Sequence Alignment & Similarity Search.

Motivations:

1. A new gene ⟶ what is its function?
   Compare with the database of known genes & based on similarity, formulate hypothesis.

   e.g. 1984 ⟶ newly discovered cancer-causing (resides in a virus) v-sis oncogene, with all known genes & found a close match with a gene involved in growth & development of cell.

2. A ~~part~~ segment of chromosome identified ~~for causing~~ with certain function ⟶ what gene resides there? ⇒ Compare sequenced segment with the gene database

   e.g. cystic fibrosis gene in chromosome 7 (human) in 1989. ⟶ matches with a gene responsible for coding ATP binding proteins.

(11) DNA "sequence similarity" is more than a "string matching" problem.

→ Evolutionary processes: Mutation, replication error causes insertion, deletion & substitution.

→ Exon - intron chains.

e.g. ATATAT → Similar from the angle of
TATATA → genome analysis.

(*) Edit distance: (Vladimir Levenshtein, 1966)

"Edit distance" between two strings is the "minimum no." of "editing operations" needed to transform from one string into another, where the edit operations are insertion of a symbol, deletion of a symbol & substitution of one symbol for another.

e.g. TGCATAT
↓ delete last T
TGCATA delete last A
↓
TGCAT
↓ insert A at the front
ATGCAT
↓ substitute G for G in the 3rd. pos.
ATCCAT
↓ insert a G before the last A
ATCCGAT

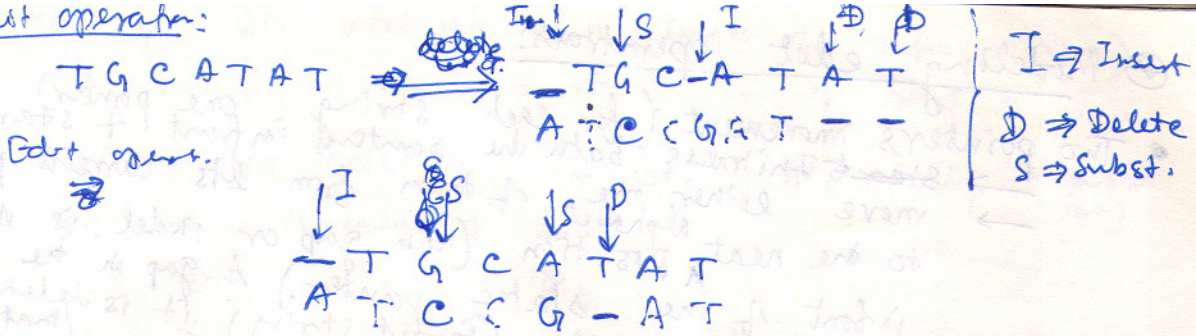
∴ no. (TGCATAT, ATCCGAT) = 5

TGCATAT
↓ insert A at the front
ATGCATAT
↓ delete T in the 6th pos.
ATGCAAT
↓ subst. G for A in the 5th
ATGCGAT
↓ substitute C for G in the third
ATCCGAT.

no. dist (TGCATAT, ATCCGAT) = 4

∴ edit dist = 4.

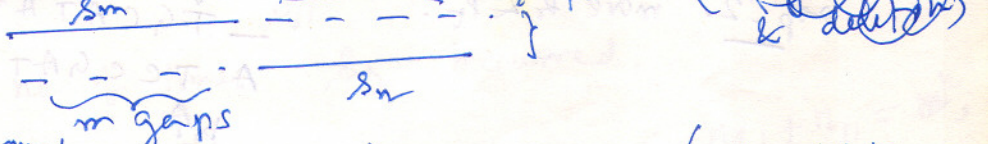
→ 5 Edit operations:



→ 4 - Edit oper.

⇒ String matching problem with insertion (indels) of gaps in any of the strings (with the constraint that simultaneous insertions at any position are not allowed) and also substitution of symbols. (a cost factor is generally associated with all these operations).

- Given m - string & n - string, $n > m$; maximum possible edit operations ⇒ $m+n$.



- min. edit operations $= (n-m)$, $n > m$. (only deletions & subst.)

$\therefore (n-m) \leq$ edit-distance $(S_m, S_n) \leq m+n$ (only D & I)

Tighter bound ↳ max. (m, n). ⇓ However it subst. is allowed.

