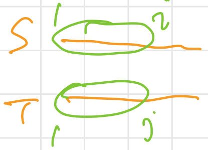
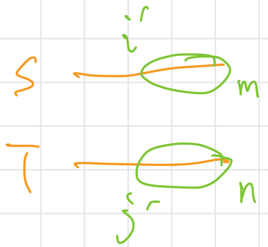
Review:

1. Affine gap penalty: $g(k) = a + b \cdot k$, $D_k[i,j]$

2. prefix alignment: $\max_{i,j} D[i,j]$, where $D[i,j]$ = optimal alignment score of $S[1..i]$, $T[1..j]$.

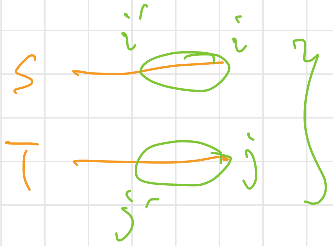


3. suffix alignment: To compute $\max_{i',j'} \text{score}(S[i'..m], T[j'..n])$

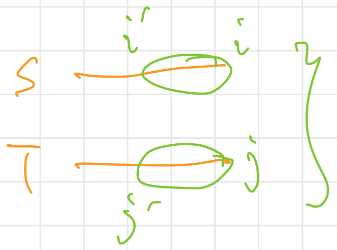


new definition

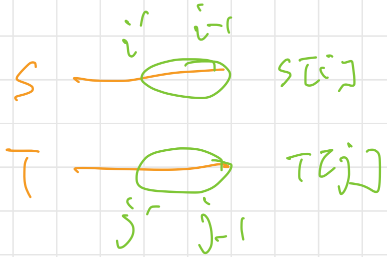
Define $D[i,j] = \max_{i',j'} \text{score}(S[i'..i], T[j'..j])$



D[i, j]

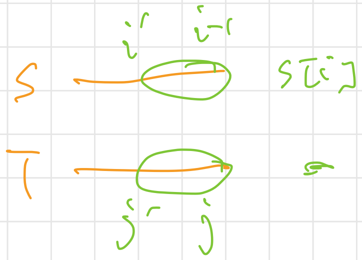


case 1



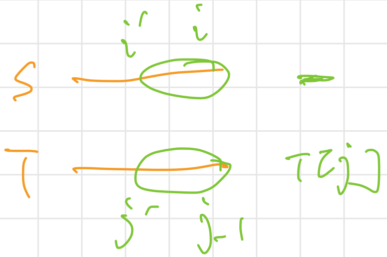
D[i-1, j-1] + f(S[i], T[j])

case 2

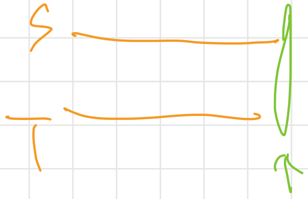


D[i-1, j] + indel

case 3



Case 4.



0

empty alignment.

Suffix Alignment Example

T

S

Match=1
Mismatch=-1
Indel=-1

		C	A	T	T	C
	0	0	0	0	0	0
A	0	0	1	0	0	0
T	0	0	0	2	1	0
T	0	0	0	1	3	2
G	0	0	0	0	2	2
A	0	0	0	0	1	1

ATTGA
C ATTC –
T

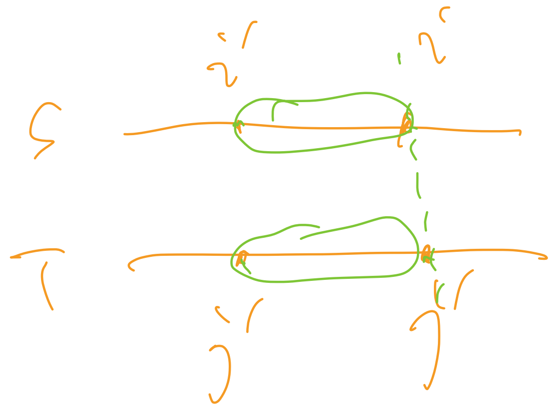
S[i']

–

A

Local Alignment

- Recall that for suffix alignment, D[i,j] denote the optimal "suffix alignment" alignment score of s[1..i], t[1..j]. I.e., D[i,j] is the maximum alignment score for s[i'..i] and t[j'..j] for all i' and j'.
- Therefore, optimal local alignment score is just $\max_{i,j}$ D[i,j].
- Algorithm:
  - Fill the dynamic progrmaming table is the same as suffix alignment.
  - Find (i,j) to maximize D[i,j], and backtrack from there.



