Objectives

- Dynamic Programming
  - RNA Substructure
  - Sequence Alignment

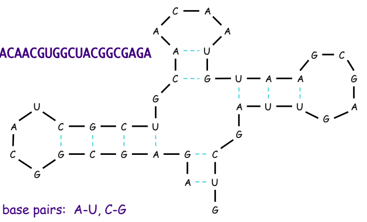
Applications of Dynamic Programming to Computational Biology

RNA SECONDARY STRUCTURE

RNA Secondary Structure

- RNA. String B = $b_1b_2\ldots b_n$ over alphabet { A, C, G, U }
- Secondary structure. RNA is *single-stranded* so it tends to loop back and form base pairs with itself
  - This structure is essential for understanding behavior of a molecule.

Ex: GUCGAUUGAGCGAAUGUAACAACGUGGCUACGGCGAGA



complementary base pairs: A-U, C-G

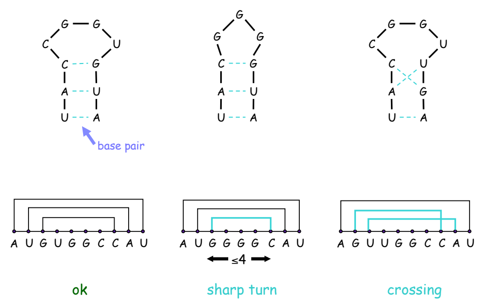
RNA Secondary Structure: Which Pairs Can We Combine?

- A set of pairs S = { (b_i, b_j) } that satisfy:
  - [Watson-Crick] S is a *matching* and each pair in S is a Watson-Crick complement: A-U, U-A, C-G, or G-C
  - [No sharp turns] The ends of each pair are separated by *at least 4* intervening bases. If $(b_i, b_j) \in S$, then $i < j - 4$
  - [Non-crossing] If (b_i, b_j) and (b_k, b_l) are two pairs in S, then we cannot have $i < k < j < l$

RNA Secondary Structure

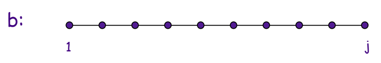
- A set of pairs S = { (b_i, b_j) } that satisfy:
  - [Watson-Crick] S is a *matching* and each pair in S is a Watson-Crick complement: A-U, U-A, C-G, or G-C
  - [No sharp turns] The ends of each pair are separated by at least 4 intervening bases. If $(b_i, b_j) \in S$, then $i < j - 4$
  - [Non-crossing] If (b_i, b_j) and (b_k, b_l) are two pairs in S, then we cannot have $i < k < j < l$
- Free energy. Usual hypothesis is that an RNA molecule will form the secondary structure with the *optimum total free energy*. ← *approximate by number of base pairs*
- Goal. Given an RNA molecule B = $b_1b_2\ldots b_n$, find a secondary structure S that *maximizes the number of base pairs*

Examples of RNA Secondary Structure



Toward a Solution: First Attempt

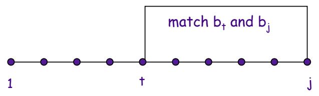
- OPT(j) = maximum number of base pairs in a secondary structure of the substring $b_1b_2\ldots b_j$



- Towards a recurrence relation…
  - What are the possibilities?
    - What does b_j match with?
  - What are the subproblems?

Toward a Solution: First Attempt

- OPT(j) = maximum number of base pairs in a secondary structure of the substring $b_1b_2\ldots b_j$



- Relation:
  - If j isn't involved in a pair: Opt(j-1)
  - If j is involved, results in two sub-problems
    - Finding secondary structure in: $b_1b_2\ldots b_{t-1}$ ← OPT(t-1)
    - Finding secondary structure in: $b_{t+1}b_{t+2}\ldots b_{j-1}$ ← need more subproblems

Doesn't match our subproblem (doesn't start at 1)
Need to start *anywhere*

Dynamic Programming Over Intervals

- Notation. OPT(i, j) = maximum number of base pairs in a secondary structure of the substring $b_ib_{i+1}\ldots b_j$
  - What are the different cases?
  - How does it affect the recurrence relation?
    - For example, when will we know that there isn't a pair?

