RNA Folding

CMSC 423
Lecture by Darya Filippova
RNA Folding

RNA is single stranded and folds up:
- G and C stick together
- A and U stick together
RNA Folding Rules

**RNA folding rules:**

1. If two bases are closer than 4 bases apart, they cannot pair.
2. Each base is matched to at most one other base.
3. The allowable pairs are \{U, A\} and \{C, G\}.
4. Pairs cannot “cross.”
If \((i,j)\) and \((k,m)\) are paired, we must have \(i < k < m < j\).

Paired bases have to be **nested**.
Given: a string $r = b_1b_2b_3,...,b_n$ with $b_i \in \{A,C,U,G\}$
Find: the largest set of pairs $S = \{(i,j)\}$, where $i,j \in \{1,2,...,n\}$
that satisfies the RNA folding rules.

Goal: match as many bases as possible.
Subproblems

- **j** is not paired with anything
  - OPT(1, j-1)

- **j** is paired with some **t** ≤ j - 4
  - OPT(1, t-1)
  - OPT(t+1, j-1)
Recurrence

**If** \( j - i \leq 4 \):

\[
OPT(i, j) = 0
\]

**If** \( j - i > 4 \):

\[
OPT(i, j) = \max \left\{ OPT(i, j - 1), \max_t \{1 + OPT(i, t - 1) + OPT(t + 1, j - 1)\} \right\}
\]

In the 2nd case above, we try all possible \( t \) with which to pair \( j \). That is, \( t \) runs from \( i \) to \( j-4 \).
Order to solve the subproblems

• In what order should we solve the subproblems?
Order to solve the subproblems

• In what order should we solve the subproblems?

• What problems do we need to solve $OPT(i,j)$?
  
  $OPT(i,t-1)$ and $OPT(t+1,j-1)$
  for every $t$ between $i$ and $j$

• In what sense are these problems “smaller?”
Order to solve the subproblems

• In what order should we solve the subproblems?

• What problems do we need to solve \( OPT(i,j) \)?

\[
OPT(i,t-1) \text{ and } OPT(t+1,j-1)
\]
for every \( t \) between \( i \) and \( j \)

• In what sense are these problems “smaller?”

• They involve smaller intervals of the string:

We solve \( OPT(i,j) \) in order of increase value of \( j - i \).
Filling in the matrix

only use half: $i < j$

$OPT(i,j)$
Filling in the matrix

in order of increasing j-i
Filling in the matrix

in order of increasing j-i
Filling in the matrix

in order of increasing j-i
Filling in the matrix

in order of increasing j-i
Filling in the matrix in order of increasing $j - i$
Case 1

\[ OPT(i,j) = \max \left\{ OPT(i,j-1), \ldots \right\} \]
Case 1

\[ OPT(i, j) = \max \left\{ OPT(i, j - 1), \ldots \right\} \]
Case 2

\[ OPT(i, j) = \max \left\{ \ldots \max_t \{1 + OPT(i, t - 1) + OPT(t + 1, j - 1)\} \right\} \]
Case 2

\[ OPT(i, j) = \max \left\{ \ldots \max_t \{ 1 + OPT(i, t - 1) + OPT(t + 1, j - 1) \} \right\} \]
Case 2

\[ OPT(i, j) = \max \left\{ \ldots \max_t \{1 + OPT(i, t - 1) + OPT(t + 1, j - 1)\} \right\} \]
def rnafold(rna):
    n = len(rna)
    OPT = make_matrix(n, n)
    Arrows = make_matrix(n, n)
    for k in xrange(5, n):     # interval length
        for i in xrange(n-k):  # interval start
            j = i + k          # interval end
            best_t = OPT[i][j-1]
            arrow = -1
            for t in xrange(i, j):
                if is_complement(rna[t], rna[j]):
                    val = 1 + \\
                    (OPT[i][t-1] if t > i else 0) + OPT[t+1][j-1]
                    if val >= best_t: best_t, arrow = val, t
            OPT[i][j] = best_t
            Arrows[i][j] = arrow
    return OPT, Arrows
def rna_backtrace(Arrows):
    Pairs = []  # holds the pairs in the optimal solution
    Stack = [(0, len(Arrows) - 1)]  # tracks cells we have to visit
    while len(Stack) > 0:
        i, j = Stack.pop()
        if j - i <= 4: continue  # if cell is base case, skip it
        if Arrows[i][j] == -1:
            Stack.append((i, j - 1))
        else:
            t = Arrows[i][j]
            Pairs.append((t, j))  # save that j matched with t
            # add the two daughter problems
            if t > i: Stack.append((i, t - 1))
            Stack.append((t + 1, j - 1))

    return Pairs
Subproblems, 2

• We have a subproblem for every interval \((i,j)\)

• How many subproblems are there?
Subproblems, 2

- We have a subproblem for every interval \((i,j)\)
- How many subproblems are there?

\[
\binom{n}{2} = O(n^2)
\]
Running Time

- $O(n^2)$ subproblems
- Each takes $O(n)$ time to solve (have to search over all possible choices of t)
- Total running time is $O(n^3)$. 
Summary

• This is essentially “Nussinov’s algorithm,” which was proposed for finding RNA structures in 1978.

• Same dynamic programming idea: write the answer to the full problem in terms of the answer to smaller problems.

• Still have an $O(n^2)$ matrix to fill.

• Main differences from sequence alignment:
  • We fill in the matrix in a different order: entries $(i,j)$ in order of increasing $j - i$.
  • We have to try $O(n)$ possible subproblems inside the max. This leads to an $O(n^3)$ algorithm.