$$D[i,j] = \max_{i',j'} score(S[i'..i], T[j'..j])$$

$$\max_{i,j}$$

Local Alignment Example

T[1..4]

S[1..3]

		C	A	T	T	C
	0	0	0	0	0	0
A	0	0	1	0	0	0
T	0	0	0	2	1	0
T	0	0	0	1	3	2
G	0	0	0	0	2	2
A	0	0	0	0	1	1

Match=1

Mismatch=-1

Indel=-1

① base case
② case 4, $=0$
③ $\max_{i,j}$

A Little History

- The algorithm was first proposed by Temple Smith and Michael Waterman in 1981. It works for both linear and affined gap penalty.
- It is known popularly as the Smith-Waterman algorithm.
- The global alignment algorithm was called the Needleman-Wunsch algorithm, which was published in 1970.

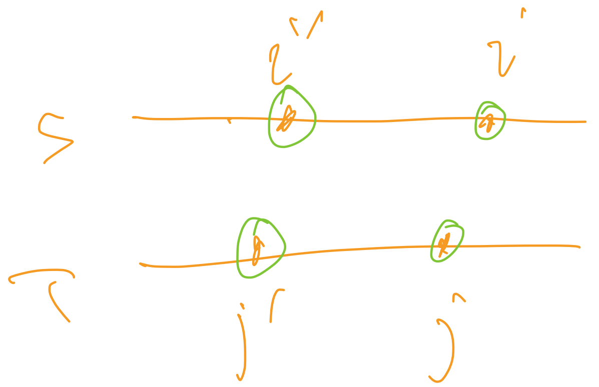
Temple Smith and Michael Waterman. (1981) “Identification of common molecular subsequences.” *Journal of molecular biology* 147(1): 195-197.

Saul Needleman and Christian Wunsch. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". *Journal of Molecular Biology*. **48**(3): 443–53.

Space Saving

$D[i,j] = (score, i', j')$

- Space complexity is O(mn).
- What if we only want the score?
- And the end positions?
- And the start positions?



maxScore
maxI
maxJ
maxI'
maxJ'

		C	A	T	T	C
	0	0	0	0	0	0
A	0	0	1	0	0	0
T	0	0	0	2	1	0
T	0	0	0	1	3	2
G	0	0	0	0	2	2
A	0	0	0	0	1	1

Affine Gap Local Alignment

$$D_0[i,j] = f(s[i], t[j]) + \max \begin{cases} D_0[i-1, j-1]; \\ D_1[i-1, j-1]; \\ D_2[i-1, j-1]; \\ 0 \end{cases}$$

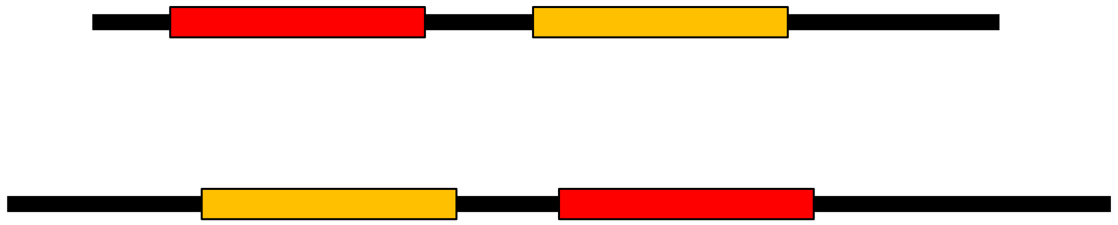
$$D_1[i,j] = \text{gapext} + \max \begin{cases} D_0[i, j-1] + \text{gapopen}; \\ D_1[i, j-1]; \\ D_2[i, j-1] + \text{gapopen}; \\ 0 \end{cases}$$

$$D_2[i,j] = \text{gapext} + \max \begin{cases} D_0[i-1, j] + \text{gapopen}; \\ D_1[i-1, j] + \text{gapopen}; \\ D_2[i-1, j]; \\ 0 \end{cases}$$

