CSE 566 Spring 2023

Sequence Alignment

Instructor: Mingfu Shao

non-crossing Sequence Alignment

- Optimization problem: given two string S and T, find an *alignment* A such that cost(A) is minimized.

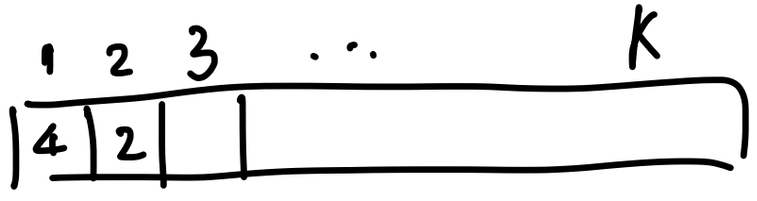
```
S = A C C G A C T A C T C G T T G A C
T = A C C G G A C C T C G T T G T C
```

```
S = A C C - G A C T A C T C G T T G A C
    | | |   | | |     | | | | | | | |
T = A C C G G A C - - C T C G T T G T C

    A C C - G A C - T A C - T C G T T G A C
    | | |   | | |   | | |   | |   |       |
    A C C G G A C C T C G T T G - T - - - C
```

Cost Functions

- Cost of mismatch(substitution)
  - Unit cost: each cost 1
  - Arbitrary cost for different x-y A-T: 3. G-A: 4
- Cost of gap (function of length of a gap)
  - Unit cost: gap(k) = k
  - Affine gap cost: gap(k) = gap-open + k * gap-extend (20, 3)
  - Convex gap cost: for any a < b < c < d,
    gap(d-a) + gap(c-b) >= gap(c-a) + gap(d-b)
  - General cost: arbitrary function of k



Alignment with Unit Cost

- Edit distance problem: the smallest number of substitutions, insertions and deletions that can transform S into T.

mismatch

```
S= ACC-GACTACTCGTTGAC
   |||  |||   |||||||||
T= ACCGGAC--CTCGTTGTC
```

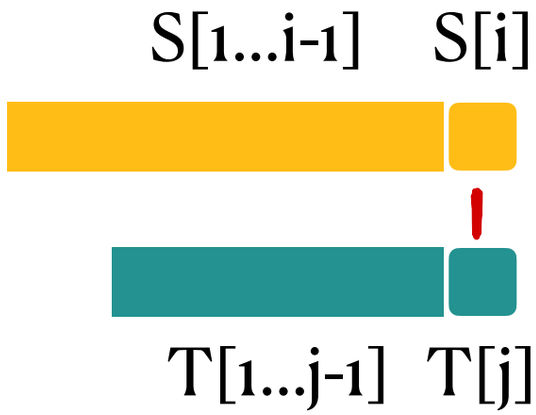
cost = 4

```
ACC-GAC-TAC-TCGTTGAC
|||  |||  |||  ||  |     |
ACCGGACCTCGTTG-T---C
```

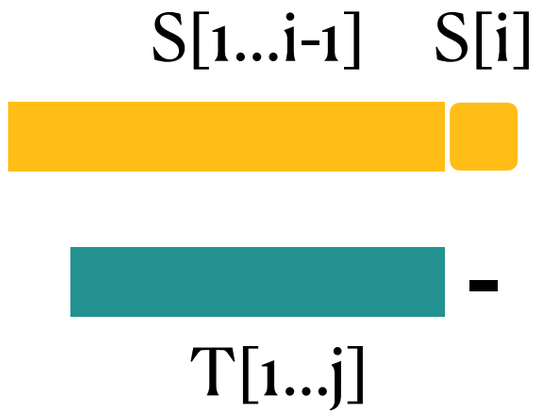
Dynamic Programming Algorithm

mn subproblems. $m = |S|$, $n = |T|$ $1 \le i \le m$, $1 \le j \le n$

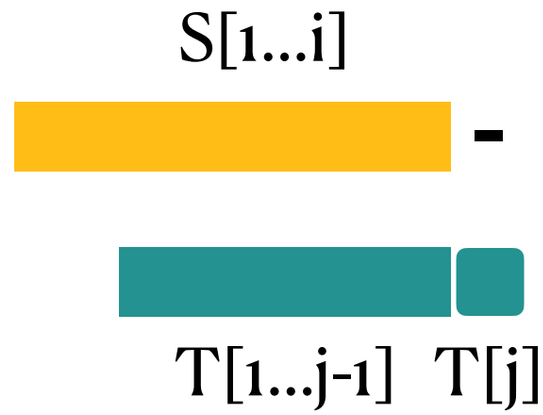
- Subproblem: OPT(i, j) as the edit distance of S[1...i] and T[1...j].
- Consider how S[i] and T[j] are aligned in the optimal alignment:



