RNA Folding, Four Russians, and Stochastic Grammars

Ran “RON” Libeskind-Hadas
Harvey Mudd College
hadas@cs.hmc.edu or hadas@mit.edu

Phylogenetic Tree of Mudd

Our name is Mudd

My own prank...

Outline

• Longest common sequence (LCS) problem revisited
  (the Four Russians Method)
• RNA folding
  – Why you should care
  – Nussinov’s Algorithm
  – Four Russians *amazingly* reappear!
  – Zuker Algorithm
  – Stochastic Context-Free Grammars
A few things about today's lecture...

- Algorithms focus!
- Your slides are a proper subset of mine
- Some things will be done on the board

### Computing the LCS

```
0 1 2 3 4 5
ACAT
```

```
AGCAGT
```

What is the LCS here?

```
def LCS(S1, S2):
    if len(S1) == 0 or len(S2) == 0: return 0
    elif S1[-1] == S2[-1]:
        return 1 + LCS(S1[:-1], S2[:-1])
    else:
        option1 = LCS(S1, S2[:-1])
        option2 = LCS(S1[:-1], S2)
        return max(option1, option2)
```

---

String lengths m and n  
"Basic" solution: Time/Space?
"Fancy" solutions: Time/Space?

Can we find the LCS score faster than O(mn)?
• For the sake of simplicity, but wlog, assume $m = n$ (square DP table)

Running time: $O(nk)$

Even Faster LCS!
The “Four Russians” Method
(Alzararov, Dinic, Kronrod, Faradzev, 1970)

Simplifying assumptions:
- Numbers only
- Care about time (not space)
- $n \times n$
Even Faster LCS!
The “Four Russians” Method
(Alrasarov, Dinic, Kronrod, Faradzev, 1970)

Simplifying assumptions:
- Numbers only
- Care about time (not space)
- n x n

CAAG

CAAG

n

n

n

n

n

n
Properties of the LCS table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Even Faster LCS!
The “Four Russians” Method
(Arazarov, Dinic, Kornrod, Faradzev, 1970)

- First, two factoids from high school:
  - “power dropping” $\log_b n^k = k \log_b n$
  - “log switching” $n^{\log_b a} = a^{\log_b n}$

Primary, Secondary, and Tertiary Structure

Watson-Crick pairing:
AU, GC, GU (weak)

“Like” DNA, but Thymine replaced by Uracil

Slide from Stefan Washietl’s 2012 guest lecture

Why Structure Matters

- Tertiary structure implies function (e.g., RNAs can catalyze chemical reactions!)
- But tertiary structure is hard to predict
- For RNA, secondary structure is a good proxy for tertiary structure and is easier to predict

Like everything in the universe, RNA seeks its lowest energy state…

Interactions stabilize molecule, putting it in a lower energy state

GC: Strong
AU: Weak
GU: Weaker

This is my low energy state!
Hairpin loop

Bifurcation loop

Stacking region

Why no base pairing here?

Interior loop

Compensatory Substitutions Maintain Secondary Structure

A region of ssrRNA with a conserved hairpin

Nussinov's Algorithm (1978)

Scoring
1 for each (AU, GC, GU) base pair
0 for unpaired bases
No pseudoknots!
Objective: Maximize total score

Prof. Ruth Nussinov
Head, Comp. Struct Bio Group, Tel Aviv U.

Pseudoknots are challenging!

Another challenging situation

Folding with Nussinov's Algorithm

>>> fold('GGGGAAAACCCC')
4

>>> fold('AAACCCAACCCUUCCCCCUUU')
5

>>> fold('A')
0
A Recursive Solution: Use-it-or-lose-it

AGCCCAUGCCUAAAAUC

“it”

Option 1: Lose “it”
Option 2: Use “it” (how?!)