⊛ Modeling edit operations:

- Two pointers movement (for each string one pointer)
  - → ~~Start~~ Initialise both the pointers in front of starting alphabet
  - → move either one of them from its current position to the next alphabet position (as gap or indel is introduced in front of the static pointer.) A gap in the reference string is insertion else (target string) it is deletion
  - → move both the pointers (substitution / matching operation)
  - → Stop when both the pointers arrived at the last alphabet.

Indicated posn.

e.g. Ref: p_r ↑ T G C A T A T

Target: p_t ↑ A T C C G A T

		Edit operation
→ Step 1. move p_t:	p_r ↓ _ T G C A T A T A T C C G A T ↑ p_t	I
Step 2. move p_r & p_t:	p_r ↓ _ T G C A T A T A T C C G A T ↑ p_t	M
Step 3: move p_r & p_t:	_ T G C A T A T A T C C G A T	S,
Step 4: — do —		M,
Step 5: — do —		S,
Step 6 move p_r	_ T G C A T A T A T C C G - A T	D
Step 7 & 8. move p_r & p_t:	_ T G C A T A T A T C C G - A T	M, M.

∴ Total no. of edit operations I + 4M + 2S + D

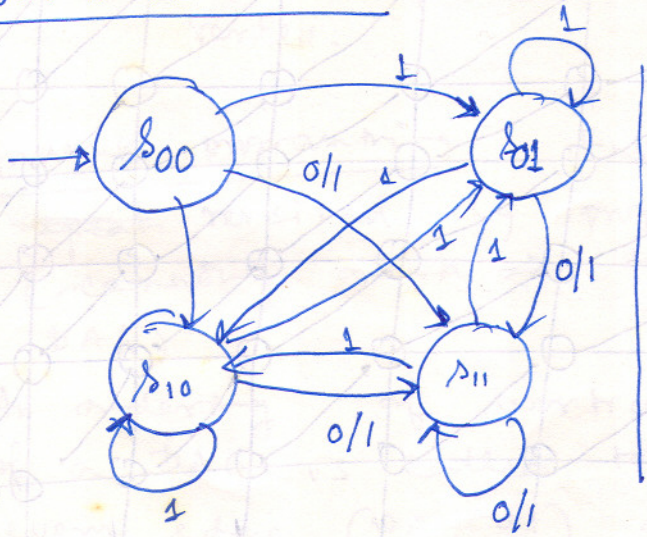
⇒ 4, M = 0.

N.B. (1) Movement is allowed from one alphabet to the next (jumping models)

(ii) No. of $\downarrow P_r$ movements = n (length of ref. string)
No. of $\uparrow P_t$ " = m (" of target ")

(iii) 00 ⇒ No movement of $\downarrow P_r$ & $\uparrow P_t$ (only at initial stage)
01 ⇒ Movement of $\downarrow P_r$
10 ⇒ " " $\uparrow P_t$
11 ⇒ " " $\downarrow P_r$ & $\uparrow P_t$

A State transition model:



0/1 → Cost for matching 0 else 1.

stop after when $\sum 1 = n+m$

n_{01} + ⊗m

Let n_{ij} = no. of times S_{ij} occurred.

$\therefore n_{01} + n_{10} + 2 \cdot n_{11} = m+n$; $n_{01} + n_{11} = n$; $n_{10} + n_{11} = m$;

$n_{11} \leq \min(m, n)$;

(~~No. of integer solutions of the above eqn. gives the no. of valid sequences~~) ✗

⇒ ~~min~~ constraints: $n_{01} + n_{11} = n$;
$n_{10} + n_{11} = m$;
$n_{01}, n_{10}, n_{11} \geq 0$;