$OPT(i,j) = OPT(i-1, j-1) + cost(S[i], T[j])$



$OPT(i,j) = OPT(i-1, j) + 1$



$OPT(i,j) = OPT(i, j-1) + 1$

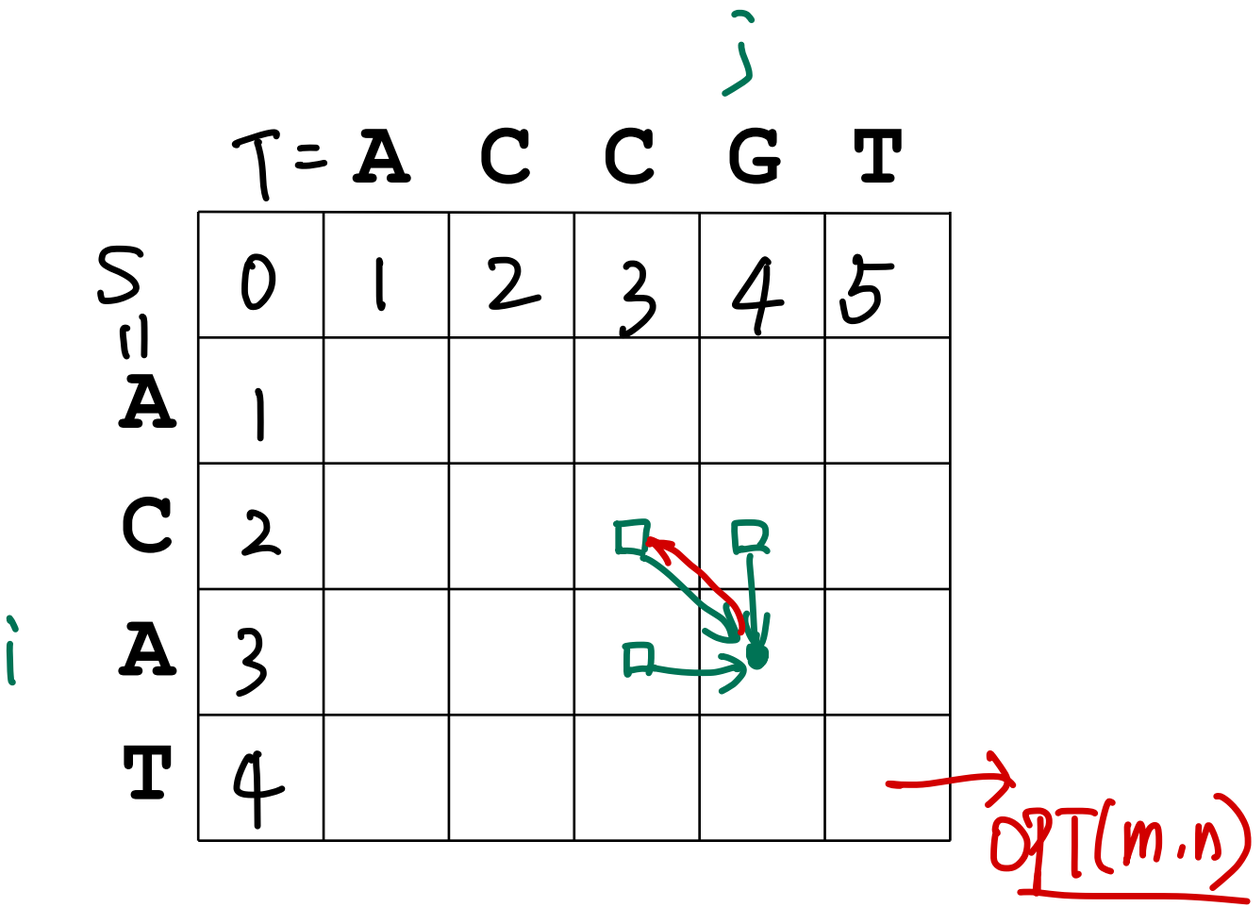
Recurrence

$$OPT(i,j) = \min \begin{cases} OPT(i-1,j-1) + cost(S[i], T[j]) \\ OPT(i-1,j) + 1 \\ OPT(i,j-1) + 1 \end{cases}$$

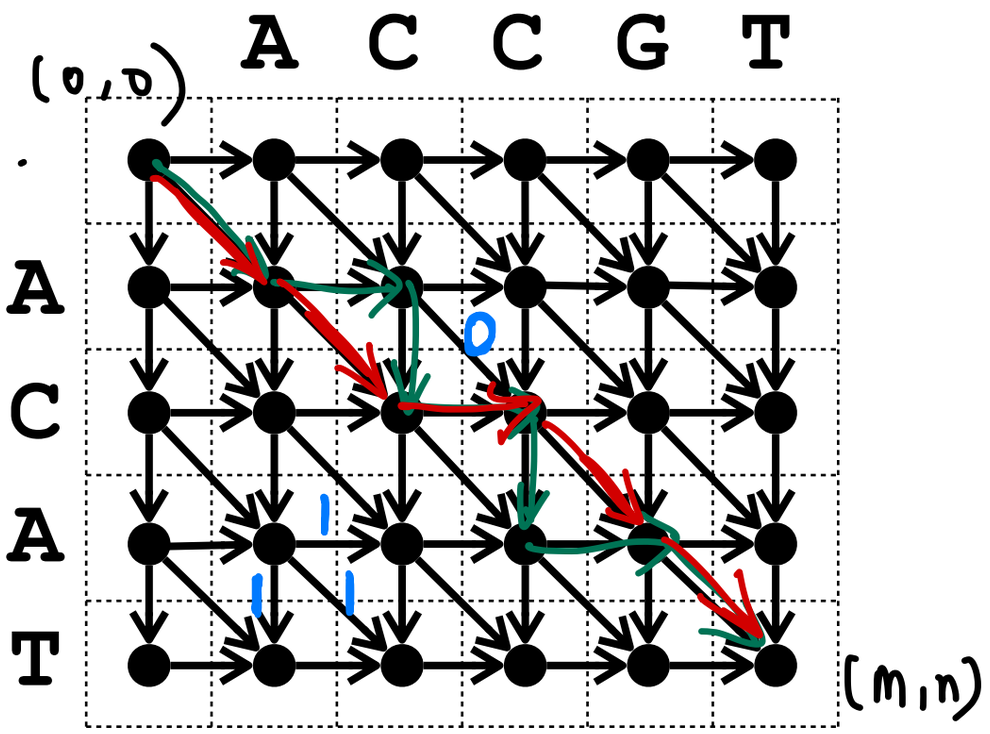
$$cost(S[i], T[j]) = \begin{cases} 1 \text{ if } S[i] \neq T[j] \\ 0 \text{ if } S[i] = T[j] \end{cases}$$

The Algorithm

- Step 1: Initialization
- Step 2: fill up the table following the recurrence
- Step 3: OPT[m,n] gives the edit distance of S and T
- Step 4: backtrace to find the optimal alignment



Connection with Shortest Path



- ~~Un~~directed acyclic graph (DAG), with mn vertices and O(mn) edges
- One-to-one correspondence between alignments and (o,o)-(m,n) paths
- cost(alignment) = length(path)
- Algorithm to find the (o,o)-(m,n) shortest path: O(E) = O(mn).

S = A - C - A - T

T = A C - C - G T

S = A C - A T

T A C C G T

Better Algorithms & Bounds

$m \approx n$

$n = \max\{|S|, |T|\}$

- Four-Russians algorithm: $O(n^2/\log n)$
- Meyer's algorithm (aka the wavefront algorithm): $O(nd)$, where d is the edit distance
- The edit distance cannot be computed in $O(n^{2-\epsilon})$ time, for any $\epsilon > 0$, unless the strong exponential time hypothesis is false.

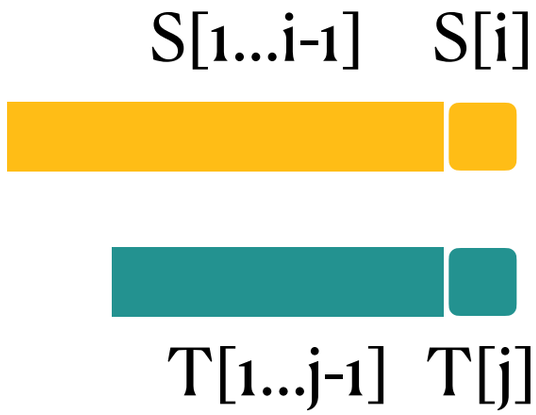
Alignment with Affine Gap

- They have the same cost under unit cost, but the 2nd alignment is more plausible.
- Affine gap cost: gap-open + k * gap-extension

20 k*3

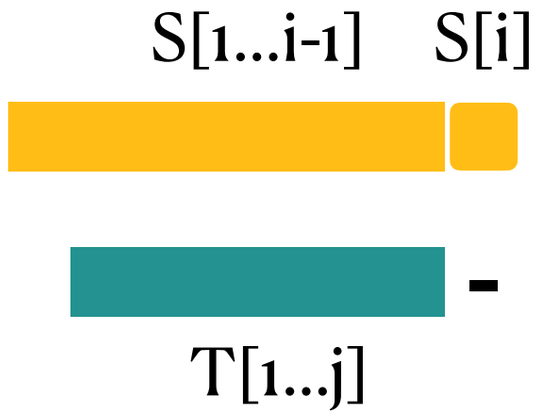
Does the Previous DP work?

- Define OPT(i, j) as the minimized cost of S[1...i] and T[1...j].
- Consider how S[i] and T[j] are aligned in the optimal alignment:



$OPT(i,j) =$

$OPT(i-1, j-1) + cost(S[i], T[j])$



$OPT(i,j) =$

$OPT(i-1, j)$