The Recursive Solution

```
def fold(RNA):
    if len(RNA) <= 1: return 0
    else:
        loseIt = fold(RNA[1:])
        useIt = 0
        for k in range(1, len(RNA)):
            if pair(RNA[0], RNA[k]):
                option = 1 + fold(RNA[1:k]) + fold(RNA[k+1:])
            if option > useIt:
                useIt = option
        return max(loseIt, useIt)
```

A helper function that returns True if this pair matches

What changes would be needed to weight GC, AU, and GU differently?

This code is un-bear-ably slow!

What does the DP table look like?
How do we fill it in?

What’s the asymptotic worst-case running time as a function of \( n \), the length of the RNA string?

```
def fold(RNA):
    if len(RNA) <= 1: return 0
    else:
        loseIt = fold(RNA[1:])
        useIt = 0
        for k in range(1, len(RNA)):
            if pair(RNA[0], RNA[k]):
                option = 1 + fold(RNA[1:k]) + fold(RNA[k+1:])
            if option > useIt:
                useIt = option
        return max(loseIt, useIt)
```

AUUGCGUAUCCGUGUAUUCUGUGUACUCU

from \( i \) to \( j \)
Can we do it even faster?

Faster Algorithms for RNA-folding using the Four-Hussians method

Ralph Vershard, Dus Gustfeld, and Vesana Whid
Department of Computer Science, UC Davis
(ralph, gustfeld, vesana)@cs.ucdavis.edu

Abstract. The secondary structure that maximizes the number of non-crossing matchings between complementary bases of an RNA sequence of length n can be computed in O(n^2) time using Weisman’s dynamic program. That program is then further refined by the O(n^2) time dynamic programming algorithm by an exponential factor after a programming via storm solution to all substructures of a fixed non-substitutionally connected and second fold (Gustfeld) designed or (Gustfeld) algorithm for RNA folding using the Four-Hussians technique. In this algorithm the programming is initiated with the algorithmic minimization.

First paper: 2010, second paper July 2013 (under review, arXiv.org)

Knots Happen!

- Finding an optimal folding in a given thermodynamic model is NP-complete (Lyngso and Pedersen, 2000)
- But, numerous efforts have been made…

How about pseudoknots?

A Dynamic Programming Algorithm for RNA Structure Prediction Including Pseudoknots

Elena Rivas and Sean R. Eddy

A Dynamic Programming Algorithm for RNA Structure Prediction Including Pseudoknots

Elena Rivas and Sean R. Eddy

Department of Genetics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
Department of Computer Science, University of California, Davis, CA 95616, USA

We describe a dynamic programming algorithm for predicting optimal RNA secondary structures, including pseudoknots. The algorithm has a worst-case complexity of O(n^2) time and O(n^3) space. The structure of the algorithm is complex, which led us to adopt a graphical representation (Rivest-Diagram) borrowed from quantum field theory. We present an implementation of the algorithm that generates the configuration, that is, we lack a systematic a priori characterization of the class of configurations that this algorithm can solve.

P = NP?

Clay Mathematics Institute

The Zuker Algorithm

(1981, mfold software)

Nussinov’s algorithm models energies due to Watson-Crick pairs

Zuker’s algorithm models energies due to substructures:

- Stacking (S) PAIRS of PAIRS!
- Hairpin loop (H)
- Bulge loop (BU)
- Bifurcation loop (BF)
- Interior loop (I)

Still no pseudoknots!
Big idea:
Assume that we have a function that takes a face as input and returns its energy. For every possible valid fold, add up the energies of all resulting faces to obtain total energy. Choose the fold that minimizes total energy (maximizes stability).

Computing $V(i, j)$: So, $i$ and $j$ are connected by an interior edge.

- Case 1: Hairpin loop. So, no other interior edges between $i$ and $j$. Energy is $E(F)$.