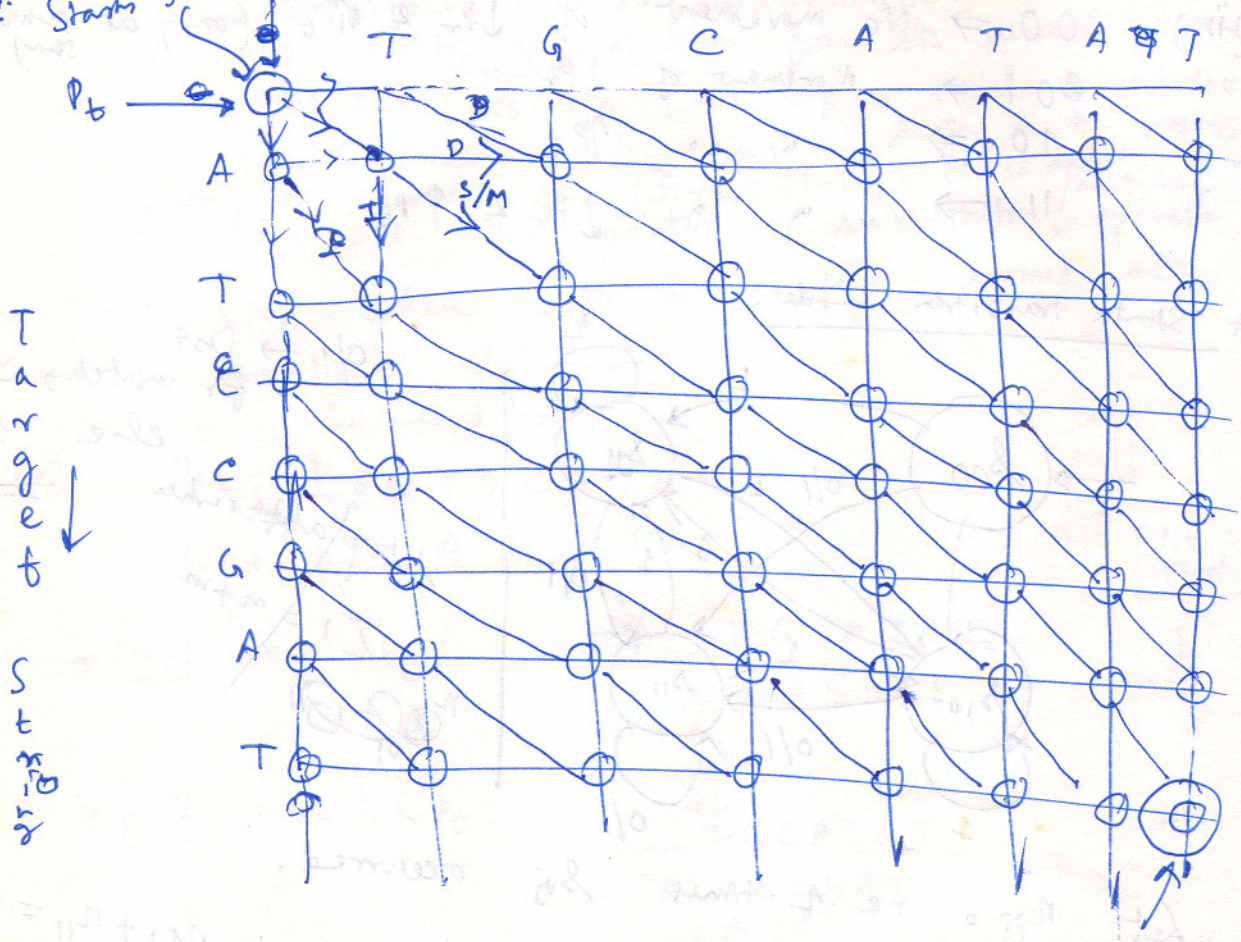
Context free transitions are not able to ~~capture the~~ accept as valid string.

It is quite difficult to ~~consider~~ enumerate all possible valid sequences & get the minimum out of it. So the states should be defined from the context.

Context Sensitive FSM: (Position of ↓Pr & ↑Pt define a State).

Ref. Sm

e.g. Starts State, Pr



Transition are possible only to right neighbour/ bottom / diagonal (R/B) neighbour for D/I/S or M operation.

Stop state

⇒ Computation of edit distance is equivalent to computation of shortest path from the starting state to the stop state following the neighborhood definition.

⇒ Directed Acyclic Graph (DAG).

Alg.: ⓪ Start from the starting vertex and initialize distances to its neighbours vertex.

① For every vertex, u,
if all its predecessors are visited, mark it visited & compute its dist. from the source as min. { dist. (Predecessor, u) + cost } & also store the predecessor into.

② Perform step 1 iteratively till you reach destination vertex.

Dynamic programming: Ensure no vertex is revisited by proper traversal order (scanning order ⇒ topological sorted ordering for a DAG)

An ordering of vertices $v_1, v_2, \ldots, v_n$ of a DAG is called topological if every edge (v_i, v_j) of the DAG connects a vertex with a smaller index to a vertex with a larger index, i.e., $i < j$.

→ In this case row-wise scanning / column-wise scanning / zigzag scanning should be o.k. ⇒ $O(m.n)$

→ For every vertex store also the predecessor which is the best in the shortest path from the source. This helps in retracing the path back.

→ If weights are inversely related to dist., then longest path will provide the solution (maximisation problem) ⇒ Used for sequence alignment.