Dynamic Programming Over Intervals

- Notation. OPT(i, j) = maximum number of base pairs in a secondary structure of the substring $b_ib_{i+1}\ldots b_j$
  - Case 1. If $i \geq j - 4$
    - OPT(i, j) = 0 by *no-sharp turns* condition
  - Case 2. Base b_j is not involved in a pair
    - OPT(i, j) = OPT(i, j-1)
  - Case 3. Base b_j pairs with b_t for some $i \leq t < j - 4$
    - *non-crossing* constraint decouples resulting sub-problems
    - $\text{OPT}(i, j) = 1 + \max_t \{ \text{OPT}(i, t-1) + \text{OPT}(t+1, j-1) \}$

pairing

take max over t such that $i \leq t < j-4$ and b_t and b_j are Watson-Crick complements

Recurrence Relation

- Putting it all together…

j not in a base pair in optimal solution

- $\text{Opt}(i,j) = \max(\ \text{Opt}(i,j-1),\ \max_t(\ 1+\text{Opt}(i,t-1)+\text{Opt}(t+1,j-1)\ )\ )$

j in a base pair in optimal solution
Adds 1 pair
Look at remaining letters

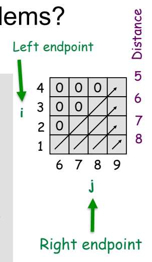
RNA Algorithm

- What order to solve the sub-problems?
  - Do shortest intervals first

```
Initialize M[i,j] = 0 for i >= j-4

RNA(b_1,…,b_n):
   for k = 5, 6, …, n-1        (distances)
      for i = 1, 2, …, n-k     (start)
         j = i + k             (end)
         M[i, j] = max(M[i,j-1],
                  max_t(1+M[i,t-1]+M[t+1,j-1]) )

   return M[1, n]
```



Costs?

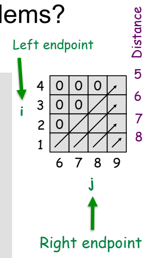
RNA Algorithm

- What order to solve the sub-problems?
  - Do shortest intervals first

```
Initialize M[i,j] = 0 for i >= j-4

RNA(b_1,…,b_n):
   for k = 5, 6, …, n-1          (distances)
      for i = 1, 2, …, n-k       (start)
         j = i + k          (end)
         M[i, j] = max(M[i,j-1],
                   max_t(1+M[i,t-1]+M[t+1,j-1]) )

   return M[1, n]
```



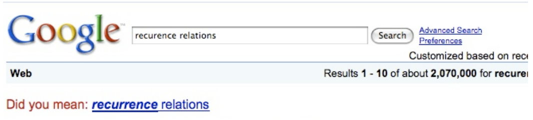
- Running time: $O(n^3)$

Dynamic Programming Summary

- Recipe
  - Characterize structure of problem
  - Recursively define *value* of optimal solution
  - Compute value of optimal solution
  - Construct *optimal solution* from computed information
- Dynamic programming techniques
  - Binary choice: weighted interval scheduling
  - Multi-way choice: segmented least squares
  - Adding a new variable: knapsack
  - Dynamic programming over intervals: RNA secondary structure
- Top-down vs. bottom-up: different people have different intuitions

SEQUENCE ALIGNMENT

Problem


recurence relations

Advanced Search
Preferences
Customized based on rec

Web — Results **1** - **10** of about **2,070,000** for **recure**

Did you mean: ***recurrence*** relations

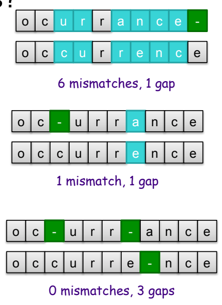
How does Google know what I really meant?

String Similarity

- How similar are two strings?
  - ocurrance
  - occurrence
- We intuitively can tell that these two are similar
  - Systematic measurement?