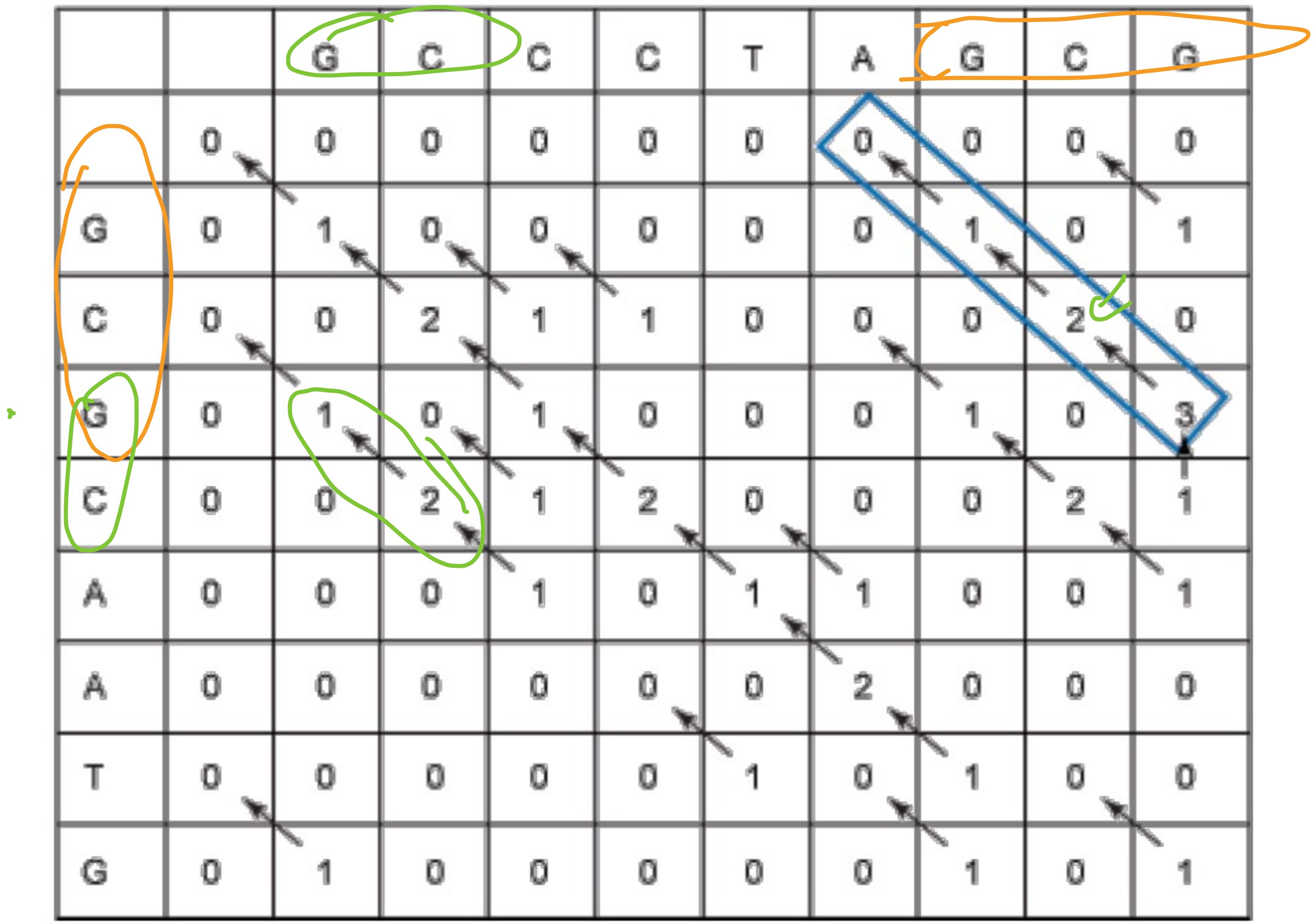
- Algorithm is as before, except that score is now lower bounded by 0.
- Afterward, find maximum element in all 3 tables, and backtrack until reaching a 0.

Compute Many Local Alignments

- It's sometimes useful to find many local alignments of S and T. E.g. when there are multiple similar regions between the two input strings.
- How?
- Time complexity?



