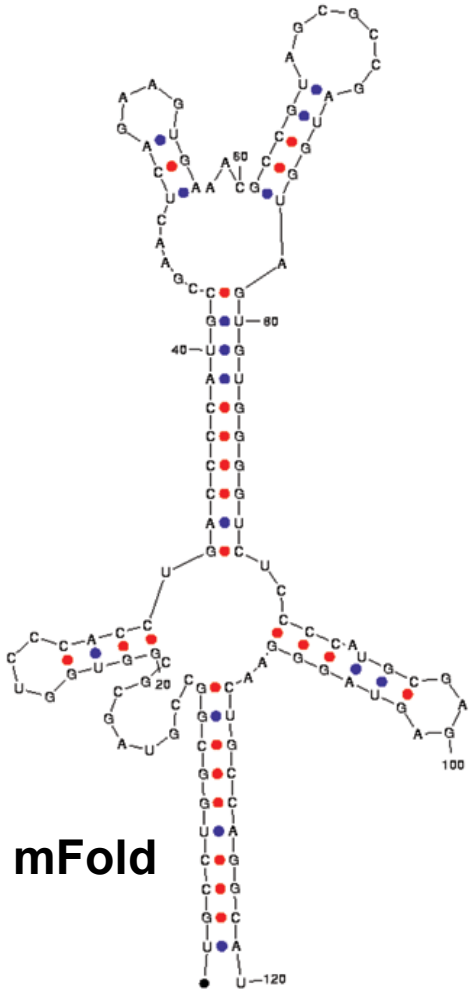
E.coli 5S rRNA



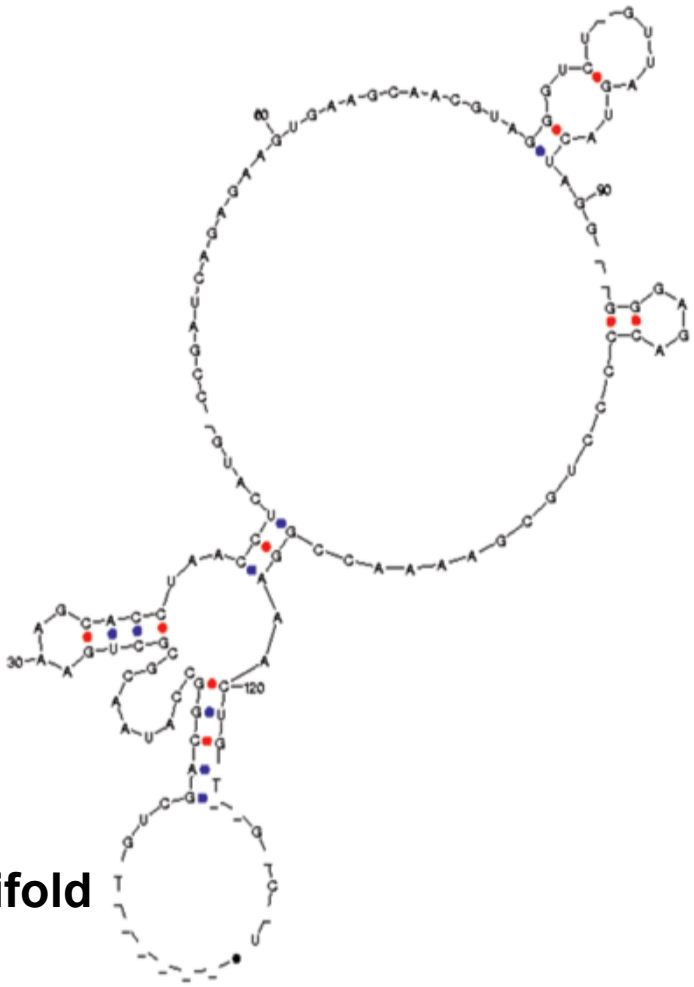


Energy Density Landscape

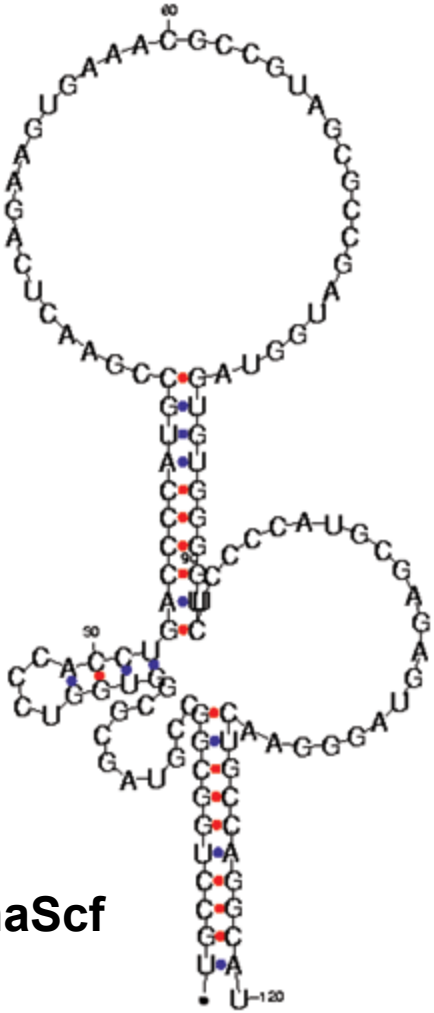
E.coli 5S rRNA predictions



mFold



RNAalifold



rnaScf

Densityfold (alteRNA)

- Instead of finding minimum global free energy, find local minimum free energies
- Emulate the folding process of RNA folding by aiming to keep locally stable substructures
- *Energy density seen by a basepair:* the free energy of the "optimal substructure" normalized by distance
- *Energy density of an unpaired base:* energy density of the nearest encapsulating basepair
- Densityfold optimizes a linear combination of free energy and total energy density
- For every potential basepair, compute the optimal contribution of the implied substructure
- The optimization function is non linear

  Hill climbing process for approximating the contributions of unpaired bases

Densityfold energy types

- *eH(i,j,)*: free energy of a hairpin loop enclosed by the base pair S[i].S[j]
- *eS(i,j,)*: free energy of the base pair S[i].S[j] provided that it forms a stacking pair with S[i+1].S[j-1]
- *eBI(i,j,i',j')*: free energy of an internal loop or a bulge that starts with S[i].S[j] and ends with S[i'].S[j']

Densityfold energy types

- *eM(i,j,i_1,j_1,…,i_k,j_k)*: free energy of multibranch loop that starts with S[i].S[j] and branches out S[i_1].S[j_1], S[i_2].S[j_2], …, S[i_k].S[j_k]
- *eDA(j,j-1)*: free energy of an unpaired dangling base S[j] when S[j-1] forms a base pair with another base

Densityfold energy tables

- *ED(j)*: minimum total free energy density of a secondary structure for substring S[1, j].
- *E(j)*: free energy of the energy density minimized secondary structure for substring S[1, j].
- $ED_S(i, j)$: minimum total free energy density of a secondary structure for S[i, j], provided that S[i].S[j] is a base pair.
- $E_S(i, j)$: free energy of the energy density minimized secondary structure for the substring S[i, j], provided that S[i].S[j] is a base pair.

Densityfold energy tables

- ED_{BI} *(i, j)*: minimum total free energy density of a secondary structure for S[i, j], provided that there is a bulge or an internal loop starting with base pair S[i].S[j].
- E_{BI} *(i, j)*: free energy of an energy density minimized structure for S[i, j], provided that a bulge or an internal loop starting with base pair S[i].S[j].
- ED_M*(i, j)*: minimum total free energy density of a secondary structure for S[i, j], such that there is a multibranch loop starting with base pair S[i].S[j].
- E_M*(i, j)*: free energy of an energy density minimized structure for S[i, j], provided there is a multibranch loop starting with base pair S[i].S[j].

Calculating energy tables

$$ED(j) = \min \left\{ \begin{array}{l} ED(j-1) \\ \min_{1 \le i \le j-1} \{ED(i-1) + ED_S(i,j)\} \end{array} \right\}$$

$$ED_S(i,j) = \min \left\{ \begin{array}{ll} +\infty, & (i) \\ eH(i,j), & (ii) \\ 2\frac{eS(i,j)+E_S(i+1,j-1)}{j-i+1} + ED_S(i+1,j-1), & (iii) \\ ED_{BI}(i,j), & (iv) \\ ED_M(i,j) & (v) \end{array} \right\}$$

- Similar calculations for other tables
- $O(n^{k+2})$ time and $O(n^2)$ space

Linear combination of MFE and ED

For any x ε {S,BI,M} let $ELC_x(i, j) = ED_x(i, j) + E_x(i, j)$.

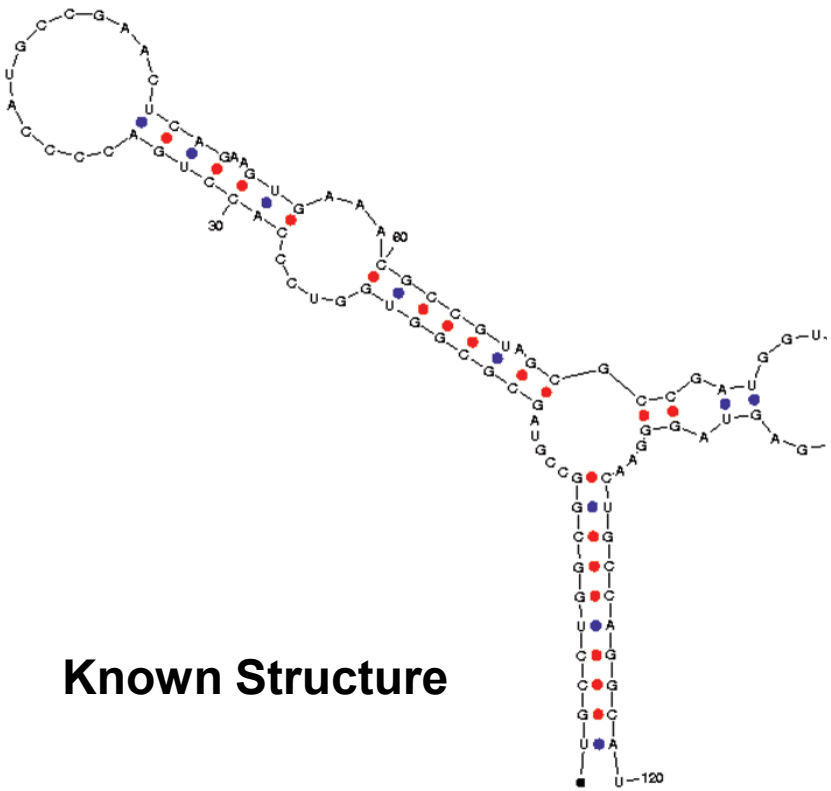
Optimize ELC(n) = ED(n) + E(n).