* Longest common subsequence problem:

A subsequence of a string v is simply an (ordered) sequence of alphabets (not necessarily consecutive) from v.

e.g. $v = ATTGCTA$

AGCA, ATTA $\in$ set of subsequences of v

but TGTT, TCG $\notin$ " "

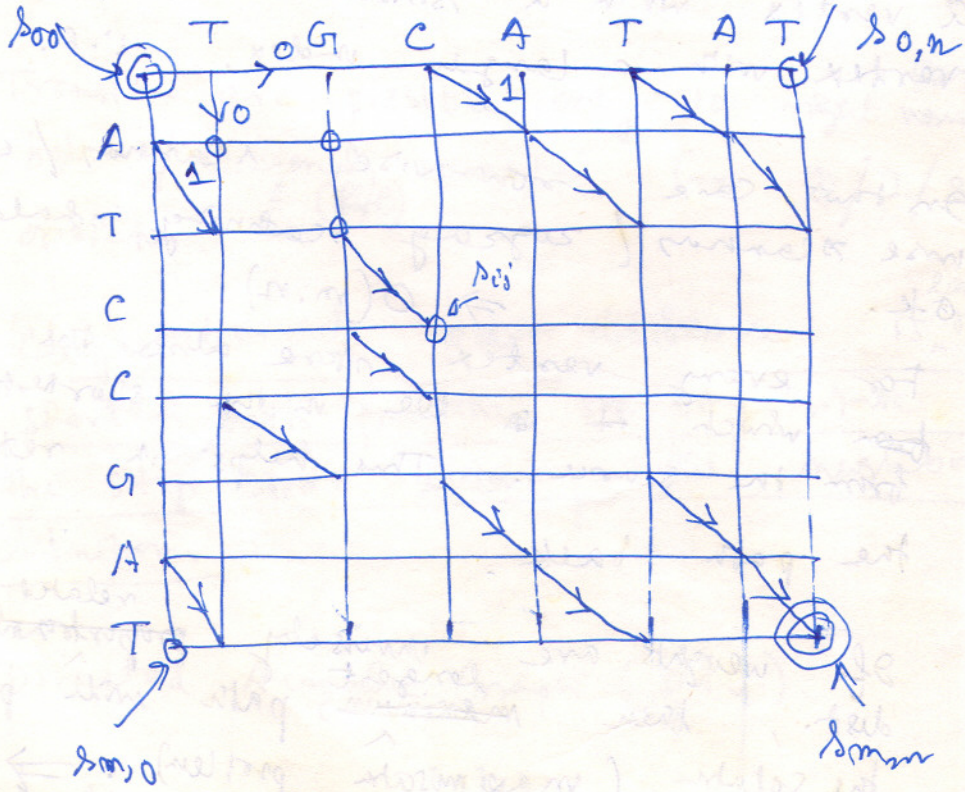
Problem Statement:

Let $v = v_1 v_2 \cdots v_n$ & $w = w_1 w_2 \cdots w_m$ be two strings. Find the longest common subsequence $u = u_1 u_2 \cdots u_k$ of v & w,

s.t. $u_t = v_{i_t} = w_{j_t}$

& $1 \le i_1 < i_2 < \cdots < n$, $1 \le j_1 < j_2 < \cdots < m$.

$\Rightarrow$ No mutation or substitution allowed.



$$\begin{cases} s_{i,0} = s_{0,j} = 0, & 1 \le i \le m, \quad 1 \le j \le n \\ s_{i,j} = \max \begin{cases} s_{i-1,j} + 0 \\ s_{i,j-1} + 0 \\ s_{i-1,j-1} + 1, & \text{if } v_i = w_j \end{cases} \end{cases}$$

N.B. ⓐ store back tracking pointers ←, ↑, or ↖ for every vertex.