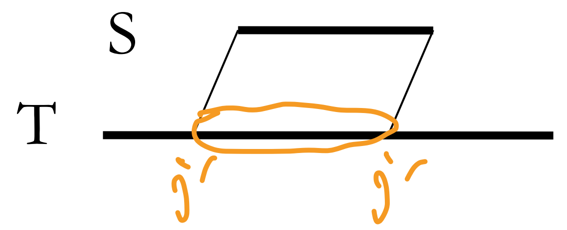
Local Alignment

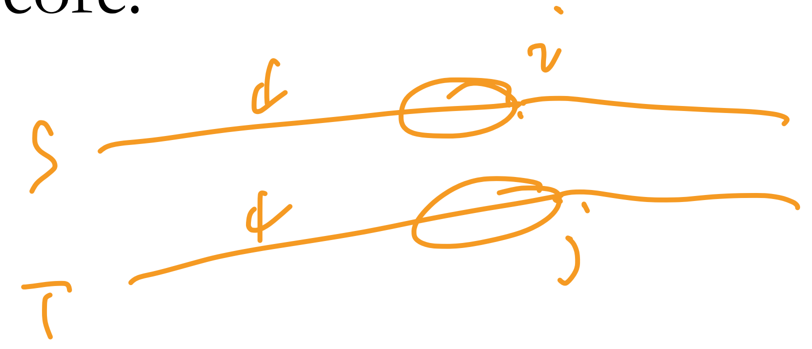


		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	0	0	0	0	0	1	0	1
C	0	0	2	1	1	0	0	0	2	0
G	0	1	0	1	0	0	0	1	0	3
C	0	0	2	1	2	0	0	0	2	1
A	0	0	0	1	0	1	1	0	0	1
A	0	0	0	0	0	0	2	0	0	0
T	0	0	0	0	0	1	0	1	0	0
G	0	1	0	0	0	0	0	1	0	1

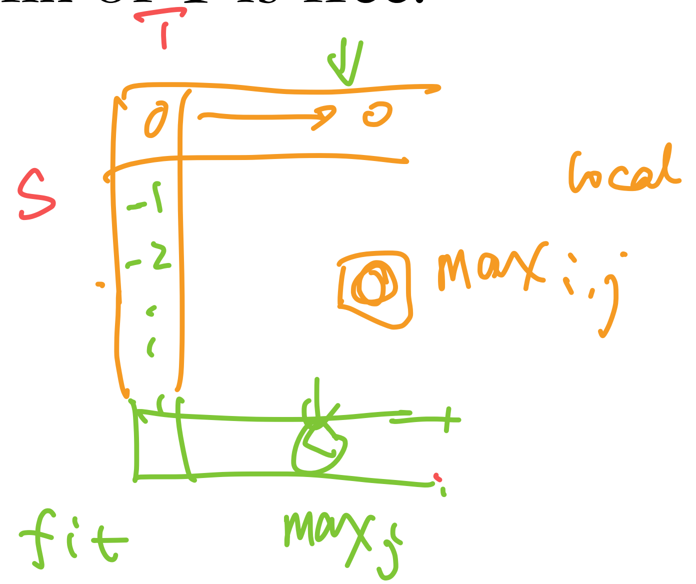
Fit Alignment

- Given sequence S and T. Find a global alignment between S and a substring of T, maximizing the alignment score.





- Deleting the prefix of T is free, deleting the suffix of T is free.
- How?
- Time Complexity?

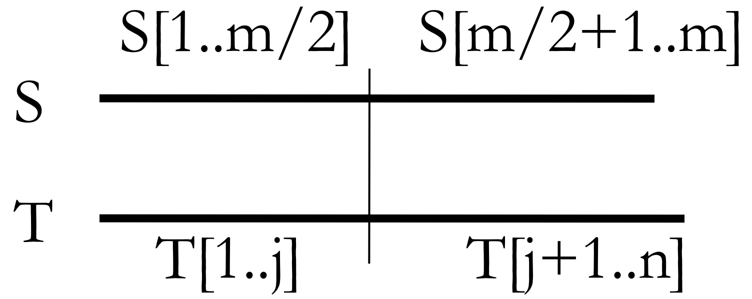


Linear Space Alignment

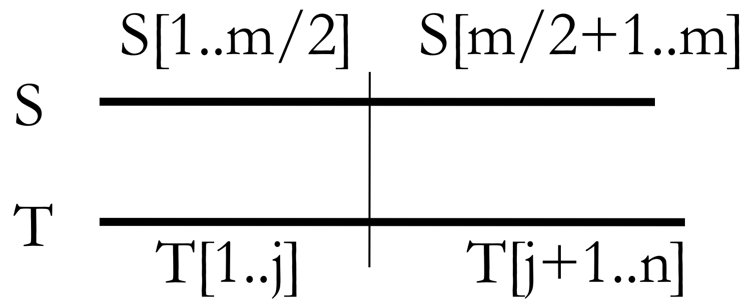
- Why linear space?
  - Computer RAM used to be very expensive in 80s.
  - "Prediction: The cost for 128 kilobytes of memory will fall below U$100 in the near future."
    - Creative Computing magazine. December 1981, page 6.
  - Even today, keeping everything in the L2 cache may speed up the computation.
- We have learned the linear space if only alignment score, instead of the alignment, is required.
- Let's now develop a linear space alignment. We focus on **global** alignment model first.