String Similarity

- How similar are two strings?
  - ocurrance
  - occurrence
- Measurements
  - Gap (-): add a letter
  - Mismatch



6 mismatches, 1 gap

1 mismatch, 1 gap

0 mismatches, 3 gaps

Which is the best measurement?

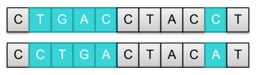
Applications of String Similarity

- Basis for Unix `diff`
  - Longest common subsequence
- Spam filters
  - Similarity to known spam message
- Computational biology
  - Ex: Figuring out how similar two genomes (sequences of A, C, G, T) are
- Alignment with **non** English/natural language strings are less obvious how to align

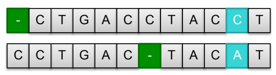
Edit Distance

- [Levenshtein 1966, Needleman-Wunsch 1970]
  - Gap penalty: δ
  - Mismatch penalty: α_{pq}
    - If p and q are the same, then mismatch penalty is 0
  - **Cost** = sum of gap and mismatch penalties

Parameters allow us to tweak cost



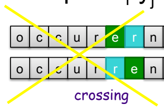
$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$



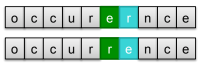
$2\delta + \alpha_{CA}$

Sequence Alignment

- Goal: Given two strings $X = x_1\ x_2 \ldots x_m$ and $Y = y_1\ y_2 \ldots y_n$ find alignment of minimum cost
- An *alignment* M is a set of ordered pairs x_i-y_j such that each item occurs in at most one pair and **no** crossings
- The pair x_i-y_j and $x_{i'}$-$y_{j'}$ *cross* if i < i', but j > j'.



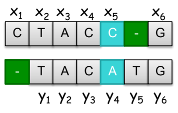
crossing



2 mismatches

Sequence Alignment Example

- X = CTACCG
- Y = TACTG
- Solution: $M = x_2\text{-}y_1,\ x_3\text{-}y_2,\ x_4\text{-}y_3,\ x_5\text{-}y_4,\ x_6\text{-}y_6$



What is the cost of M?

$$\text{cost}(M) = \underbrace{\sum_{(x_i,y_j)\in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i:x_i \text{ unmatched}} \delta + \sum_{j:y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Recall: mismatch penalty is 0 if x_i and y_j are the same

Sequence Alignment Case Analysis

- Consider the last character of the strings X and Y: x_M and y_N
- What are the possibilities for x_M and y_N in terms of the alignment?

Sequence Alignment Case Analysis

- Consider last character of strings X and Y: x_M and y_N
  - Case 1: x_M and y_N are aligned
  - Case 2: x_M is not matched
  - Case 3: y_N is not matched

Formulate the optimal solution's value

Sequence Alignment Case Analysis

- Consider last character of strings X and Y: x_M and y_N
  - Case 1: x_M and y_N are aligned
  - Case 2: x_M is not matched
  - Case 3: y_N is not matched

What are the costs for these cases?

- $\overset{x}{\mathrm{OPT}(i,}\ \overset{y}{j)}$ = min cost of aligning strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$

Sequence Alignment Cost Analysis

- Consider last character of strings X and Y: x_M and y_N
  - Case 1: x_M and y_N are aligned
    - Pay mismatch for x_M-y_N + min cost of aligning rest of strings
    - OPT(M, N) = $\alpha_{x_M y_N}$ + OPT(M-1, N-1)
  - Case 2: x_M is not matched
    - Pay gap for x_M + min cost of aligning rest of strings
    - OPT(M, N) = δ + OPT(M-1, N)
  - Case 3: y_N is not matched
    - Pay gap for y_N + min cost of aligning rest of strings
    - OPT(M, N) = δ + OPT(M, N-1)

Sequence Alignment Cost Analysis

- Base costs? → i or j is 0
  - What happens when we run out of letters in one string before the other?

X = CTACCG
Y = TACTG

Sequence Alignment: Problem Structure

Gaps for remainder of Y

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i=0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j=0 \end{cases}$$

Gaps for remainder of X

Sequence Alignment: Algorithm

Cost parameters

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

Costs?