CSE 421

RNA Secondary Structure, Sequence Alignment

Shayan Oveis Gharan
Knapsack Problem: Running Time

Running time: $\Theta(n \cdot W)$
- Not polynomial in input size!
- "Pseudo-polynomial."
- Decision version of Knapsack is NP-complete.

Knapsack approximation algorithm:
There exists a polynomial algorithm that produces a feasible solution that has value within 0.01% of optimum in time Poly(n, log W).
DP Ideas so far

• You may have to define an ordering to decrease the #subproblems

• You may have to strengthen DP, equivalently the induction, i.e., you have may have to carry more information to find the Optimum.

• This means that sometimes we may have to use two dimensional or three dimensional induction
RNA Secondary Structure
RNA Secondary Structure

RNA: A String $B = b_1b_2...b_n$ 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 molecule.

Ex: GUCGAUUGAGCGAUGUAACAACGUGGCUACGGCGAGA

complementary base pairs: A-U, C-G
RNA Secondary Structure (Formal)

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 pair: 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 maximize total free energy.

Goal: Given an RNA molecule $B = b_1b_2\ldots b_n$, find a secondary structure $S$ that maximizes the number of base pairs.

approximate by number of base pairs
Secondary Structure (Examples)

- Base pair
- Sharp turn
- Crossing
First attempt. Let $OPT(n) =$ maximum number of base pairs in a secondary structure of the substring $b_1b_2\ldots b_n$.

Suppose $b_n$ is matched with $b_t$ in $OPT(n)$.

What IH should we use?

**Difficulty:** This naturally reduces to two subproblems

- Finding secondary structure in $b_1, ..., b_{t-1}$, i.e., $OPT(t-1)$
- Finding secondary structure in $b_{t+1}, ..., b_{n-1}$, ???

\[ \begin{array}{c}
1 & \ldots & t & \ldots & n \\
\end{array} \]
Definition: $OPT(i, j) = \text{maximum number of base pairs in a secondary structure of the substring } b_i, b_{i+1}, ..., b_j$

The most important part of a correct DP; It fixes IH

Case 1: If $j - i \leq 4$.
- $OPT(i, j) = 0$ 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
- $OPT(i, j) = \max_{t: b_i \text{ pairs with } b_t} \{ 1 + OPT(i, t - 1) + OPT(t + 1, j - 1) \}$
Recursive Code

Let \( M[i,j] = \text{empty} \) for all \( i, j \).

\[
\begin{align*}
\text{Compute-OPT}(i,j) \{ \\
    &\textbf{if } (j-i \leq 4) \\
    &\quad \text{return } 0; \\
    &\textbf{if } (M[i,j] \text{ is empty}) \\
    &\quad M[i,j] = \text{Compute-OPT}(i,j-1) \\
    &\quad \textbf{for } t = i \text{ to } j-5 \text{ do} \\
    &\quad \quad \textbf{if } (b_t, b_j \text{ is in } \{A-U, U-A, C-G, G-C\}) \\
    &\quad \quad \quad M[i,j] = \max(M[i,j], 1 + \text{Compute-OPT}(i,t-1) + \\
    &\quad \quad \quad \quad \text{Compute-OPT}(t+1,j-1)) \\
    &\quad \text{return } M[j] \\
\}
\]

Does this code terminate? What are we inducting on?
Formal Induction

Let $OPT(i, j) =$ maximum number of base pairs in a secondary structure of the substring $b_i, b_{i+1}, ..., b_j$

**Base Case:** $OPT(i, j) = 0$ for all $i, j$ where $|j - i| \leq 4$.

**IH:** For some $\ell \geq 4$, Suppose we have computed $OPT(i, j)$ for all $i, j$ where $|i - j| \leq \ell$.

**IS:** Goal: We find $OPT(i, j)$ for all $i, j$ where $|i - j| = \ell + 1$. Fix $i, j$ such that $|i - j| = \ell + 1$.

**Case 1:** Base $b_j$ is not involved in a pair.
- $OPT(i, j) = OPT(i, j-1)$ [this we know by IH since $|i - (j - 1)| = \ell$]

**Case 2:** Base $b_j$ pairs with $b_t$ for some $i \leq t < j - 4$
- $OPT(i, j) = \max_{t: b_i \text{ pairs with } b_t} \{ 1 + OPT(i, t - 1) + OPT(t + 1, j - 1) \}$

  We know by IH since difference $\leq \ell$
Bottom-up DP

for k = 1, 2, ..., n-1
    for i = 1, 2, ..., n-1
        j = i + k
        if (j-i <= 4)
            M[i,j]=0;
        else
            M[i,j]=M[i,j-1]
        for t=i to j-5 do
            if (b_t, b_j is in {A-U, U-A, C-G, G-C})
                M[i,j]=max(M[i,j], 1 + M[i,t-1] + M[t+1,j-1])

return M[1, n]

Running Time: $O(n^3)$
Lesson

We may not always induct on $i$ or $w$ to get to smaller subproblems.

We may have to induct on $|i - j|$ or $i + j$ when we are dealing with more complex problems, e.g., intervals.
Sequence Alignment
Word Alignment

How similar are two strings?

occurrence

occurrance

occurrence

5 mismatches, 1 gap

1 mismatch, 1 gap

0 mismatches, 3 gaps
Edit Distance

Cost = # of gaps + #mismatches.

Applications.
• Basis for Unix diff and Word correct in editors.
• Speech recognition.
• Computational biology.

Cost: 5

Cost: 3
Sequence Alignment

Given two strings $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ find an alignment with minimum number of mismatch and gaps.

An alignment is a set of ordered pairs $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}), \ldots$ such that $i_1 < i_2 < \cdots$ and $j_1 < j_2 < \cdots$

Example: CTACCG vs. TACATG.
Sol: We aligned $x_2$-$y_1$, $x_3$-$y_2$, $x_4$-$y_3$, $x_5$-$y_4$, $x_6$-$y_6$.
So, the cost is 3.
Let $OPT(i, j)$ be min cost of aligning $x_1, ..., x_i$ and $y_1, ..., y_j$

**Case 1**: OPT matches $x_i, y_j$
- Then, pay mis-match cost if $x_i \neq y_j$ + min cost of aligning $x_1, ..., x_{i-1}$ and $y_1, ..., y_{j-1}$ i.e., $OPT(i - 1, j - 1)$

**Case 2**: OPT leaves $x_i$ unmatched
- Then, pay gap cost for $x_i + OPT(i - 1, j)$

**Case 3**: OPT leaves $y_j$ unmatched
- Then, pay gap cost for $y_j + OPT(i, j - 1)$
Bottom-up DP

```c
Sequence-Alignment(m, n, x_1x_2...x_m, y_1y_2...y_n) {
    for i = 0 to m
        M[0, i] = i
    for j = 0 to n
        M[j, 0] = j

    for i = 1 to m
        for j = 1 to n
            M[i, j] = min( (x_i=y_j ? 0:1) + M[i-1, j-1],
                            1 + M[i-1, j],
                            1 + M[i, j-1])

    return M[m, n]
}
```

Analysis: \( \Theta(mn) \) time and space.

**English words or sentences**: \( m, n \leq 10,\ldots,20 \).

**Computational biology**: \( m = n = 100,000 \). 10 billions ops OK, but 40GB array?
Optimizing Memory

If we are not using strong induction in the DP, we just need to use the last (row) of computed values.

```plaintext
Sequence-Alignment(m, n, x_1 x_2 ... x_m, y_1 y_2 ... y_n) {
    for i = 0 to m
        M[0, i] = i
    for j = 0 to n
        M[j, 0] = j

    for i = 1 to m
        for j = 1 to n
            M[i, j] = min( (x_i=y_j ? 0:1) + M[i-1, j-1],
                            1 + M[i-1, j],
                            1 + M[i, j-1])

    return M[m, n]
}
```

Just need $i - 1, i$ rows to compute $M[i,j]$
DP with $O(m + n)$ memory

- Keep an Old array containing values of the last row
- Fill out the new values in a New array
- Copy new to old at the end of the loop

```cpp
Sequence-Alignment(m, n, x1x2...x_m, y1y2...y_n) {
    for i = 0 to m
        O[i] = i
    for i = 1 to m
        N[0]=i
        for j = 1 to n
            N[j] = min( (x_i=y_j ? 0:1) + O[j-1],
                        1 + O[j],
                        1 + N[j-1])
    for j = 1 to n
        O[j]=N[j]
    return N[n]
}
```
Lesson

Advantage of a bottom-up DP:

It is much easier to optimize the space.