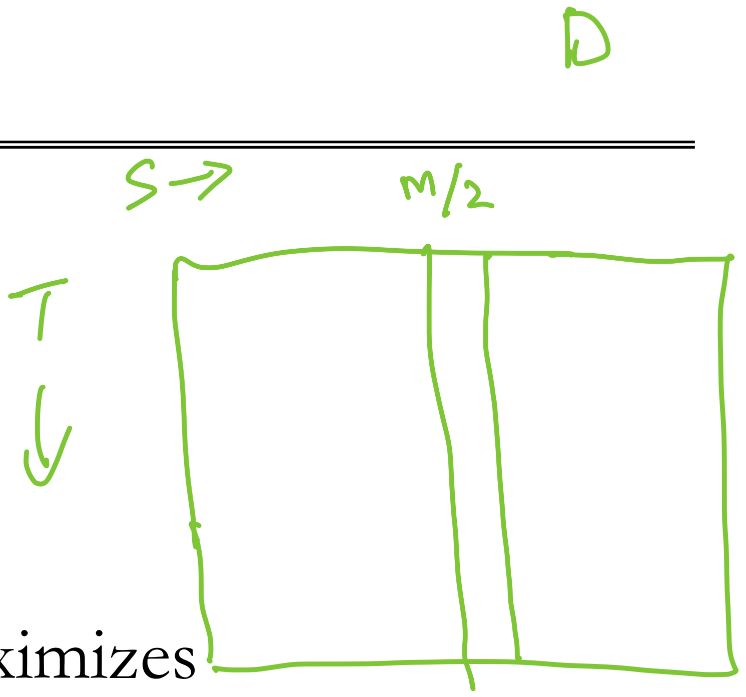
Divide and Conquer

- We want to find j such that the optimal alignment between S and T consists of two parts
  - S[1..m/2] aligns with T[1..j]
  - S[m/2+1..m] aligns with T[j+1..n]
- Then we can use divide and conquer.
- However, we need to comptue j in linear space.
- Note that there may be more than one j satisfying the condition. Any one of them will do the job.



Divide and Conquer





for j'

- Claim: j satisfies the desired condition iff it maximizes alignScore(S[1..m/2],T[1..j]) + alignScore(S[m/2+1..m],T[j+1,n])
- Let D be the dynamic programming table for aligning S and T, where do we find alignScore(S[1..m/2],T[1..j]) for every j?
- How about alignScore(S[m/2+1..m],T[j+1,n])?

Divide

S[1..m/2]

S: CAT TCA

T: AT TGA

S[1..m/2]

	C	A	T
	0	-1	-2	-3
A	-1	-1	0	-1
T	-2	-2	-1	1
T	-3	-3	-2	0
G	-4	-4	-3	-1
A	-5	-5	-4	-2

T

S[m/2+1..m]