- Case 2: Edge $i, j$ “sees” exactly one interior edge between $i$ and $j$. That edge goes between some $i'$ and $j'$ ($i' < i < j < j'$). Energy is computed as...
$W(i, j) = \text{optimal solution for substring from } i \text{ to } j$

$V(i, j) = \text{optimal solution for substring from } i \text{ to } j$

assuming $i$ and $j$ are matched

Computing $V(i, j)$: So, $i$ and $j$ are connected by an interior edge

Case 1: Hairpin loop. So, no other interior edges between $i$ and $j$.

Energy is $E(F)$.

Case 2: Edge $i, j$ "sees" exactly one interior edge between $i$ and $j$. That edge goes between some $i'$ and $j'$ ($i < i' < j'$).

Energy is computed as:

$E(F_{i', j'}) + V(i', j')$

We need to consider every possible $i'$ and $j'$ but the DP will have these values already computed!

Remaining cases:

$i, j$ sees 2 "chords"

$i, j$ sees 3 "chords"

I think we're in serious trouble now!

Assume that bifurcation loops have energy 0. Zuker, 1981

What a nice face!

Why is this computationally useful?

There exists some $k$ ($i+2 < k < j-2$) that "splits" the chords on one side from chords on the other side of $k$ (WHY?)

$V(i, j) = \min_{i+2 < k < j-2} W(i+1, k) + W(k+1, j-1)$. 

Why is this computationally useful?
Putting it together

\[
V(i, j) = \min \{ \\
E(F) + \min_{i \leq i' \leq j} V(i', j'), \\
\min_{i+2 \leq k \leq j-2} W(i+1, k) + W(k+1, j-1) \} \\
\]

But how do we compute \( W(i, j) \)?

\[
W(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
V(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
\text{assuming } i \text{ and } j \text{ are matched} \\
\]

What have we not yet considered?

\[
W(i, j) = \min \{ \ldots \} \\
\]

But how do we compute \( W(i, j) \)?

\[
W(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
V(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
\text{assuming } i \text{ and } j \text{ are matched} \\
\]

\[
W(i, j) = \min \{ V(i, j), \, W(i+1, j), \ldots \} \\
\]

What have we still not yet considered?

But how do we compute \( W(i, j) \)?

\[
W(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
V(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
\text{assuming } i \text{ and } j \text{ are matched} \\
\]

\[
W(i, j) = \min \{ V(i, j), W(i+1, j), W(i, j-1), \ldots \} \\
\]

What have we still not yet considered?

But how do we compute \( W(i, j) \)?

\[
W(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
V(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
\text{assuming } i \text{ and } j \text{ are matched} \\
\]

\[
W(i, j) = \min \{ V(i, j), W(i+1, j), W(i, j-1), \ldots \} \\
\]

What have we still not yet considered?

But how do we compute \( W(i, j) \)?

\[
W(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
V(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
\text{assuming } i \text{ and } j \text{ are matched} \\
\]

\[
W(i, j) = \min \{ V(i, j), W(i+1, j), W(i, j-1), \ldots \} \\
\]

What have we still not yet considered?

But how do we compute \( W(i, j) \)?

\[
W(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
V(i, j) = \text{optimal solution for substring from } i \text{ to } j \\
\text{assuming } i \text{ and } j \text{ are matched} \\
\]

\[
W(i, j) = \min \{ V(i, j), W(i+1, j), W(i, j-1), \ldots \} \\
\]

What have we still not yet considered?
Computing V and W

\[ V(i, j) = \min \left\{ \sum_{i \leq i' < j} V(i', j), \ \sum_{i < j' < j} W(i, j') + W(i+1, k) + W(k+1, j-1) \right\} \]

\[ W(i, j) = \min \{ V(i, j), W(i+1, j), W(i, j-1), \ \text{min}_{i < i' < j-1} V(i, i') + W(i'+1, j) \} \]

Stochastic Context-Free Grammars (SCFGs)

Rather than optimizing folds based on a thermodynamic model, use machine learning methods to learn a stochastic model from examples. Similar to Hidden Markov Model approach (but more "powerful")

A generative model!