$$ELC(j) = \min \left\{ \begin{array}{l} ELC(j-1) \\ \min_{1 \le i \le j-1} \{ELC(i-1) + ELC_S(i,j)\} \end{array} \right\}$$

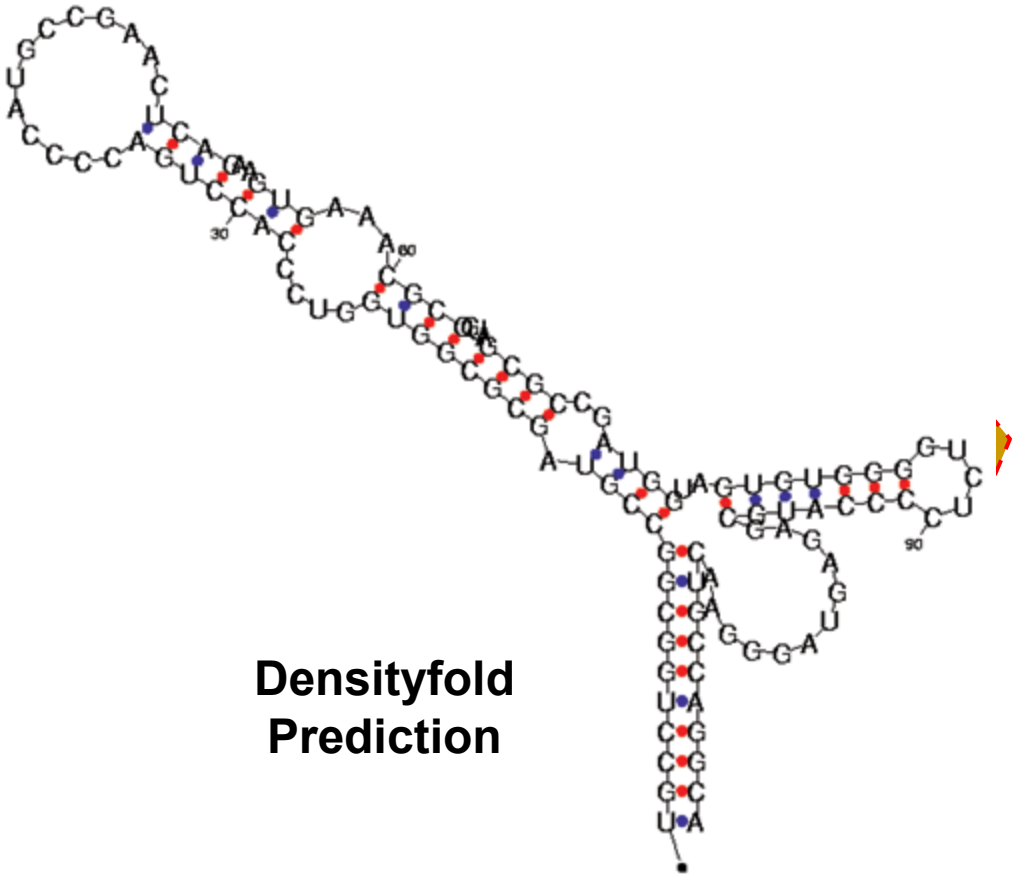
$$ELC_S(i,j) = \min \left\{ \begin{array}{ll} +\infty, & (i) \\ eH(i,j) \cdot (1+\sigma), & (ii) \\ 2\frac{eS(i,j)+E_S(i+1,j-1)}{j-i+1} + ELC_S(i+1,j-1) + \sigma \cdot eS(i,j), & (iii) \\ ELC_{BI}(i,j), & (iv) \\ ELC_M(i,j) & (v) \end{array} \right\}$$

- Similar formulations for ELC_{BI} and ELC_M
- $O(n^4)$ running time

Densityfold prediction: E.coli 5S rRNA



Known Structure



Densityfold Prediction

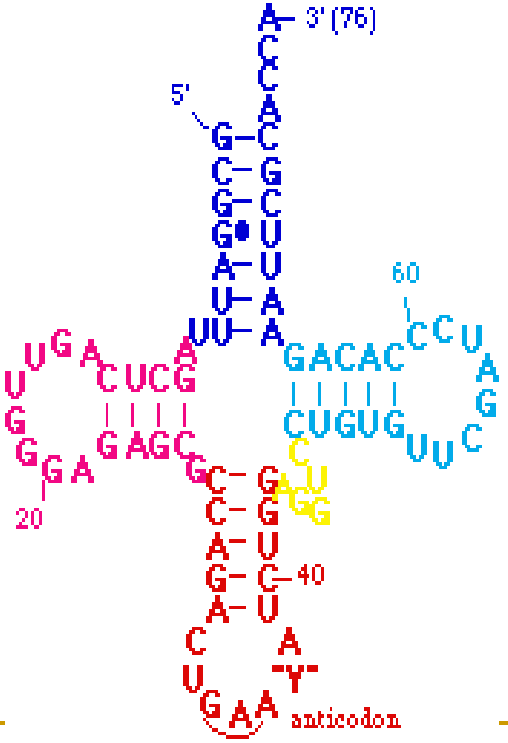
CONTRAFOLD

CONTRAfold

Probabilistic RNA folding algorithm

Problem: Given an RNA sequence, predict the most likely secondary structure

AUCCCCGUAUCGAUC
AAAAUCCAUGGGUAC
CCUAGUGAAAGUGUA
UAUACGUGCUCUGAU
UCUUUACUGAGGAGU
CAGUGAACGAACUGA



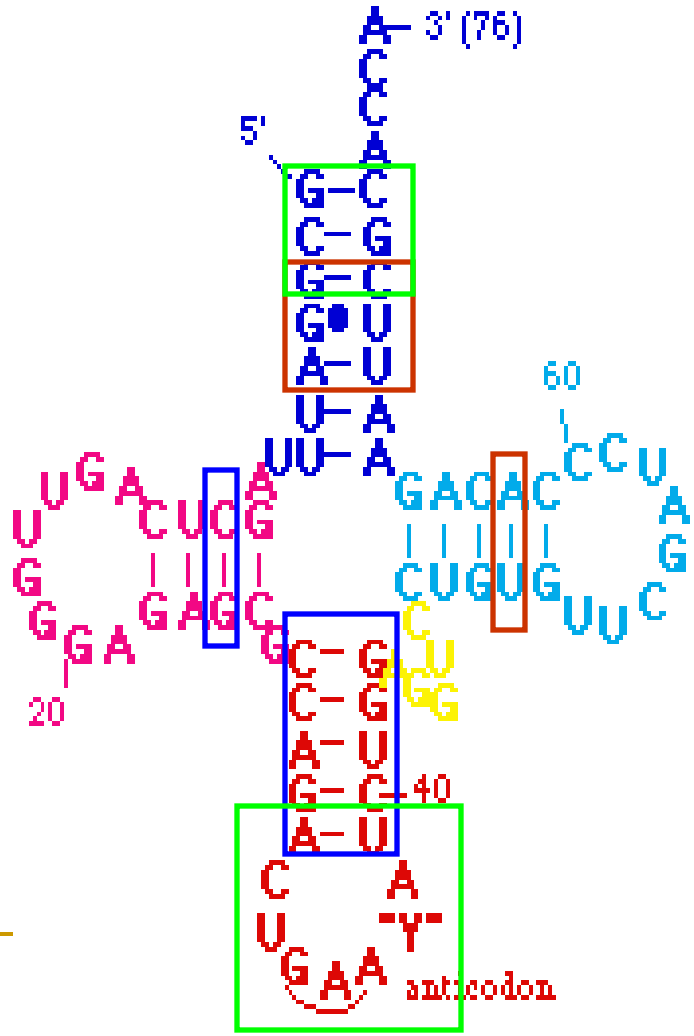
Do et al, Bioinformatics, 2006

CONTRAfold

- CONTRAfold looks at *features* that indicate a good structure

For example:

- C-G base pairings
- A-U base pairings
- Helices of length 5
- Hairpin loops of size 9
- Bulge loops of size 2
- CG/GC Base-pair stacking interactions



Do et al, Bioinformatics, 2006

Choosing a structure

- Every *feature* f_i is associated with a *weight* w_i.
- **The probability of a structure y, given a sequence x, is determined by the following relationship:**

# of occurrences of feature *i*, in structure y generated from sequence x

$$P(\mathbf{y} \mid \mathbf{x}) \propto \exp\left(\sum_i w_i \cdot F_i(\mathbf{x}, \mathbf{y})\right)$$

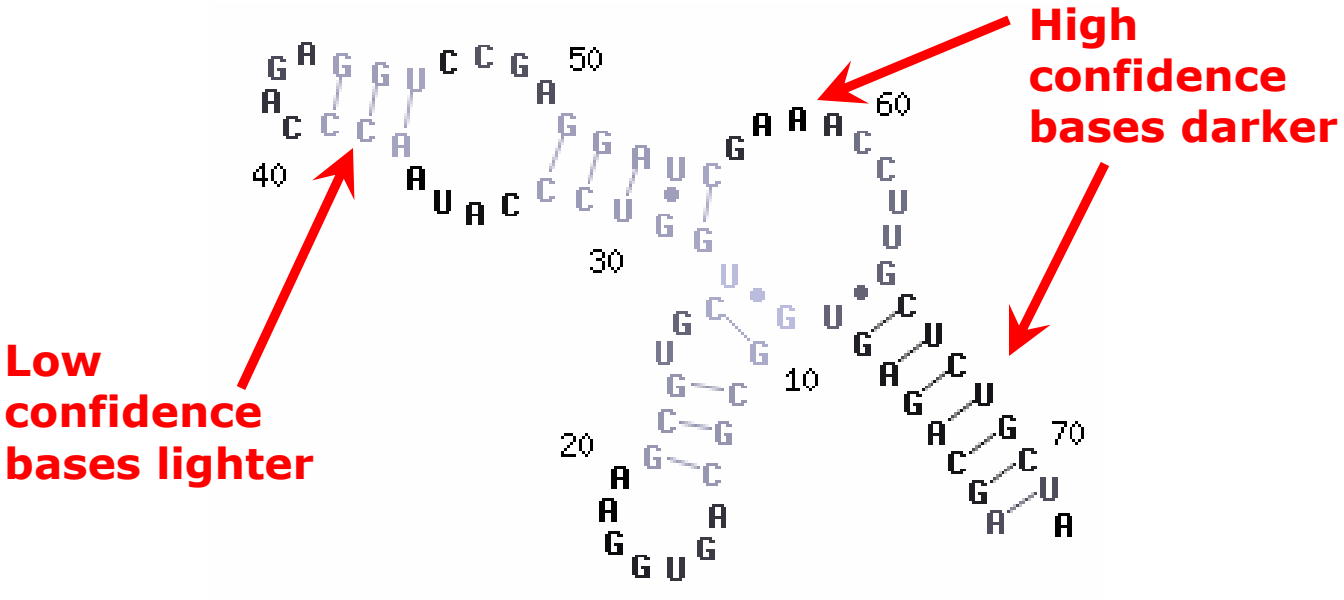
structure **sequence** **weight of Feature *i***

$$\mathbf{F}(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} F_1(\mathbf{x}, \mathbf{y}) \\ F_2(\mathbf{x}, \mathbf{y}) \\ \vdots \end{bmatrix} = \begin{bmatrix} \text{\# of hairpin loops in } \mathbf{y} \text{ of length 4} \\ \text{\# of GC/GC stacking interactions in } \mathbf{y} \\ \vdots \end{bmatrix}.$$

Do et al, Bioinformatics, 2006

Choosing a structure

- **Considers all structures and finds optimal structure via *dynamic programming* in $O(n^3)$**
- **Added bonus: probability associated with each base**



Do et al, Bioinformatics, 2006

Maximum Expected Accuracy

For a candidate structure $\hat{\mathbf{y}}$ with true structure $\mathbf{y}$

$$\hat{\mathbf{y}}_{mea} = \underset{\hat{\mathbf{y}}}{\text{argmax}}\, E_y\,[\text{accuracy}_{\gamma}(\hat{\mathbf{y}}, \mathbf{y})]$$

$$M_{1,L} = \max_y E_y\,[\text{accuracy}_{\gamma}(\hat{\mathbf{y}}_{mea}, \mathbf{y})]$$

$$M_{i,j} = \max \begin{cases} qi & \text{if } i=j \\ qi + Mi+1,j & \text{if } i<j \\ qj + Mi,j-1 & \text{if } i<j \\ \gamma.2pij + Mi+1,j+1 & \text{if } i+2<j \\ M_{i,k}+M_{k+1,j} & \text{if } i \le k<j \end{cases}$$

Do et al, Bioinformatics, 2006

Sensitivity vs Specificity: γ

$$\text{Sensitivity} = \frac{\text{\# correct base pairings}}{\text{\# true base pairings}}$$

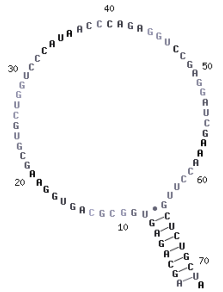
$$\text{Specificity} = \frac{\text{\# correct base pairings}}{\text{\# predicted base pairings}}$$

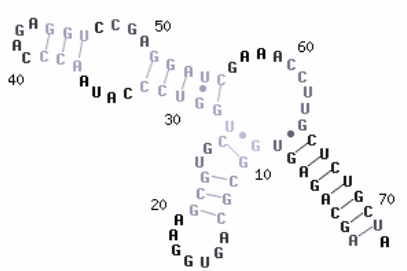
AUCCCCGUAUCGAUC
AAAAUCCAUGGGUAC
CCUAGUGAAAGUGUA
UAUACGUGCUCUGAU
UCUUUACUGAGGAGU
CAGUGAACGAACUGA

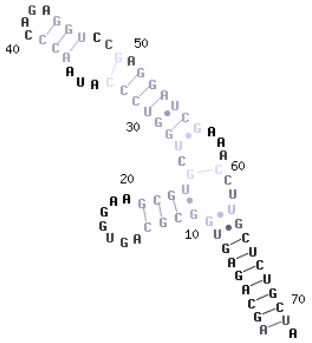
γ = 1

γ = 8

γ = 1024







Do et al, Bioinformatics, 2006

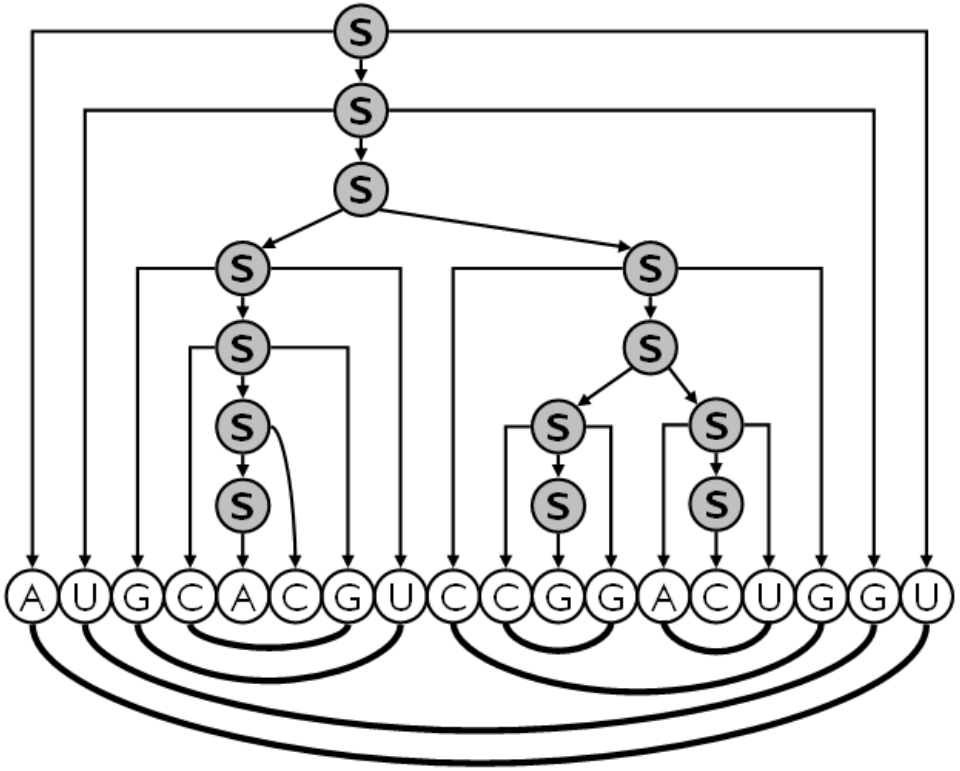
Learning to predict good structures

- CONTRAfold learns the relative value, or *weight,* of each of its *features*
- A *training set* is a collection of known correct solutions that a program learns from.
- CONTRAfold trains on set of published examples of known RNA structures taken from a database called Rfam (RNA families)
- CONTRAfold determines the *weight* for each *feature* that maximizes its performance on the training set.

Do et al, Bioinformatics, 2006

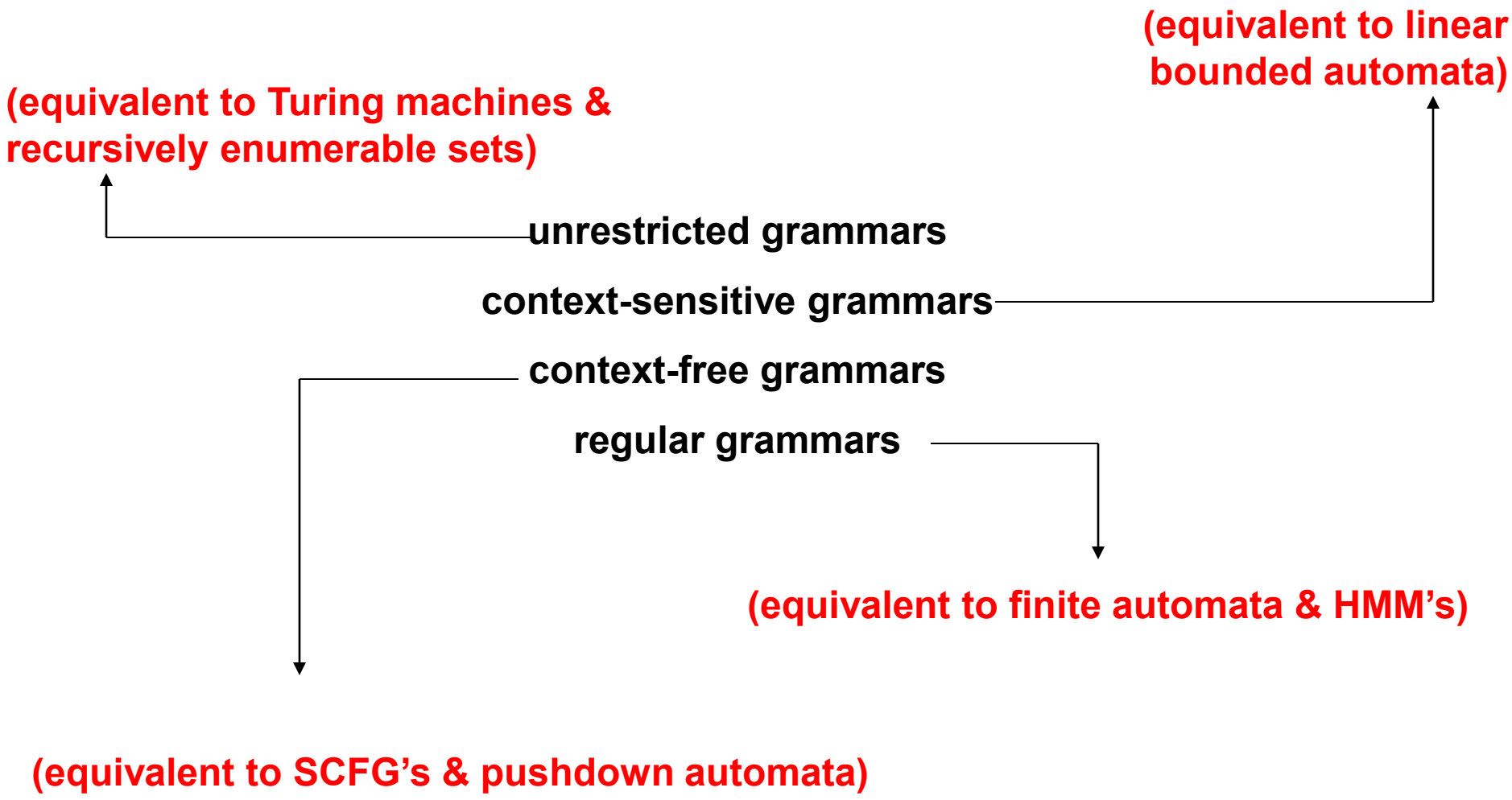
STOCHASTIC CONTEXT-FREE GRAMMARS

SCFG



- RNA folding can be represented as context-free grammars

Chomsky hierarchy



Context-free grammars

A *context-free grammar* is a generative model denoted by a 4-tuple:

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

where:

α is a *terminal alphabet*, *(e.g., {a, c, g, t})*
V is a *nonterminal alphabet*, *(e.g., {A, B, C, D, E, ...})*
$S \in V$ is a special *start symbol*, and
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* $\mathbf{S} \Rightarrow^* \mathbf{x}$ denotes a possible 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'll concentrate of leftmost derivations where the leftmost nonterminal is always replaced first.

Context-free vs. regular

The advantage of CFG's over HMM's 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 a DFA or an HMM is inadequate to enforce the constraint that all left elements must have a matching right element.

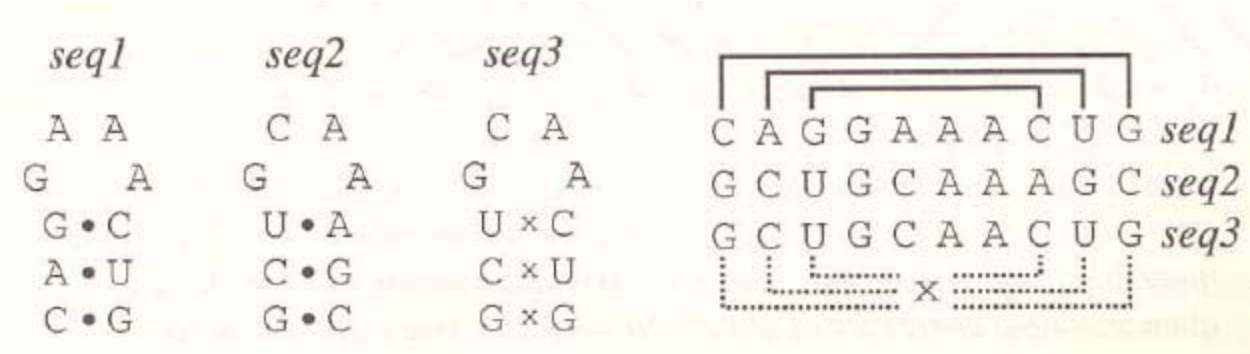
In contrast, in a CFG we can use rules such as $X \rightarrow (X)$. A sample derivation using such a rule is:

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

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

A CFG for an RNA

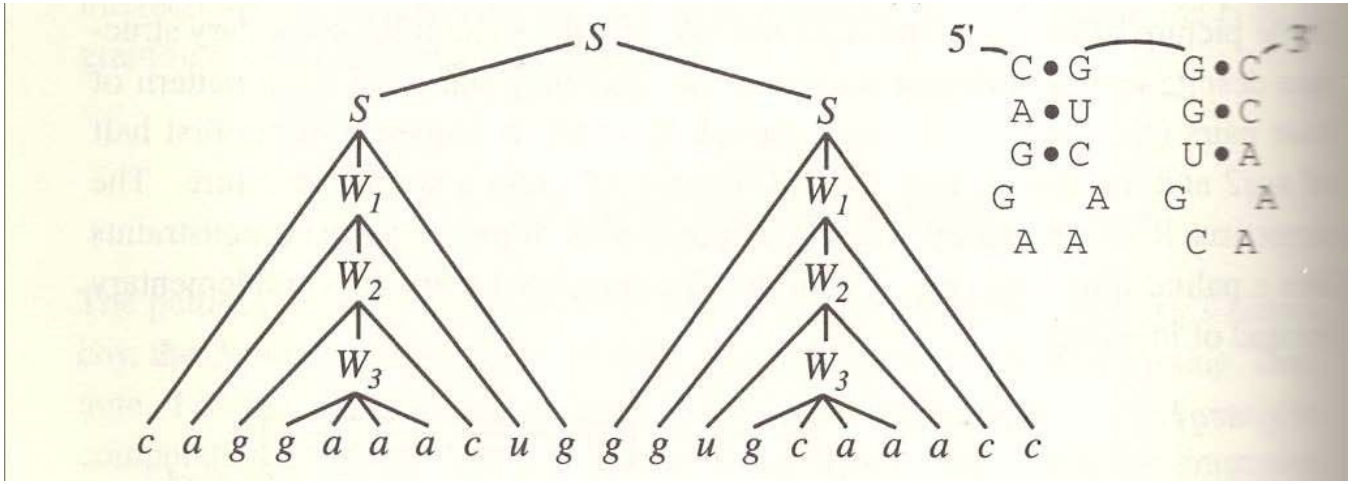
- RNA hairpin with 3 bp stem and a 4-base loop (GAAA or GCAA)



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
- **Root –** start nonterminal S
- **Leaves –** terminal symbols in the given string
- **Internal nodes -** nonterminals
- The children of an internal node are the productions of that nonterminal (left-to-right order



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 $P_p : R$ a ¡, and probabilities are normalized at the level of each l.h.s. symbol X:

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

Thus, we can compute the probability of a single derivation $S \Rightarrow^* x$ by multiplying the probabilities for all productions used in the derivation:

$$\prod_i P(X_i \to \lambda_i)$$

We can sum over all possible (leftmost) derivations of a given string x to get the probability that G will generate x at random:

$$P(x \mid G) = \sum_j P(S \Rightarrow_j^* x \mid G).$$

An example

As an example, consider G=(V_G, α, S, R_G, P_G), for V_G={S, L, N}, α={a,c,g,t}, and R_G the set consisting of:

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

$$L \rightarrow N\,N\,N\,N \qquad \textbf{(P=1.0)}$$

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

Then the probability of the sequence acgtacgtacgt is given by:

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

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

$$0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad 1 \quad 0.25 \quad 0.25 \quad 0.25 \quad 0.25 = 1.25\ 10^{-6}$$

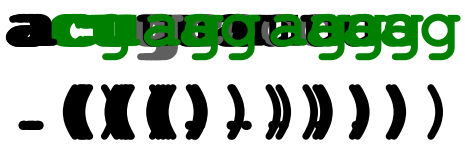
because this sequence has only one possible (leftmost) derivation under grammar G.

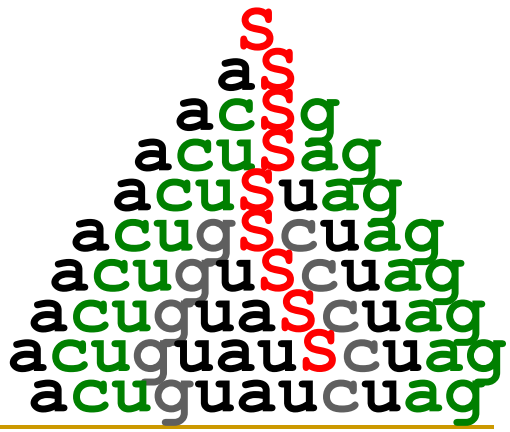
Structure using SFCG

- Grammar rules with associated probabilities

S →	aSu	cSg	aS	uS	…	Su	SS	ε
P	.21	.15	.11	.08		.03	.22	.02

- We select the set of transformations that highest probability of generating the input sequence. This set gives us our structure.
- Let's generate a structure for the sequence acuuauuag





Chomsky Normal Form

A CNF grammar is one in which all productions are of the form:

$X \rightarrow Y\,Z$

or:

$X \rightarrow \mathtt{a}$

Non-CNF:

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

$L \rightarrow N\,N\,N\,N$

$N \rightarrow \mathtt{a} \mid \mathtt{c} \mid \mathtt{g} \mid \mathtt{u}$

CNF:

$S \rightarrow A\,S_T \mid T\,S_A \mid C\,S_G \mid G\,S_C \mid N\,L_1$

$S_A \rightarrow S\,A$

$S_T \rightarrow S\,T$

$S_C \rightarrow S\,C$

$S_G \rightarrow S\,G$

$L_1 \rightarrow N\,L_2$

$L_2 \rightarrow N\,N$

$N \rightarrow \mathtt{a} \mid \mathtt{c} \mid \mathtt{g} \mid \mathtt{u}$

$A \rightarrow \mathtt{a}$

$C \rightarrow \mathtt{c}$

$G \rightarrow \mathtt{g}$

$T \rightarrow \mathtt{u}$

Parsing CFG

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*?

Parsing CFG

- CYK Algorithm (Cocke-Younger-Kasami)
  - Dynamic Programming method
- Modified CYK for SCFG
  - “Inside algorithm”
  - Training similar to HMM
    - If parses are known for training data sequences, simply count the number of times for each production, calculate probabilities (labeled sequence training for HMM)
    - If parses are not known, apply an EM algorithm called “Inside-Outside” (“forward-backward” for HMM)