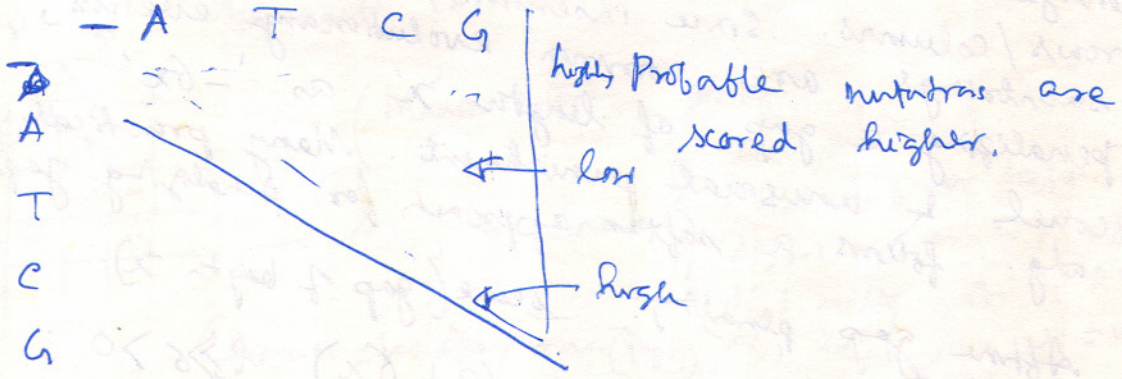
ⓑ Computation of edit distance:

$$d_{i,0} = i; \quad d_{0,j} = j; \quad 1 \le i \le m, \ 1 \le j \le n;$$

$$d_{i,j} = \min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + 0 & \text{if } v_i = w_j \end{cases}$$

* Global Sequence Alignment: Needleman & Wunsch (1970)
(The use of context sensitive FSM with all transitions).

scoring matrix ($\delta(\ )$)



For protein comparison 21×21 scoring matrix:

PAM (Point accepted Mutation) →

BLOSUM (Block substitution) Matrix →

$$S_{i,j} = \max \begin{cases} S_{i-1,j} + \delta(v_i, -) \\ S_{i,j-1} + \delta(-, w_j) \\ S_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

If, $\delta(v_i, -) = \delta(-, w_j) = -\sigma$ [Insertion & deletion

$\delta(v_i, w_j) = -\mu$ (Mutation), for $v_i \neq w_j$

$= 1$ (matches).

$$\therefore S_{i,j} = \max \begin{cases} S_{i-1,j} - \sigma \\ S_{i,j-1} - \sigma \\ S_{i-1,j-1} - \mu, & \text{if } v_i \neq w_j \\ S_{i-1,j-1} + 1, & \text{if } v_i = w_j \end{cases}$$

∴ Resulting score = no. of matches $- \mu *$ no. of mismatches $- \sigma *$ no. of insertion & deletion.

◎ Alignment with 'Gap' penalties:

A gap in an alignment is defined as a contiguous sequence of spaces (indels) in one of the rows / columns. Since insertion and deletions of substrings are common evolutionary events, penalizing a gap of length 'x' as '$-\sigma x$' is cruel & unusual punishment. Many practical alg. follows a softer approach for penalizing gaps.

Affine gap penalty: score(gap of length x)

$= -(\rho + \sigma x)$, $\rho \gg \sigma > 0$;

$$\downarrow S_{i,j} = \max \begin{cases} \downarrow S_{i-1,j} - \sigma \\ S_{i-1,j} - (\rho + \sigma) \end{cases}$$

$$\rightarrow S_{i,j} = \max \begin{cases} \rightarrow S_{i,j-1} - \sigma \\ S_{i-1,j} - (\rho + \sigma) \end{cases}$$

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) \\ \downarrow \\ s_{i,j} \\ \rightarrow \\ s_{i,j} \end{cases}$$

N.B.
[For every $(i,j)^{th}$ vertex maintain $s_{v_{i,j}}$, $s_{i,j}$ & $\vec{s}_{v_{i,j}}$ separately].

* Local Sequence Alignment problem: Smith-Waterman (1981)

Problem statement: Given strings v & w and a scoring matrix δ, compute substrings of v & w whose global alignment, as defined by δ, is maximal among all global alignments of all substrings of v & w.

Model: Additional edges from the starting state (vertex) to each state (vertex) with "zero" cost & also edges from each vertex to the destination vertex.

(Maximisation problem)

$$\therefore s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} - \delta \text{ or } s_{i,j} + \delta(v_i, -) \\ s_{i,j-1} - \delta \text{ or } s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} - \text{neg}, & \text{if } v_i \neq w_j \\ s_{i-1,j-1} + 1 \left(\searrow \delta(v_i, w_j) \right), & \text{if } v_i = w_j \end{cases}$$

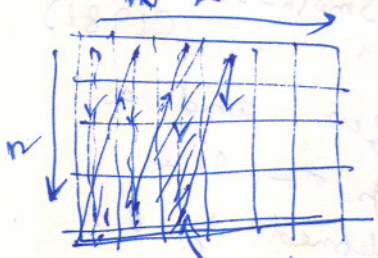
Smith-Waterman (1980).