Application 1

Some sequence \( \xrightarrow{\text{SCFG RNA model}} \) How likely is it that this is a foldable RNA? (What is the sum of the probs. of all possible folds for this string?)

RNA sequence \( \xrightarrow{\text{SCFG RNA model}} \) Maximum probability fold
A quick primer on grammars...

Upper case letters are variables:
S → aB | bA | ε
A → aS | bAA

B → bS | aBB

Lower case letters are terminals:
S is the start variable
ε is the empty string

Variable B...
Can be replaced by bS or aBB

Derivations and Parse Trees

A parse tree
S
 a
 B
 a
 B
 b S
 b S
 ε
 ε

A derivation:
S => aB => aaBB => aabSB => aabB => aabbS => aabb

What is the set of sequences generated by this grammar?

A (too) simple grammar for RNA

S → aSu | uSa | gSc | cSg | gSu | uSg |
aS | Sa | gS | Sg | uS | Su | cS | Sc |
SS | ε

What’s “wrong” here?

Parse Trees, Folds, and Ambiguity

S → aSu | uSa | gSc | cSg | gSu | uSg |
aS | Sa | gS | Sg | uS | Su | cS | Sc |
SS | ε

Parse: acaggaacuguaacggucgaaccg

Real Headlines…

“Iraqi Head Seeks Arms”
“Red tape holds up bridge”
“Police begin campaign to run down jay-walkers”
“Farmer bill dies in house”
“Enraged cow injures farmer with ax”
“Milk drinkers are turning to powder”
“British left waffles on Falkland Islands”
“Safety experts say passengers should be belted”

Real Headlines…

“Iraqi Head Seeks Arms”
“Red tape holds up bridge”
“Police begin campaign to run down jay-walkers”
“Farmer bill dies in house”
“Enraged cow injures farmer with ax”
“Milk drinkers are turning to powder”
“British left waffles on Falkland Islands”
“Safety experts say passengers should be belted”
Real Headlines…

“Iraqi Head Seeks Arms”
“Red tape holds up bridge”
“Police begin campaign to run down jay-walkers”
“Farmer Bill Dies in House”
“Enraged cow injures farmer with ax”
“Milk drinkers are turning to powder”
“British left waffles on Falkland Islands”
“Safety experts say passengers should be belted”

Real Headlines…

“Iraqi Head Seeks Arms”
“Red tape holds up bridge”
“Police begin campaign to run down jay-walkers”
“Farmer Bill Dies in House”
“Enraged cow injures farmer with ax”
“Milk drinkers are turning to powder”
“British left waffles on Falkland Islands”
“Safety experts say passengers should be belted”

Real Headlines…

“Iraqi Head Seeks Arms”
“Red tape holds up bridge”
“Police begin campaign to run down jay-walkers”
“Farmer Bill Dies in House”
“Enraged cow injures farmer with ax”
“Milk drinkers are turning to powder”
“British left waffles on Falkland Islands”
“Safety experts say passengers should be belted”

Real Headlines…

“Iraqi Head Seeks Arms”
“Red tape holds up bridge”
“Police begin campaign to run down jay-walkers”
“Farmer Bill Dies in House”
“Enraged cow injures farmer with ax”
“Milk drinkers are turning to powder”
“British left waffles on Falkland Islands”
“Safety experts say passengers should be belted”
Reconstructing Parse Trees

\[ S \rightarrow aSu \mid uSa \mid gSc \mid cSg \mid gSu \mid uSg \mid aS \mid Sa \mid gS \mid Sg \mid uS \mid Su \mid cS \mid Sc \mid SS \mid \varepsilon \]

Parse: acaggacugauacgugucsaaccg

def Parsable(string, i, j):
    ''' returns True if string from index i to j is parsable and False otherwise '''
    if i > j: return True  # S can generate empty string
    if string[i] == 'a' and string[j] == 'u':
        check1 = Parsable(string, i+1, j-1)
    if string[i] == 'u' and string[j] == 'a':
        check2 = Parsable(string, i+1, j-1)
    ...
    ???