-1	0	1	-5	A
0	-2	-2	-4	T
1	-1	-1	-3	T
-1	0	0	-2	G
-1	0	1	-1	A
-3	-2	-1	0	
T	C	A		

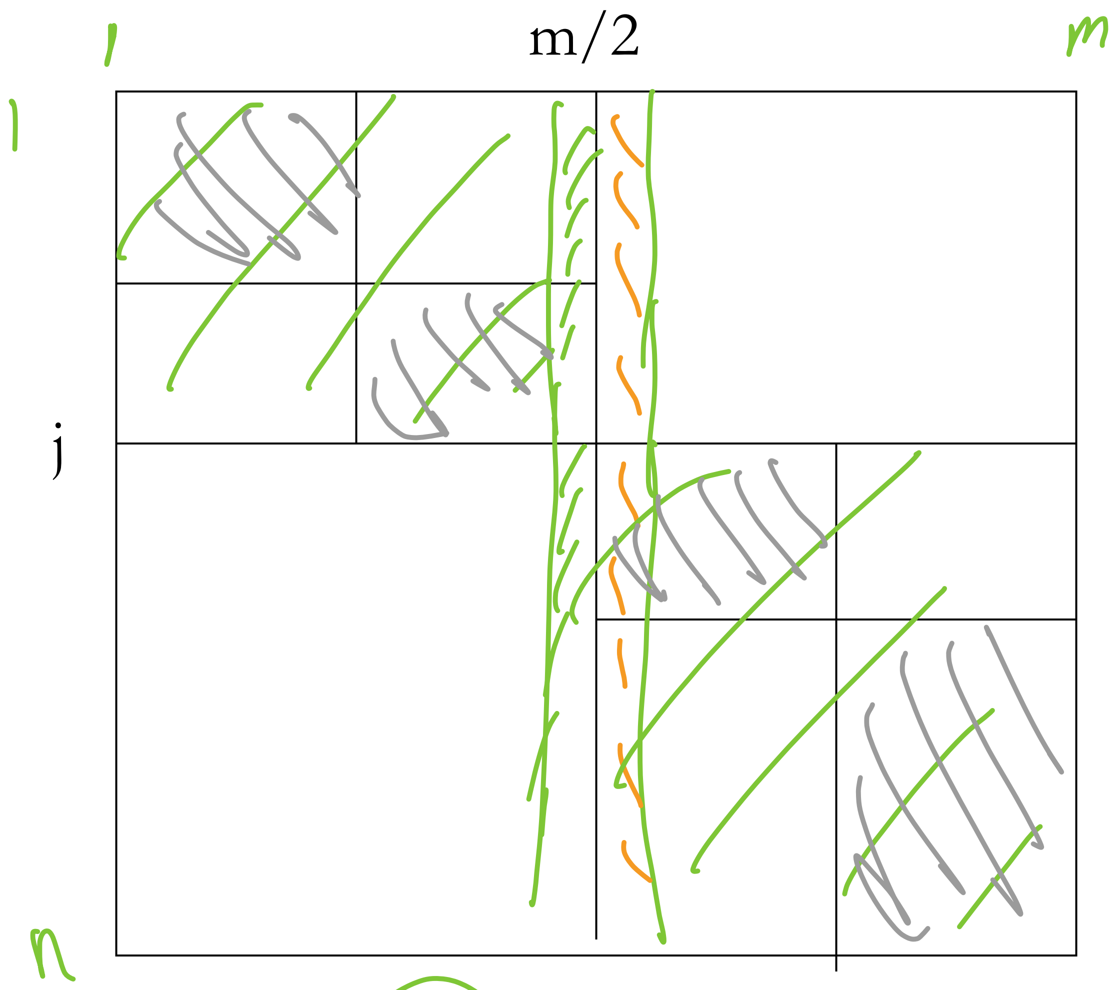
Computing the two center columns requires linear space.

Algorithm

Algorithm Align(S,T):

1. If |S| = 1, return a trivial alignment.
2. Use the previous idea to find j that maximizes alignScore(S[1..m/2], T[1..j]) + alignScore(S[m/2..m], T[j+1..n])
3. Concatenate Align(S[1..m/2], T[1..j]) and Align(S[m/2..m], T[j+1..n]) and return it.

Time Complexity

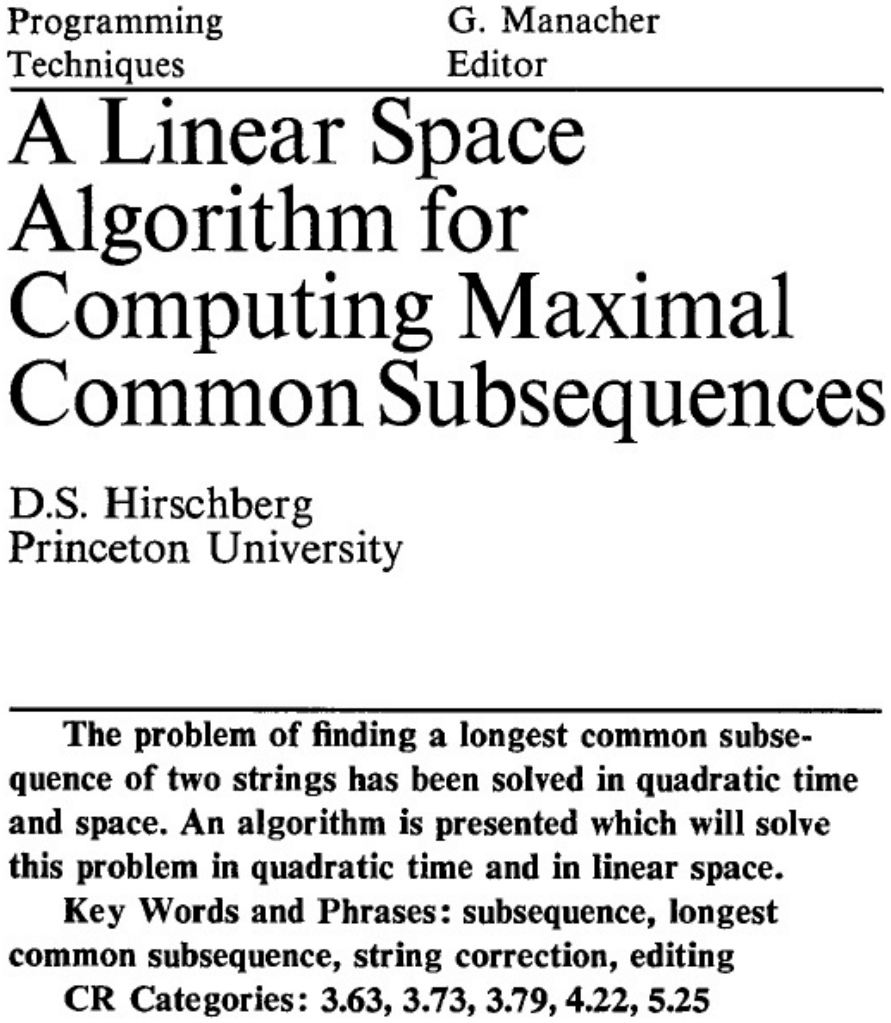


$mn + \frac{m \cdot n}{2} + \frac{m \cdot n}{4} + \cdots$

$\leq 2mn.$

- $T(m,n) \leq \underline{mn} + \underline{T(m/2,j)} + \underline{T(m/2,n\text{-}j)} \leq \underline{2\ mn}$

D.S. Hirschberg, A linear space algorithm for computing longest common subsequences, Comm. ***ACM*** 18 (1975) 341-343.



Programming Techniques — G. Manacher, Editor

A Linear Space Algorithm for Computing Maximal Common Subsequences

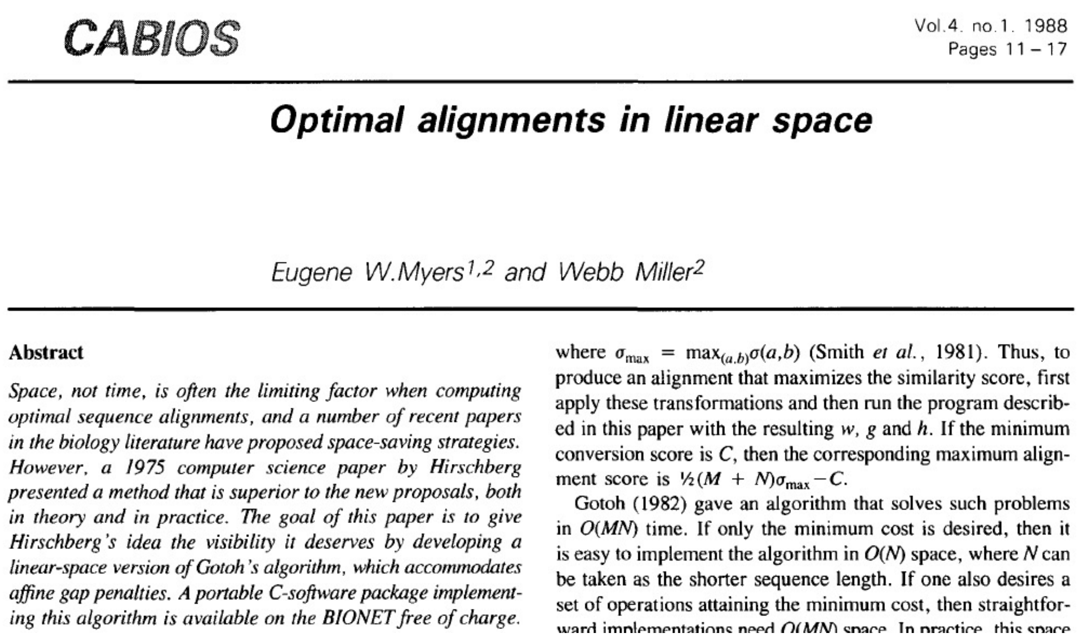
D.S. Hirschberg
Princeton University

The problem of finding a longest common subsequence of two strings has been solved in quadratic time and space. An algorithm is presented which will solve this problem in quadratic time and in linear space.

Key Words and Phrases: subsequence, longest common subsequence, string correction, editing

CR Categories: 3.63, 3.73, 3.79, 4.22, 5.25

Linear Space Affine Gap Penalty



CABIOS Vol.4. no.1. 1988 Pages 11 – 17

Optimal alignments in linear space

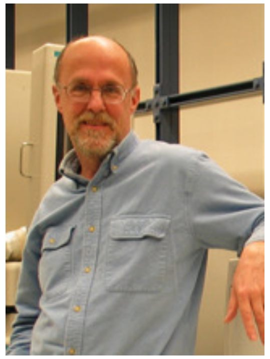
Eugene W.Myers[1,2] and Webb Miller[2]

Abstract

Space, not time, is often the limiting factor when computing optimal sequence alignments, and a number of recent papers in the biology literature have proposed space-saving strategies. However, a 1975 computer science paper by Hirschberg presented a method that is superior to the new proposals, both in theory and in practice. The goal of this paper is to give Hirschberg's idea the visibility it deserves by developing a linear-space version of Gotoh's algorithm, which accommodates affine gap penalties. A portable C-software package implementing this algorithm is available on the BIONET free of charge.

where $\sigma_{\max} = \max_{(a,b)}\sigma(a,b)$ (Smith *et al.*, 1981). Thus, to produce an alignment that maximizes the similarity score, first apply these transformations and then run the program described in this paper with the resulting *w*, *g* and *h*. If the minimum conversion score is *C*, then the corresponding maximum alignment score is $\frac{1}{2}(M + N)\sigma_{\max} - C$.

Gotoh (1982) gave an algorithm that solves such problems in $O(MN)$ time. If only the minimum cost is desired, then it is easy to implement the algorithm in $O(N)$ space, where N can be taken as the shorter sequence length. If one also desires a set of operations attaining the minimum cost, then straightforward implementations need $O(MN)$ space. In practice, this space

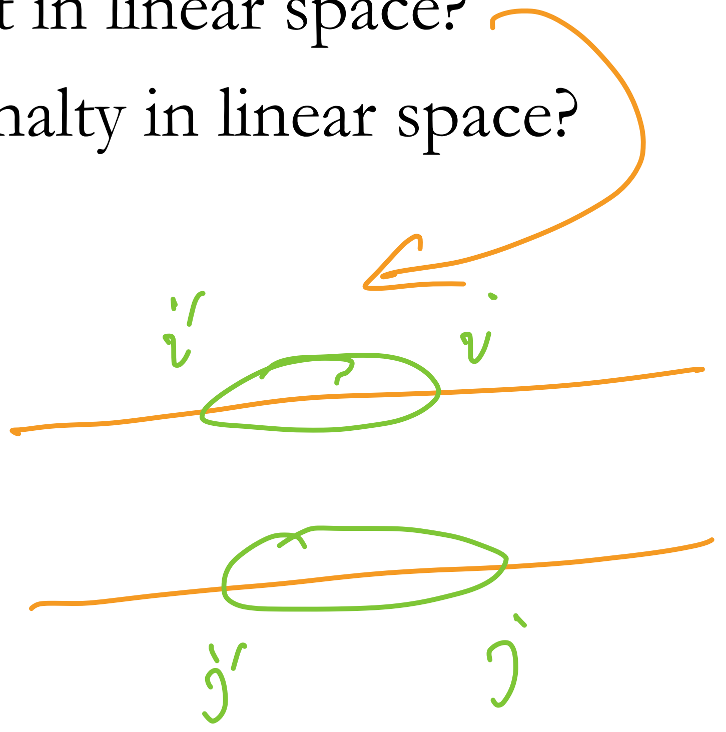


"The goal of this paper is to give Hirschberg's idea the visibility it deserves by developing a linear-space version of Gotoh's algorithm."

Question

How to do local alignment in linear space?

How to do affined gap penalty in linear space?



reduction