✱ Space Efficient Alignment Algo.:

Divide & Conquer Strategy:

→ Usual Dynamic Algo.: Time complexity: $O(nm)$ ∝ no. of edges.
Storage " : $O(nm)$ ∝ no. of vertices,
[For storing back-back pointers].

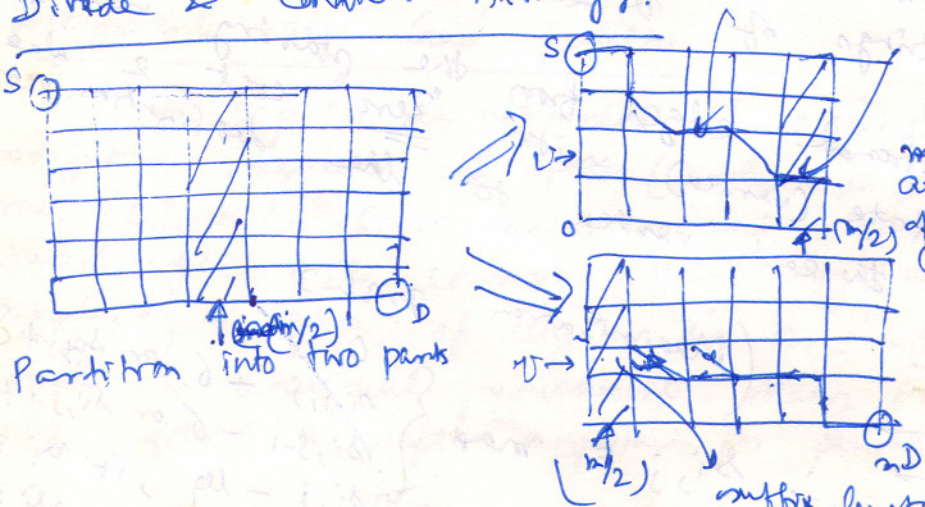
→ However, if only 'score' is required (not the aligned path), $O(n$ or $m)$ is sufficient. Store only scores of previous row or column during scanning.



Time complexity: $O(nm)$
Space " : $O(n)$
} When only maximal score is required (or edit distance is required).

remember only scores at these vertices for computing scores at next column.

→ Divide & Conquer strategy:



∴ maximal length $(S, D) = \max_{\forall i} (\text{Pref\_length}(i, m/2) + \text{suffix\_length}(i, m/2))$

& the vertex $(i, m/2)$ will be on the path for which it is max.

i.e. $i = \text{argmax}(\quad)$

Computation requirement: Time: $O(n \cdot m/2) + O(n \cdot m/2) \approx O(mn)$
Storage: $O(n) + O(n)$.

6 Four Russian Speed up: (Proposed by Vladimir Arlazov, Effim Dinic, Michael Kronrod & Igor Faradzev in 1970 applied to sequence comparison by William Masek & Michael Paterson in 1980).

Store all optimal paths & score for all possible combination of 't' length strings: (Storage space: $4^t \times 4^t$) ⇒ for nucleotides

Let, $t = \frac{\log n}{4}$,

$\therefore n = 2^{4t} = 4^{2t}$.

$\therefore 4^t \times 4^t = n^{\frac{1}{2}} \times n^{\frac{1}{2}} = n$.

∴ Storage space requirement : O(n) for storage of all optimal paths.

Thus makes computation of $\beta_{v,j,i,j'} \Rightarrow O(2t) \Rightarrow O(\log n)$ for access of 't' strings

∴ Computation of all $\beta_{v,j} \Rightarrow O\left(\frac{n}{t} \times \frac{n}{t}\right) \times \log n$

$\Rightarrow O\left(\frac{n^2}{t^2}\right) \times \log n)$.

$\Rightarrow O\left(\frac{n^2 (\log n)}{(\log n)^2}\right)$

$\Rightarrow O\left(\frac{n^2}{\log n}\right)$.

⇒ Discuss (i) Exon chaining problem
(ii) Splicing based Alignment.

* Heuristic Similarity Search Techniques:

→ Dynamic Algorithm: $O(n^2)$ not suitable for searching through large genome database (10^{10} nucleotides) & query string ($\sim 10^3$)

→ Starting in the early 1990's biologists had no choice but to use fast heuristics as an alternative to quadratic sequence algorithm.

→ D. Lipman & W.R. Pearson (1985) → FASTA

→ S. Altschul, W. Gish, W. Miller, E. Myers and J. Lipman (1990) → Basic Local Alignment Search Tool (BLAST)

↳ Modified in 1997.

→ l-mer filtration technique for approximate pattern matching (with k-mismatches)

If an n-letter substring of a query approximately matches an n-letter substring of the text, then the two substrings share at least one l-mer for a sufficiently large value of l.

Th.: If the strings $x_1, \ldots, x_n$ & $y_1, \ldots, y_n$ match with at most k mismatches, then they share an l-mer for $l = \lfloor \frac{n}{k+1} \rfloor$, i.e. $x_{i+1} \cdots x_{i+l} = y_{i+1}, \ldots y_{i+l}$ for some $1 \le i \le n-l+1$

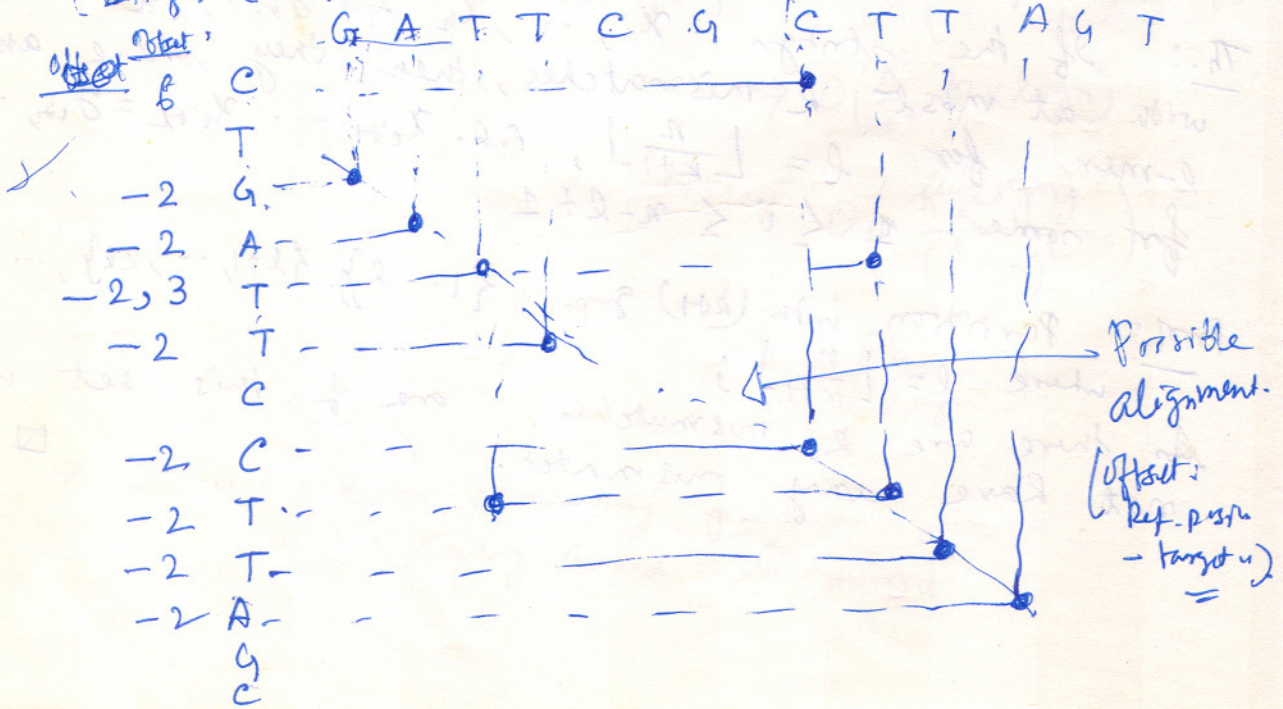
Proof: Partition into $(k+1)$ grps., $\{1 \ldots l\}, \{l+1, \ldots, 2l\}, \ldots$ where $l = \lfloor \frac{n}{k+1} \rfloor$;

As there are k mismatches, one of this set will not have any mismatch. □

This theorem motivates the following l-mer filtration algorithm for query matching with k-mismatches:

- Potential match detection: Find all matches of l-mers in both the query & the text for $l = \lfloor \frac{n}{k+1} \rfloor$. [Use of hashing or suffix tree]
- Potential match verification: Verify each potential match by extending it to the left & to the right until the first $k+1$ mismatches are found or the beginning/end of the query/text is found.

- FASTA:
  - → FASTA search begins by breaking the search sequence into words (for nucleotides words of length 4 to 6, for proteins 1 to 2).
  - → Find the positions of l-mer matches & represent in the form of dot matrices. (for $l=2$):



→ From this dot matrix, it selects some diagonal with a high concentration of 1's and groups these runs of diagonal 1's into longer runs. [~~Dyn~~ Eg Local sequence alignment on these segments by dynamic programming).

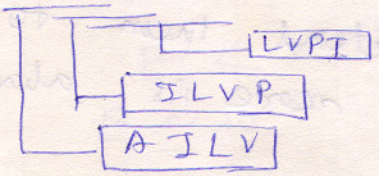
→ 'dots' are stored as offsets from the ref positions. - Adjacent ~~Alphabets~~ with similar offsets are grouped together.

- **BLAST :**

Input sequence: AILVPTV

1. Break the query sequence into words.

AILVPTVIGCT



select these words which are biologically more relevant e.g. sequence (words) with frequently observed amino acids are discarded.

2. Search for exact word matches (also called high-scoring pairs or HSPs) in the database sequence.

AILV PTV

MVQ GWA LYD FLK CR AILV GTV IA ML ...

(in both directions)

3. Extend the match - until the local alignment score falls below a fixed threshold (the most recent version of BLAST allows gaps in the extended match.

→ For any two l-mers $x_1 \ldots x_l$ & $y_1 \ldots y_l$, BLAST defines the segment score as $\sum_{i=1}^{l} \delta(x, y)$, $\delta(x, y)$ is the similarity score between x & y.

→ A segment pair is locally maximal if its score cannot be improved by shifting it to any direction. BLAST attempts to find all locally maximal segment pairs in the query & database sequences with scores above some set threshold. If the threshold is high enough, then the set of all l-mers that have score above a threshold is not too large. This becomes a multiple pattern matching problem (use of keyword tree / suffix tree).

→ After the potential matches are located, BLAST attempts to extend them to see whether the resulting score is above the threshold.

→ The choice of the threshold in BLAST is guided by the Altschul – Dembo – Karlin statistics.

$E(\theta)$ = Number of matches with scores above θ

$= K.m.n.e^{-\lambda\theta}$; $m, n \Rightarrow$ length of two strings.

$K \Rightarrow$ constant

$\lambda \Rightarrow$ Parameter

λ is a true root of the eqn.:

$$\sum_{x,y \in A} p_x p_y e^{\lambda \delta(x,y)} = 1$$

$p_x, p_y \Rightarrow$ frequencies of x & y.

The probability that there is a match of score greater than θ betn. two "random" sequences of length n & m is computed as

$1 - e^{-E(\theta)}$. (Is it $1 - e^{-E(\theta)}$).

* Alignment Scores and Statistical Significance of Database Searches :

E-score(S): expected no. of sequences of score $>= S$, would have been found randomly

P-score(S): prob. that one or more sequences of score $>= S$ would have been found randomly.

Low values of E & P indicate the search result was unlikely to have been obtained by random chance & thus is likely to bear evolutionary relationship to the query sequence.

e.g. $E < 10^{-3}$ $\Rightarrow$ indicative of stat. signif.

$E \sim 10^{-50}$ $\Rightarrow$ strong indication.

* Multiple Sequence Alignment

for 3-D recurrence relation in DP

hypercube vertices are $(x_1^{-1}+0, x_2^{-1}+0, \ldots, x_k^{-1}+0)$

($x_1^{-1}+0$, $x_2^{-1}+0$, ... for each vertex) $\Rightarrow 2^k$

$\rightarrow$ Extend pairwise sequence alignment for k multiple sequences with dynamic programming in a k-dimensional space. $\Rightarrow O((2n)^k)$

$\rightarrow$ Heuristic method: (i) $\binom{k}{2}$ pair alignment & combine them.

or (ii) Get the best alignment & merge them into a new string (once a gap is always a gap). Progressively, merge the best possible alignment with the combined string.

e.g. CLUSTAL (Higgins, Thompson & Gibson (1996))

- CLUSTAL (Higgins & Sharp, 1988) :
  - → Constructs a phylogenetic tree to determine the degrees of similarity among the sequences being aligned.
  - → Using this tree as a guide, closely related sequences are aligned two at a time using DP of the pairwise alignment.
  - → Sensitive to initial choice of pairs.
  - → CLUSTALW (1996) uses sequence weighting, position specific gap penalties & weight matrix choice. Scoring matrix depend on how the sequences are divergent.