- How do we deal with SS?
- What do we return at the end?
- What does the DP table look like?
- What's the running time of the DP?
- How can we recover the parse trees?

In general...

- Convert grammar to a special form called Chomsky Normal Form
- Use the DP we just described (known as the CYK Algorithm) to parse
- Can we do it faster?

Stochastic CFGs

\[ S \rightarrow aXu \mid uXa \mid cXg \mid gXc \mid uXg \mid gXu \]
\[ X \rightarrow aYu \mid uYa \mid cYg \mid gYc \mid uYg \mid gYu \]
\[ Z \rightarrow agag \mid cccc \]

Problem 0: For a given grammar, how do we learn the probabilities from a training set?
Problem 1: Given a sequence (a string), what is the most likely derivation = parse tree = fold? (For a given grammar and probs.)
Problem 2: Given a sequence (a string), what's the total probability that this string would be generated by this model? (For a given grammar and probs.)

What is the most probable folding?

- We want to maximize a product of probabilities \( p_1 p_2 \cdots p_k \)
- Equivalent to maximizing their log...

\[ \log p_1 p_2 \cdots p_k = \log p_1 + \log p_2 + \cdots + \log p_k \]
RNA folding software

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
<th>Links</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>FoldmRNA</td>
<td>Secondary structure prediction based on generalised terminal estimation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semarin</td>
<td>Secondary structure prediction by using alignment information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DnaAlgo</td>
<td>Secondary structure prediction by using folding information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNAfold</td>
<td>An RNA secondary structure prediction software based on thermodynamic models</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CORNA</td>
<td>Secondary structure prediction method based on combinatorial linear models</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRT</td>
<td>A method of stabilizing unstructured RNA sequences</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPfold</td>
<td>Secondary structure prediction method based on thermodynamic theory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Foldx</td>
<td>Foldx is a secondary structure prediction tool with a number of modules</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PredictRNA</td>
<td>PredictRNA is a software tool for predicting RNA secondary structures</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOLD-PC</td>
<td>FOLD-PC is a secondary structure prediction tool that uses physics-based</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNAfold-ai</td>
<td>RNAfold-ai is a secondary structure prediction tool that uses artificial</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNAfold-2d</td>
<td>RNAfold-2d is a secondary structure prediction tool that uses physics-based</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNAfold-ai2</td>
<td>RNAfold-ai2 is a secondary structure prediction tool that uses artificial</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SCFGs are “more powerful” than HMMs...

```
S         aS | cS | aA | cA | aB | cB
A         aA | ...
B         ...
```

Learning objectives:

- A “regular grammar”
- Regular, Context-free, and other grammars
- Unrestricted grammar

<table>
<thead>
<tr>
<th>HMM algorithm</th>
<th>SCFG algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimal alignment</td>
<td>Viterbi</td>
</tr>
<tr>
<td>probability of sequence</td>
<td>forward</td>
</tr>
<tr>
<td>EM parameter estimation</td>
<td>forward-backward</td>
</tr>
<tr>
<td>memory complexity</td>
<td>$O(L^2)$</td>
</tr>
<tr>
<td>time complexity</td>
<td>$O(LM^2)$</td>
</tr>
</tbody>
</table>

Borrowed from Colin Dewey, CS 776, U. Wisconsin, Spring 2013

The “Send Ran to Spago” Story

```
L = length of string
M = number of states
```

Unrestricted grammar:

```
S         aAu | ... | uAu a | gAc | cAg | ... | uccDgu |
A         aBu | ...
```

Borrowed from Colin Dewey, CS 776, U. Wisconsin, Spring 2013
Further Reading (courtesy of Stefan Washietl)

Overview

RNA function: review papers by John Mattick

Single sequence RNA folding:

Consensus RNA folding:

RNA gene finding: