



Algorithms in Bioinformatics: A Practical Introduction

Database Search



Biological databases

- Biological data is double in size every 15 or 16 months
- Increasing in number of queries: 40,000 queries per day
- Therefore, we need to have some efficient searching methods for genomic databases



Problem definition

- Consider a database D of genomic sequences (or protein sequences)
- Given a query string Q ,
 - we look for string S in D which is the **closest mach** to the query string Q
 - There are two meanings for closest match:
 - S and Q has a semi-global alignment (forgive the spaces on the two ends of Q)
 - S and Q have a local alignment



Measurement of the goodness of a search algorithm

- Sensitivity

- Ability to detect “true positive”.
- Sensitivity can be measured as the probability of finding the match given the query and the database sequence has only $x\%$ similarity.

- Specificity

- Ability to reject “false positive”
- Specificity is related to the efficiency of the algorithm.

- A good search algorithm should be both sensitive and specific



Different approaches

- This presentation covers only local alignment methods.
- Exhaustive approach
 - Smith-Waterman Algorithm
- Heuristic methods
 - FastA
 - BLAST and BLAT
 - PatternHunter
- Filter and refine approaches
 - LSH
 - QUASAR
- BWT-SW



Smith-Waterman Algorithm

- **Input:**
 - the database D (total length: n) and
 - the query Q (length: m)
- **Output:** all closest matches (based on local alignment)

Algorithm

- For every sequences S in the database,
 - Use Smith-Waterman algorithm to compute the best local alignment between S and Q
- Return all alignments with the best score
- Time: $O(nm)$
- This is a brute force algorithm. So, it is the most sensitive algorithm.



What is FastA?

- Given a database and a query,
 - FastA does local alignment with all sequences in the database and return the good alignments
 - Its assumption is that good local alignment should have some exact match subsequences.
- History of FastA
 - 1983: Wilbur-Lipman algorithm
 - 1985: FastP
 - 1988: FastA



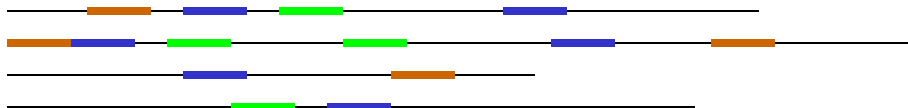
FastP (I)

- Step 1: Look for hot spots
 - For every **k-tuple** (length-k substring) of the query and every k-tuple of the database sequence,
 - If they are the same, the pair is called a **hot spot**.
 - The larger the value of k, the algorithm is faster but less sensitivity
 - Usually, k= 4-6 for DNA sequence and
 - k= 1-2 for protein sequence.

Query



Database



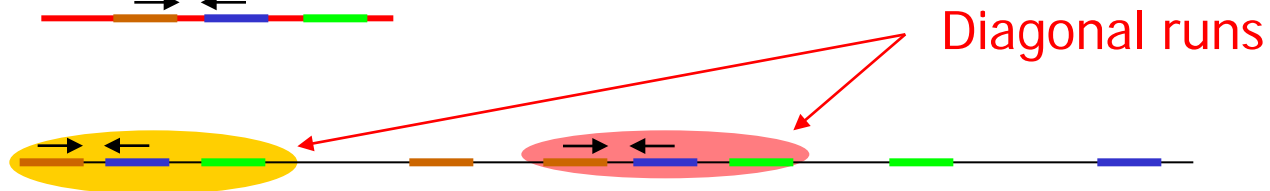
FastP (II)

- Step 2: Find the 10 best diagonal runs for every database sequence
 - **Diagonal run** is a sequence of nearby hot spots on the same diagonal (spaces are allowed between hot spots)
 - Each hot spot is assigned a positive score. Interspot space is given a negative score that decrease with length.
 - The score of a diagonal run is the sum of scores for hot spots and interspot spaces
 - This steps identifies the 10 highest scoring diagonal runs

Query



Database
sequence





FastP (III)

- Step 3: Rescore the 10 best diagonal runs for every database sequence
 - Using the substitution matrix for amino acid (or nucleotide), the diagonal runs are rescored.
 - Sub-region of each diagonal run whose similarity score is maximum is determined.
 - The score of the best of the 10 sub-regions is called the **init1** score.



FastP (IV)

- Step 4: Rank the sequences
 - Step 3 assigns an init1 score for every sequence in the database
 - This step ranks the sequences based on their init1 scores



FastA (I)

- FastA using the same first 3 steps of FastP.
- Then, it executes 2 more steps



FastA (II)

- Step 4: Attempts to join the sub-regions by allowing indels
 - For the 10 sub-regions in Step 3, discard those with scores smaller than a given threshold
 - For the remaining sub-regions, attempts to join them by allowing indels
 - The score of the combined regions is the sum of the scores of the sub-regions minus the penalty for gaps
 - The best score can be computed using dynamic programming and it is called **initn** score.



FastA (III)

- Step 5: banded Smith-Waterman DP
 - Sequences with $initn$ smaller than a threshold are discarded
 - For the remaining sequences, apply banded Smith-Waterman dynamic programming to complete the score opt .
 - Rank the sequences based on their opt scores.



Conclusion for FlastA

- FlastA is much faster than Smith-Waterman.
- It is less sensitive than Smith-Waterman Algorithm.



What is BLAST?

- BLAST = Basic Local Alignment Search Tool
- Input:
 - A database D of sequences
 - A sequence s
- Aim of BLAST:
 - Compare s against all sequences in D in an reasonable time based on heuristics. Faster than FastA
- Disadvantage of BLAST:
 - To be fast, it scarifies the accuracy. Thus, less sensitive

NCBI *nucleotide-nucleotide* **BLAST**
 Nucleotide Protein Translations Retrieve results for an RID

gatggcccaggagaaccccaagatgcacaactcggagatcagcaagcgctggggcgccg
 a

[Search](#)

[Set subsequence](#) From: To:

[Choose database](#) ▾

Now: **BLAST!** or **Reset query** **Reset all**

Options for advanced blasting

[Limit by entrez query](#) or select from: ▾

[Choose filter](#) Low complexity Human repeats Mask for lookup table only Mask lower case

[Expect](#)

```

Query 241 CTTGGCTCCATGGGTTTCGGTGGTCAAGTCCGAGGCCAGCT 280
          |||
Sbjct 1136 CTTGGCTCCATGGGTTTCGGTGGTCAAGTCCGAGGCCAGCT 1175

```

> [gi|854181|emb|Z31560.1|HSSOX2G](#) **U E G** H.sapiens sox-2 mRNA (partial)
 Length=1085

Score = 555 bits (280), Expect = 2e-155
 Identities = 280/280 (100%), Gaps = 0/280 (0%)
 Strand=Plus/Plus

```

Query 1 ATGGACAGTTACGCGCACATGAACGGCTGGAGCAACGGCAGCTACAGCATGATGCAGGAC 60
          |||
Sbjct 481 ATGGACAGTTACGCGCACATGAACGGCTGGAGCAACGGCAGCTACAGCATGATGCAGGAC 540

```

```

Query 61 CAGCTGGGCTACCCGCAGCACCCGGGCTCAATGCGCACGGCGCAGCGCAGATGCAGCCC 120
          |||
Sbjct 541 CAGCTGGGCTACCCGCAGCACCCGGGCTCAATGCGCACGGCGCAGCGCAGATGCAGCCC 600

```

```

Query 121 ATGCACCGCTACGACGTGAGCGCCCTGCAGTACAACCTCCATGACCAGCTCGCAGACCTAC 180
          |||
Sbjct 601 ATGCACCGCTACGACGTGAGCGCCCTGCAGTACAACCTCCATGACCAGCTCGCAGACCTAC 660

```

```

Query 181 ATGAACGGCTCGCCCACCTACAGCATGTCTACTCGCAGCAGGGCACCCCTGGCATGGCT 240
          |||
Sbjct 661 ATGAACGGCTCGCCCACCTACAGCATGTCTACTCGCAGCAGGGCACCCCTGGCATGGCT 720

```

```

Query 241 CTTGGCTCCATGGGTTTCGGTGGTCAAGTCCGAGGCCAGCT 280
          |||
Sbjct 721 CTTGGCTCCATGGGTTTCGGTGGTCAAGTCCGAGGCCAGCT 760

```

> [gi|21591813|gb|AC117415.7|](#) **D** Homo sapiens 3 BAC RP11-43F17 (Roswell Park Cancer Institute Human BAC Library) complete sequence
 Length=161000

RID=1124972832-31271-29540375534.BLASTQ3, - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites Media Print Mail

Address <http://www.ncbi.nlm.nih.gov/BLAST/Blast.cgi> Go

> [gi|30024607|dbj|AB108673.1|](#) **U E G** Mus musculus Sox2 mRNA for transcriptional factor SOX2, complete
cds
Length=1008

Score = 436 bits (220), Expect = 1e-119
Identities = 265/280 (94%), Gaps = 0/280 (0%)
Strand=Plus/Plus

```
Query 1  ATGGACAGTTACGCGCACATGAACGGCTGGAGCAACGGCAGCTACAGCATGATGCAGGAC 60
          |||
Sbjct 508 ATGGACAGCTACGCGCACATGAACGGCTGGAGCAACGGCAGCTACAGCATGATGCAGGAG 567

Query 61  CAGCTGGGCTACCCGACACCCGGGCCTCAATGCGCACGGCGCAGCGCAGATGCAGCCC 120
          |||
Sbjct 568 CAGCTGGGCTACCCGACACCCGGGCCTCAACGCTCACGGCGCGGCACAGATGCAACCG 627

Query 121 ATGCACCGCTACGACGTGAGCGCCCTGCAGTACAACTCCATGACCAGCTCGCAGACCTAC 180
          |||
Sbjct 628 ATGCACCGCTACGACGTGAGCGCCCTGCAGTACAACTCCATGACCAGCTCGCAGACCTAC 687

Query 181 ATGAACGGCTCGCCACCTACAGCATGTCTACTCGCAGCAGGGCACCCCTGGCATGGCT 240
          |||
Sbjct 688 ATGAACGGCTCGCCACCTACAGCATGTCTACTCGCAGCAGGGCACCCCGGTATGGCG 747

Query 241 CTTGGCTCCATGGGTTTCGGTGGTCAAGTCCGAGGCCAGCT 280
          ||
Sbjct 748 CTTGGCTCCATGGGCTCTGTGGTCAAGTCCGAGGCCAGCT 787
```

> [gi|20068540|emb|AL606746.17|](#) **D** Mouse DNA sequence from clone RP23-423J10 on chromosome 3, complete
sequence
Length=203345

Score = 436 bits (220), Expect = 1e-119
Identities = 265/280 (94%), Gaps = 0/280 (0%)

Internet



CGCTCAGGAT AAGACTTCGCGCTAGAGATCGGATCCCCGGGCTGATTATATAGCTCGATCGATC1
TTCTCTATAT CCGCGGATGGGATATATACACACAGCCCGCGGATAGCATGACTGATCTA
CCCCCACTTCCCTTCTCGCATACGTCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTA
CACAGACACCGCTCACACTTACTTAACCAATACGGGAGCGGCTGULGUAUGGAGGAG

Nucleotide

My NCBI
[\[Sign In\]](#) [\[Register\]](#)

PubMed Nucleotide Protein Genome Structure PMC Taxonomy OMIM Books

Search for

Limits Preview/Index History Clipboard Details

Display Show Send to

Range: from to Reverse complemented strand Features: SNP graph CDD MGC HPRD STS tRNA

1: [U31967](#). Reports Mus musculus high...[gi:3419872] [Links](#)

LOCUS MMU31967 2283 bp mRNA linear ROD 14-AUG-1998

DEFINITION Mus musculus high mobility group box protein (sox2) mRNA, complete cds.

ACCESSION U31967

VERSION U31967.1 GI:3419872

KEYWORDS .

SOURCE Mus musculus (house mouse)

ORGANISM [Mus musculus](#)
Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia; Eutheria; Euarchontoglires; Glires; Rodentia; Sciurognathi; Muroidea; Muridae; Murinae; Mus.

REFERENCE 1 (bases 1 to 2283)
AUTHORS Yuan,H., Corbi,N., Basilico,C. and Dailey,L.
TITLE Developmental-specific activity of the FGF-4 enhancer requires the synergistic action of Sox2 and Oct-3
JOURNAL Genes Dev. 9 (21), 2635-2645 (1995)
PUBMED [7590241](#)

REFERENCE 2 (bases 1 to 2283)
AUTHORS Basilico,C.
TITLE Direct Submission
JOURNAL Submitted (20-JUL-1995) Claudio Basilico, Microbiology, NYU Medical Center, 550 First Avenue, New York, NY 10016, USA

COMMENT On Aug 14, 1998 this sequence version replaced gi:1101763



History of BLAST

- 1990: Birth of BLAST1
 - It is very fast and dedicate to the search of local similarities **without** gaps
 - Altschul et al, Basic local alignment search tool. J. Mol. Biol., 215(3):403-410, 1990.
 - The most highly cited paper in 1990 and the third most highly cited paper in the past 20 years (1983-2002).
- 1996-1997: Birth of BLAST2
 - BLAST2 allows insertion of gaps
 - BLAST2 have two versions. Developed by two groups of authors independently
 - 1997: NCBI-BLAST2 (National Center for Biotechnology Information)
 - 1996: WU-BLAST2 (Washington University)



BLAST1

- A heuristic method which searches for local similarity without gap
- It can be divided into four steps:
 - Step 1: Query preprocessing
 - Step 2: Scan the database for hits
 - Step 3: Extension of hits



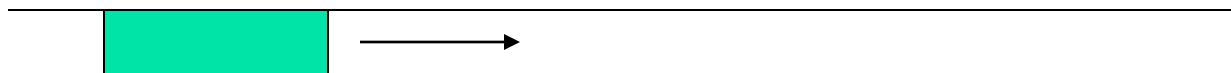
Step 1: Query preprocessing

- For every position p of the query, find the list of w -tuples (length- w strings) scoring more than a **threshold T** when paired with the word of the query starting at position p . This list of w -tuples are called **neighbors**.
 - For DNA, $w=11$ (default)



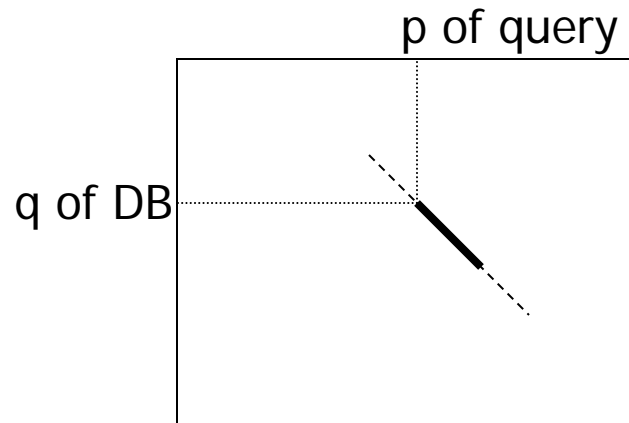
Step 2: Generation of hits

- Scan the database DB.
 - For each position p of the query, if there is an exact match between the neighbors of p and a w -tuple in DB, a **hit** is made.
- A hit is characterized by the positions in both query and DB sequences.



Step 3: Extension of hits (I)

- For every hit, extend it in both directions, without gaps.
- The extension is stopped as soon as the score decreases by more than X (parameter of the program) from the highest value reached so far.





Step 3: Extension of hits (II)

- If the extended segment pair has score better than or equal to S (parameter of the program), it is called an **HSP (High scoring segment pair)**. Then, they will be reported.
- For every sequence in the database, the best scoring HSP is called the **MSP (Maximal segment pair)**.

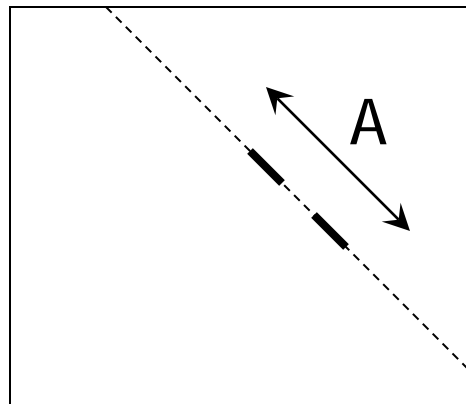


NCBI-Blast2

- Allows local alignment with gaps.
- The first 2 steps are the same as BLAST1.
- Two major differences:
 - Two-hits requirement (implemented for protein)
 - Gapped extension

Step 3: Two-hits requirement

- To extend a hit, we require that there is another hit on the same diagonal within a distance smaller than A
- By default, $A=40$



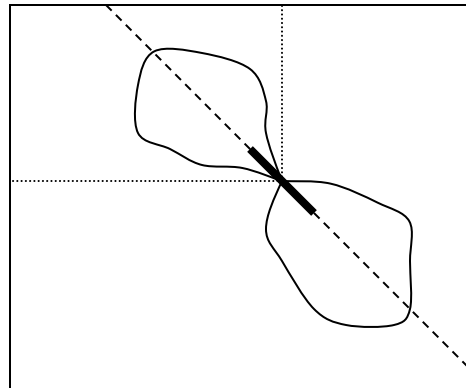


Step 4: Gapped extension (I)

- For hits satisfying the two-hits requirement, extend them similar to Step 3 of BLAST1
- Among the generated HSP, we perform **gapped extension** for those with score > some threshold

Step 4: Gapped extension (II)

- Gapped extension is a modified Smith-Waterman algorithm
 - Explore the dynamic programming starting from the middle of the hit
 - When the alignment score drops off by more than X_g , stop





BLAST1 vs. NCBI-BLAST2

- BLAST1 spends 90% of its time on extension
- For NCBI-BLAST2, due to the two-hits requirement, the number of extensions is reduced.
 - NCBI-BLAST2 is about 3 times faster than BLAST1.



BLAST program options

Program	Query Sequence	Database	type of alignment
Blastp	Protein	Protein	The protein query sequence is searched in the protein database.
Blastn	DNA	DNA	The DNA query sequence is searched in the DNA database.
Blastx	DNA	Protein	The DNA query sequence is converted into protein sequences in all six reading frames. Then, those translated proteins are searched in the protein database.
Tblastn	Protein	DNA	The protein query sequence is searched against the protein sequences generated from the six reading frames of the DNA sequences in the database.
Tblastx	DNA	DNA	The DNA query sequence is translated into protein sequences in all six reading frames. Then, the translated proteins are searched against the protein sequences generated from the six reading frames of the DNA sequences in the database.



Statistics for local alignment

- A local alignment without gaps consists simply of a pair of equal length segments.
- BLAST and FASTA find the local alignments whose score cannot be improved by extension. In BLAST, such local alignments are called high-scoring segment pairs or HSPs.
- To determine the significance of the local alignments, BLAST and FASTA show E-value and bit score. Below, we give a brief discussion on them.
- Assumption: We required the expected score for aligning a random pair of residues/bases to be negative.
 - Otherwise, the longer the alignment, the higher is the score independent of whether the segments aligned are related or not.



E-value

- E-value is the expected number of alignments having raw score $> S$ totally at random.
- Let m and n be the lengths of the query sequence and the database sequence.
- When both m and n are sufficiently long,
 - the expected number E of HSPs with score at least S follows the extreme distribution (Gumbel distribution). We have
 - $E = K m n e^{-\lambda S}$
for some parameters K and λ which depends on the scoring matrix δ and the expected frequencies of the residues/bases.
- The formula is reasonable since:
 - Double the length of either sequence will double the expected number of HSPs.
 - Double the score S will exponentially reduce the expected number of HSPs.
- Hence, when E-value is small, the HSP is significant.



Bit score

- The raw score S of an alignment depends on the scoring system.
- Without knowing the scoring system, the raw score is meaningless.
- The bit score is defined to normalize the raw score, which is defined as follows.

$$S' = \frac{\lambda S - \ln K}{\ln 2}$$

- Note that $E = m n e^{-S'}$.
- Hence, when S' is big, the HSP is significant.



P-value

- The number of random HSPs with score $\geq S$ follows a Poisson distribution.
- $\Pr(\text{exactly } x \text{ HSPs with score } \geq S) =$
 - $e^{-E} E^x / x!$
 - where $E = K m n e^{-\lambda S}$ is the E-score
- Hence, p-value = $\Pr(\text{at least 1 HSPs with score } \geq S) =$
 - $1 - e^{-E}$.
- Note:
 - when E increases, p-value is approaching 1.
 - When $E=3$, p-value is $1 - e^{-3} = 0.95$.
 - When $E=10$, p-value is $1 - e^{-10} = 0.99995$
 - when $E < 0.01$, $1 - e^{-E} \approx E$.
- Hence, in BLAST, p-value is not shown since we expect p-value and E-value are approximately the same when $E < 0.01$ while p-value is almost 1 when $E > 10$.



Local alignment with gaps

- There is no solid theoretical foundation for local alignment with gaps.
- Moreover, experimental results suggested that the theory for ungapped local alignment can be applied to the gapped local alignment as well.



Variation of BLAST

- MegaBLAST
- BLAT
- PatternHunter
- PSI-BLAST



MegaBLAST

- Only for DNA
- For DNA, in BLAST, $w = 11$ by default.
- To improve efficiency, MegaBLAST uses longer w -tuples (by default, $w=28$).
- The cost is the reduction in sensitivity.



BLAT

- BLAT is also for DNA only.
- It improves the efficient by a lot.
- The main trick is to put the index in the main memory
- For DNA, by default, they use two-hit and $w=11$.
- Note that BLAT is less sensitive than BLAST, but more sensitive than MegaBLAST.



PatternHunter

- PatternHunter can only apply to DNA
- PatternHunter is similar to BLAST. Moreover, it uses **gapped w-tuple**.

- For $w=11$, they use 111010010100110111
- Example,

```
111010010100110111
ACTCCGATATGCGGTAAC
|||-|--|--||-|||
ACTTCACTGTGAGGCAAC
```

- They found that gapped w-tuple can increase the **sensitivity** while increase the **efficiency**.

Advantage of gapped w-tuple (I)

- Increase sensitivity

- Gapped w-tuples are more independent.

- Examples:

- Two adjacent ungapped 11-tuples share 10 symbols

- 111111111111
111111111111

- Two adjacent gapped 11-tuples share 5 symbols

- 111010010100110111
111010010100110111

- If the w-tuples are more independent, the probability of having at least one hit in a homologous region is higher.

Advantage of gapped w-tuple (II)

- Reduce the number of hits.
 - For the same query length (says, 64),
 - It covers by 54 ungapped 11-tuples
 - It covers by 47 gapped 11-tuples
 - So, the number of hits is smaller.
- Thus, the efficiency is increased!

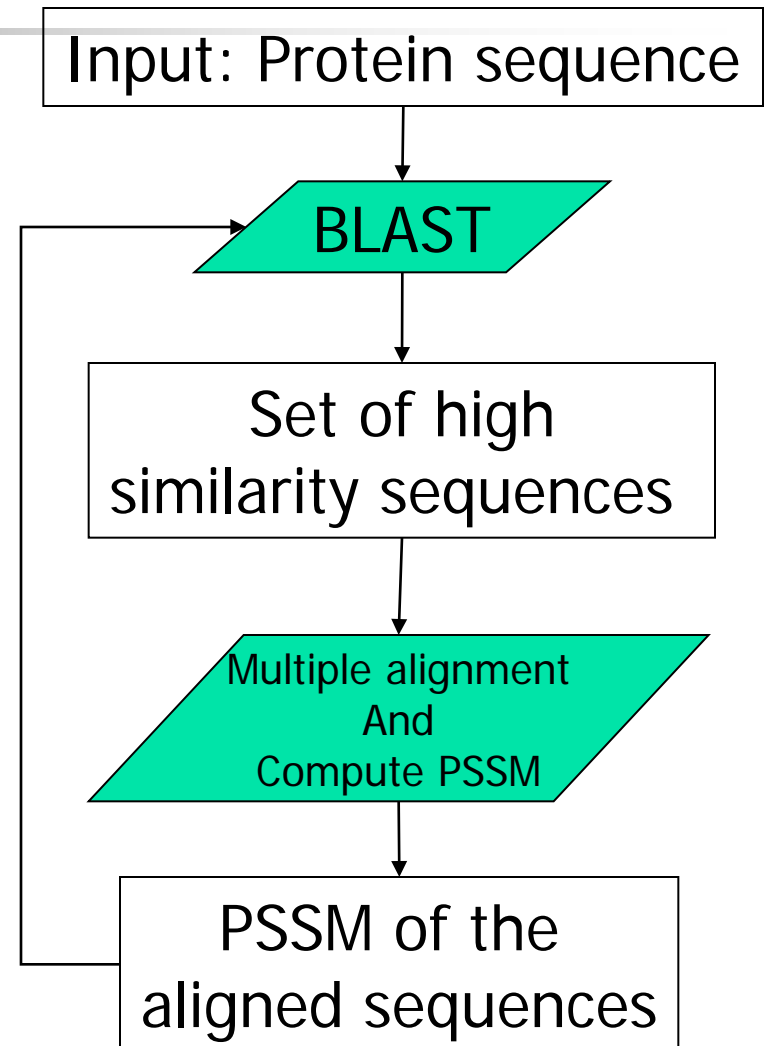


More for PatternHunter

- To further improve the efficiency,
 - PatternHunter uses a variety of advanced data structures including priority queues, a variation of red-black tree, queues, hash tables.
 - PatternHunter also uses a new method of sequence alignment.
- To further improve the accuracy,
 - PatternHunter suggested to use multiple gapped seeds.
 - They show that the accuracy can approach smith-waterman algorithm while the speed 3000 times faster than smith-waterman.
- PatternHunter is both faster and sensitive than BLAST, MegaBLAST.

PSI-BLAST (Position Specific Iterated BLAST)

- PSI-BLAST is an implementation of BLAST for finding protein families. It allows us to detect distant homology.
- Input: a protein sequence
 - Using BLAST, we get a set of sequences that align with the query protein with E-score below a threshold, 0.01 (by default).
 - Align the selected sequences
 - Generate a PSSM profile from the multiple alignment
 - Iterate until no significant alignment found,
 - Using a modified BLAST, search the database with the PSSM profile.
 - Align the selected sequences
 - Generate a PSSM from the multiple alignment
- This version automatically combines statistically significant alignments produced by BLAST into a position-specific score matrix.
- It is much more sensitive to weak but biologically relevant sequence similarities





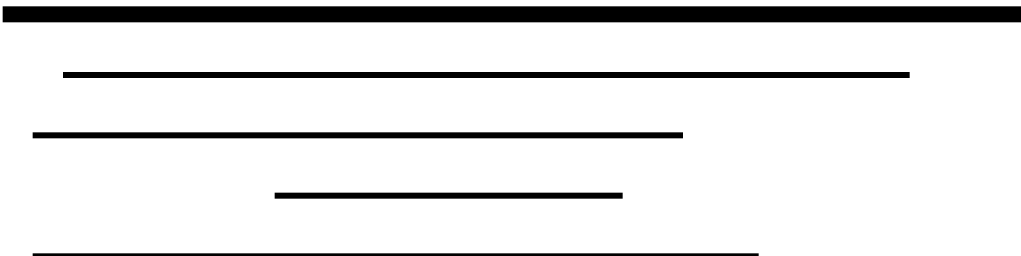
Find a set of sequences similar to the query

- Using BLAST 2.0, we get a set of sequences that align with the query protein with E-score below a threshold, 0.01 (by default).
- Keep one copy of the selected sequences which are >98% identical.



Multiple sequence alignment of the selected sequences

- Using the query sequence as the template, we aligned the selected sequences.
- All gap characters inserted into the query sequence are ignored.
- Note:
 - the length of the alignment is the same as the query sequence.
 - Some columns of the multiple sequence alignment may include nothing except the query.

query 



Generate a PSSM profile from the alignment

- Given the multiple alignment of length n ,
 - We generate the position-specific score matrix (PSSM) profile, which is a $20 \times n$ matrix.
 - For each column and each residue a in the profile, we generate a log-odds score $\log(O_{ia}/P_a)$.
 - where O_{ia} is the observed frequency of residue a at position i and P_a is the expected frequency respectively of the residue a .
- Since number of sequences may be small, data-dependent pseudo frequency is added to O_{ia} .



Find a set of sequences similar to the PSSM profile

- We apply a modified BLAST to the PSSM profile.
 - Basically, when we compare a position of the PSSM and a residue in the database, we use the corresponding log-odds score in that position.



QUASAR

- QUASAR stands for Q-gram Alignment based on Suffix ARrays
- It is a good searching tool for identifying strong similar strings.
- **Problem:**
 - Input: a database D , a query S , k , w
 - Output: a set of (x, y) where
 - x and y are length- w substring in D and S , respectively
 - $\text{edit_dist}(x, y) \leq k$

gcagactgctac k=3

gccgacagccac w=12

q=3

$t = w + 1 - (k + 1)q$
 $= 12 + 1 - (3 + 1)3$

$= 1$



Observation

Lemma:

Given two length- w sequences X and Y , if they have edit distance $\leq k$, then they share at least t common q -grams (length- q substrings) where $t = w + 1 - (k + 1)q$.

Proof:

- Suppose X and Y has r differences.
- X has $(w + 1 - q)$ q -grams
- Note that a q -gram in X overlaps with some difference iff X and Y does not share that q -gram
- For each difference, there are at most q q -grams overlap with the difference. In total, rq q -grams overlap with the r differences
- Thus, X and Y share $(w + 1 - q - rq)$ q -grams, which is bigger than $w + 1 - (k + 1)q$.

- We make use of this observation to do filtering!

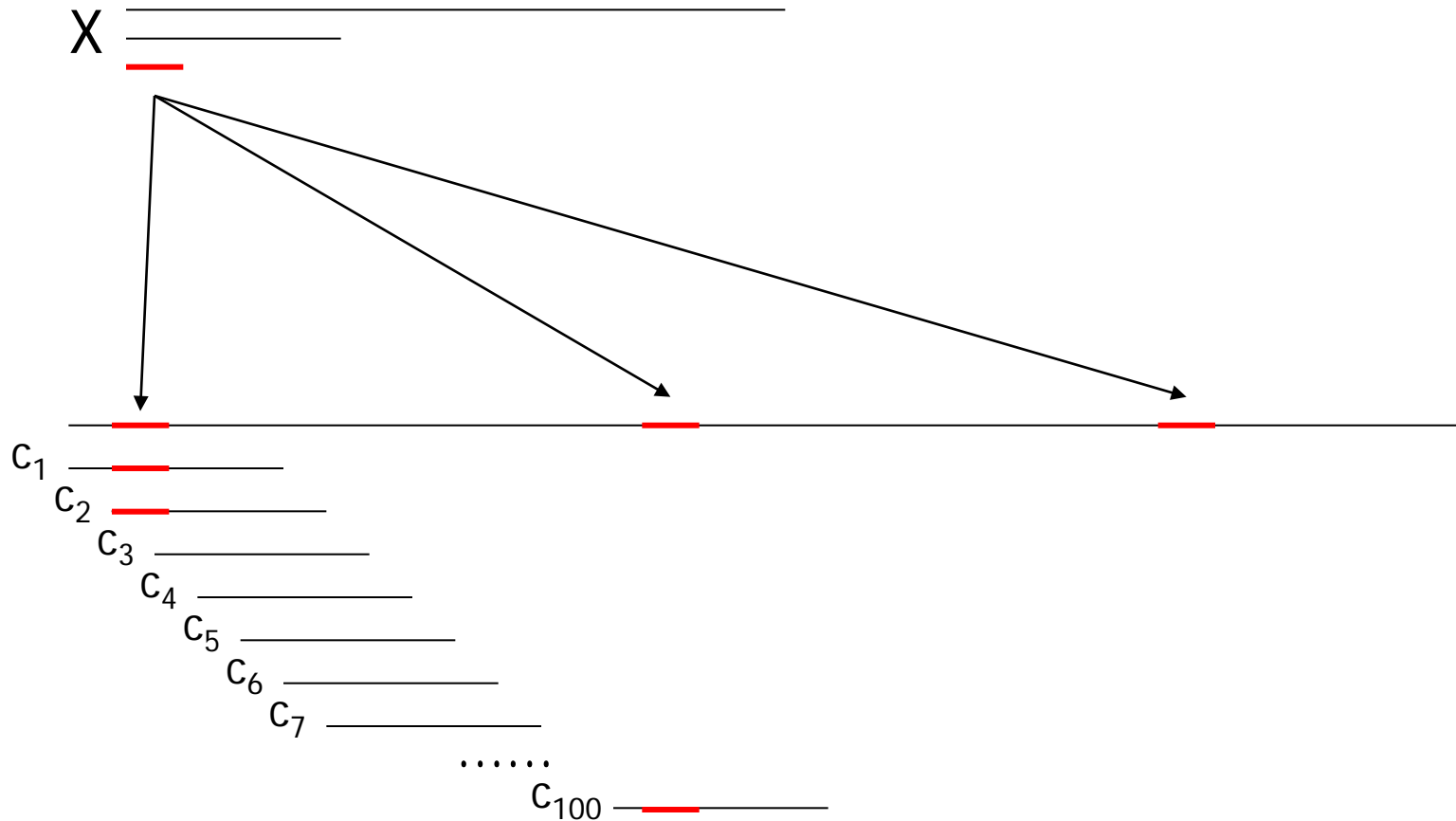


Algorithm for finding potential approximate matches of S in D

- For every $X = S[i..i+w-1]$ of the query where $i=1, 2, \dots$
 - For every length- w substring Y in D , associate a **counter** with it and initialize it to zero
 - For each q -gram Q in X ,
 - Find the **hitlist**, that is, the list of positions in D so that Q occurs
 - Increment the counter for every length- w substring Y which contains Q
 - For every length- w substring Y in D with **counter** $> t$, X and Y are potential approximate match. We check it using an alignment algorithm!



Illustration of the algorithm

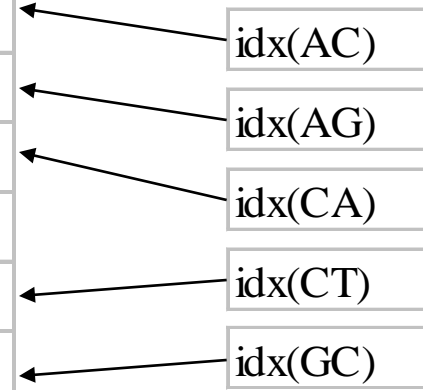


How to get the hitlist?

- Based on the data-structures
 - A **suffix array SA** of the database D is the lexicographically ordered sequence of all suffixes in D.
 - An **auxiliary array idx** where for each q-gram Q, $\text{idx}[Q]$ is the start of the hitlist for Q!

	1	2	3	4	5	6	7
Database D =	C	A	G	C	A	C	T

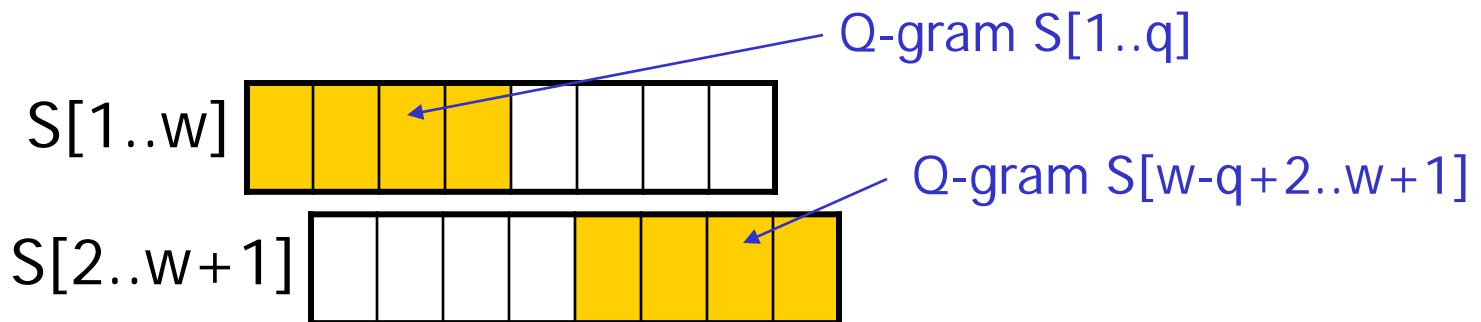
i	SA[i]	
1	5	ACT
2	2	AGCACT
3	4	CACT
4	1	CAGCACT
5	6	CT
6	3	GCACT
7	7	T



The diagram shows five boxes on the right labeled $\text{idx}(AC)$, $\text{idx}(AG)$, $\text{idx}(CA)$, $\text{idx}(CT)$, and $\text{idx}(GC)$. Arrows point from these boxes to the SA entries in the table: $\text{idx}(AC)$ points to row 1, $\text{idx}(AG)$ points to row 2, $\text{idx}(CA)$ points to row 3, $\text{idx}(CT)$ points to row 5, and $\text{idx}(GC)$ points to row 6.

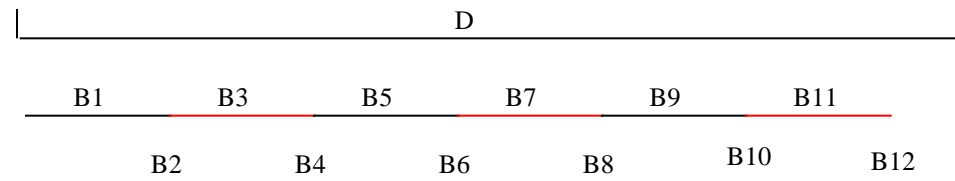
Speedup Feature 1: Window shifting

- In the previous algorithm, building the counters list for $S[i..w+i-1]$ is time consuming!
- Suppose the counters list for $S[1..w]$ is given, can we determine the counters list for $S[2..w+1]$ easily?
 - Idea: For every length- w string Y in D ,
 - Decrement counter for Y if it contains q -gram $S[1..q]$
 - Increment counter for Y if it contains q -gram $S[w-q+2..w+1]$
- The window shifting idea reduce the time complexity.



Speedup Feature 2: Block addressing

- Another problem: too many counters
- Solution (Block addressing scheme):
 - Instead of associate a counter for every length- w substring Y in D
 - The database D is divided into blocks of size b ($b \geq 2w$). Each block is associating a counter.
 - If a block contains more than t q -grams, this block has to be checked for approximate matches using an alignment algorithm





Weakness of QUASAR

- Extensive memory requirement
 - Construction phase:
 - Memory space = $9n$ (where n is DB size)
 - Query phase:
 - Memory space = $5n$
- Not suitable for distant homologous sequences



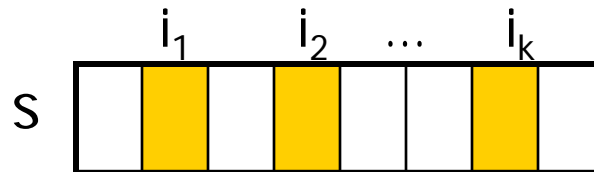
Locality-Sensitive Hashing (LSH)

LSH-ALL-PAIRS

- **Input**: biosequence database D
- **Aim**: find pairs of w -mers that differ by at most d substitutions (ungapped local alignment) in a collection of biosequences D .

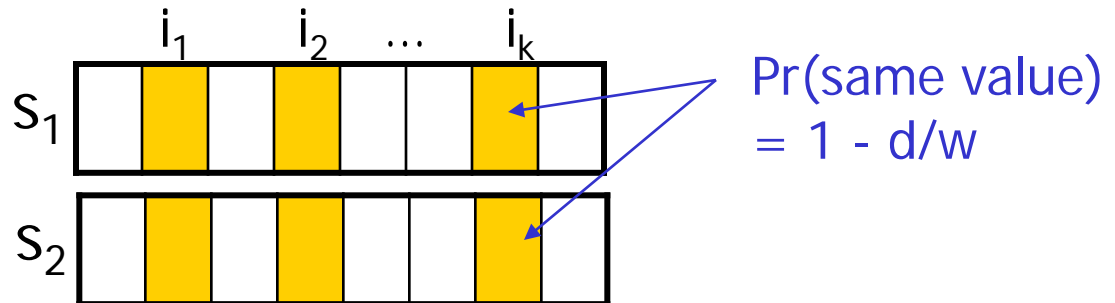
Locality-sensitive hash function

- Consider an w -mers s ,
 - choose k indices i_1, i_2, \dots, i_k uniformly from the set $\{1, 2, \dots, w\}$
 - Define $\pi(s) = (s[i_1], s[i_2], \dots, s[i_k])$. This function is called the **locality-sensitive hash function**



Property of locality-sensitive hash function (I)

- Consider two w -mers s_1 and s_2 ,
 - the more similar are they, the higher probability that $\pi(s_1) = \pi(s_2)$.
- More precisely, if the hamming distance of s_1 and $s_2 = d$,
 - $\Pr[\pi(s_1) = \pi(s_2)] = \prod_{j=1, \dots, k} \Pr[s_1[i_j] = s_2[i_j]]$
 $= (1 - d/w)^k$





Property of locality-sensitive hash function (II)

- Hence, s_1 and s_2 are similar if
 - $\pi(s_1) = \pi(s_2)$
- However, we may have false positive and false negative
 - **False positive:** s_1 and s_2 are dissimilar but $\pi(s_1) = \pi(s_2)$.
 - False positive can be distinguished from true positive by computing hamming distance between s_1 and s_2
 - **False negative:** s_1 and s_2 are similar but $\pi(s_1) \neq \pi(s_2)$.
 - We cannot detect false negative.
 - We can only reduce the number of false negative by repeating the test using different $\pi()$ functions



LSH-ALL-PAIRS

Algorithm:

1. Generate m random locality-sensitive hash functions $\pi_1(\), \pi_2(\), \dots, \pi_m(\)$.
2. For every w -mer s in the database, compute $\pi_j(s)$ for $1 \leq j \leq m$.
3. For every pair of w -mers s and t such that $\pi_j(s) = \pi_j(t)$ for some j ,
 - If hamming distance(s, t) $< d$, report (s, t) -pair.



Local alignment

- In the sequence alignment lecture,
 - We say local alignment is very time consuming to compute.
- For example,
 - It takes more than 15 hours to align 1000 nucleotides with the human genome.
- Hence,
 - Many heuristic solutions like BLAST, BLAT, Pattern Hunter have been proposed.
 - However, they cannot guarantee to find the optimal local alignment.
- Question: Can we compute optimal local alignment within reasonable time?



Aligning pattern to suffixes

- Given a text S and a pattern Q
- Finding the best local alignment can be rephrased as
 - finding the best global alignment between any substring of Q and any prefix of $S[k..n]$
 - among all suffix- k $S[k..n]$ of S .



Detail of the alignment computation

- Consider a particular suffix-k S' of S , we apply dynamic programming to find the best alignment between any substring of Q and any prefix of S' .
- Let $V(i,j)$ be the optimal alignment score between any substring of $Q[1..i]$ and $S'[1..j]$.
- Our aim is to compute $\max_{1 \leq i \leq m, 1 \leq j \leq n} \{V(i,j)\}$
- Base case ($i=0$ or $j=0$):
 - $V(i,0)=0$
 - $V(0,j)=-j$
- Recursive case ($i>0$ and $j>0$):
 - $V(i,j) = \max\{ V(i-1,j-1)+\delta(Q[i],S'[j]), V(i,j-1)-1, V(i-1,j)-1 \}$

Detail of the alignment computation

- Example: Assuming match=2, mismatch/insert/delete=-1.
- S = acacag, Q = ctc
- Below is an example for finding the best alignment between any substring of Q and any prefix of S[1..n].

	_	a	c	a	c	a	g
_	0	-1	-2	-3	-4	-5	-6
c	0	-1	1	0	-1	-2	-3
t	0	-1	0	0	-1	-2	-3
c	0	-1	1	0	2	1	0

Hence, we have
 acac
 -ctc



Getting alignment between pattern Q and suffixes of S

- Example: Assuming match=2, mismatch/insert/delete=-1.
- S = acacag
- Q = ctc

- All suffixes of S:
 - acacag, cacag, acag, cag, ag, g

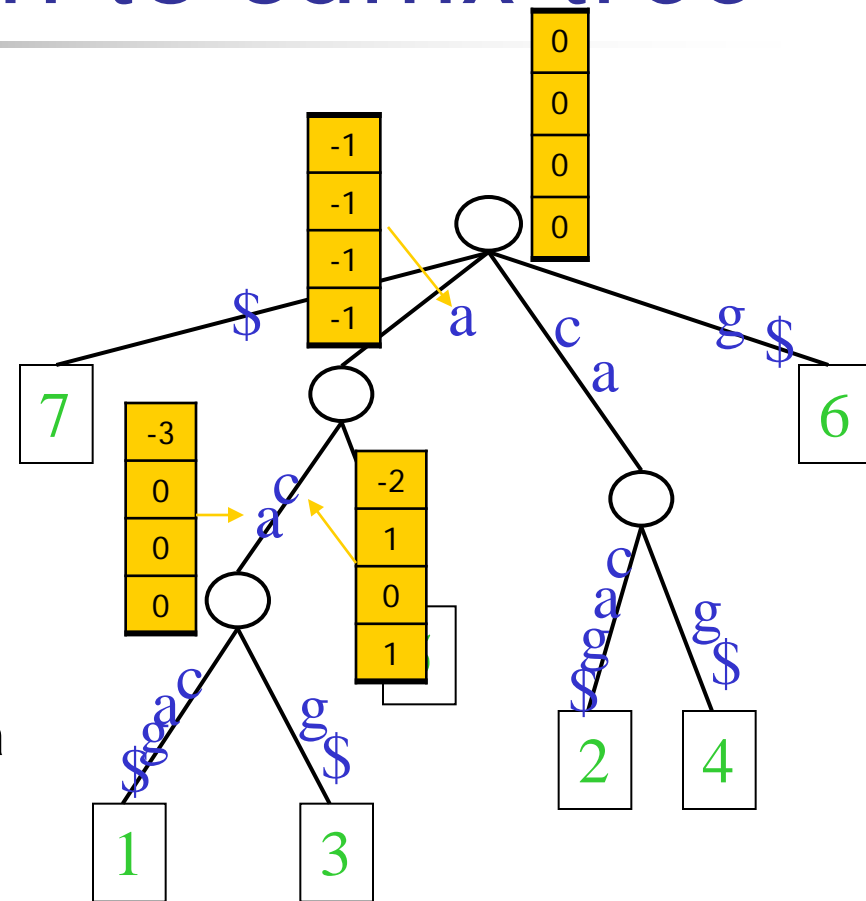
- Suffix 1: (score=2)
 - acac
 - -ctc
- Suffix 2: (score=3)
 - cac
 - ctc

- Suffix 3: (score=1)
 - ac
 - -c
- Suffix 4: (score=2)
 - c
 - c
- Suffix 5: (score=0)
 - -
 - -
- Suffix 6: (score=0)
 - -
 - -

- Best local alignment score is 3.

Aligning pattern to suffix tree

- Note that every suffix of S corresponds to a path in the suffix tree.
- Suffix tree helps to avoid redundant table filling if two suffixes share a common prefix.
- For example, for suffix-1 and suffix-3, they share the common prefix *aca*. By suffix tree, we only need to fill-in the 3 columns for *aca* once.





How deep should we go when we align a pattern to a suffix tree?

- The depth of the suffix tree is n (since suffix-1 is of length n). Do we need to go to depth- n ?
 - No!
- If the pattern is of length m ,
 - we only need to go down the tree by at most cm characters
 - for some constant c depending on the scoring matrix.
- For our scoring matrix, we need to go down to at most $3m$.
 - Reason: In the alignment,
 - No. of match/mismatch positions $x \leq m$
 - No. of spaces = y
 - The alignment score $\leq 2x - y \leq 2m - y$
 - Since the alignment score must be non-zero, we have $2m - x \geq 0$.
 - Hence, we need to go down at most $x + y$ characters, which is smaller than $3m$.



Algorithm for local alignment using suffix tree

- Input:
 - the suffix tree T of the text S
 - the pattern Q of length m
- Algorithm:
 - Traverse the suffix trie up to depth cm in DFS order.
 - When we go down the tree T by one character, we fill-in one additional column of the DP table.
 - When we go up the tree T by one character, we undo one column of the DP table.



Time analysis

- Let L be the number of paths in T with depth = cm .
- Note that $L = \min\{n, \sigma^{cm}\}$.
- The number of nodes in those paths is at most cmL .
- For each node, we compute a column of size m .
- Hence, the worst case running time is $cm^2L = O(\min\{cm^2n, cm^2\sigma^{cm}\})$ time.

- When m is small, the worst case running time is faster than $O(nm)$.
- When m is big, the worst case running time is bad.
- Moreover, in practice, the running time is ok.



Can we do better?

- In the rest of this discussion,
 - we proposed another concept called **meaningful alignment** which enable effective pruning.
- Hence, we can improve the running time by a lot in practice.



Meaningful alignment

- Consider an alignment A of a substring X of S and a substring Y of the pattern Q .
- If the alignment score of some prefix X' of X and some prefix Y' of Y is less than or equal to zero,
 - We say A is a **meaningless** alignment;
 - Otherwise, A is a **meaningful** alignment.

Example of Meaningless alignment

- Below is an example alignment A between Q[1..3] and S[1..4].

acac

-ctc

- Note that the alignment between Q[1..2] and S[1..3] has score zero. Hence, A is meaningless.

	_	a	c	a	c	a	g
_	0	-1	-2	-3	-4	-5	-6
c	0	-1	1	0	-1	-2	-3
t	0	-1	0	0	-1	-2	-3
c	0	-1	1	0	2	1	0



Lemma

- Lemma: Consider an alignment A of substring $S[h..i]$ and substring $Q[k..j]$. If A is meaningless and has score C ,
 - then there exists a meaningful alignment of $X'=S[s..i]$ and $Y'=Q[t..j]$ with score at least C .
- Corollary: The optimal local alignment between S and Q is the optimal meaningful alignment between any substring of S and any substring of Q .

Example of Meaningless alignment

- Consider the meaningless alignment A between S[1..4] and Q[1..3]. (The score of A is 2.)

acac

-ctc

- The prefixes S[4..4] and Q[3..3] form a meaningful alignment and has score 2.

	_	a	c	a	c	a	g
_	0	-1	-2	-3	-4	-5	-6
c	0	-1	1	0	-1	-2	-3
t	0	-1	0	0	-1	-2	-3
c	0	-1	1	0	2	1	0



How to find the best meaningful alignment?

- For any suffix-k S' of S ,
 - we apply dynamic programming to find the best meaningful alignment between any substring of Q and any prefix of S' .
- Let $V(i,j)$ be the best meaningful alignment score between any suffix of $Q[1..i]$ and $S'[1..j]$.
- Our aim is to compute $\max_{1 \leq i \leq m, 1 \leq j \leq n} \{V(i,j)\}$
- Base case ($i=0$ or $j=0$):
 - $V(i,0)=0$
 - $V(0,j)=-\infty$
- Recursive case ($i>0$ and $j>0$):
 - If $V(i-1,j-1)>0$, $V_1(i,j)=V(i-1,j-1)+\delta(Q[i],S'[j])$; $V_1(i,j)=-\infty$, otherwise
 - If $V(i,j-1)>0$, $V_2(i,j)=V(i,j-1)-1$; $V_2(i,j)=-\infty$, otherwise
 - If $V(i-1,j)>0$, $V_3(i,j)=V(i-1,j)-1$; $V_3(i,j)=-\infty$, otherwise
 - $V(i,j)=\max\{V_1(i,j), V_2(i,j), V_3(i,j)\}$

Example of Meaningless alignment

- Below is an example meaningful alignment A between S[2..4] and Q[1..3].

cac

ctc

	_	c	a	c	a	g
_	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
c	0	2	1	0	$-\infty$	$-\infty$
t	0	$-\infty$	1	0	$-\infty$	$-\infty$
c	0	$-\infty$	$-\infty$	3	2	1



Getting meaningful alignment between pattern Q and suffixes of S

- Example: Assuming match=2, mismatch/insert/delete=-1.
- S = acacag
- Q = ctc

- All suffixes of S:
 - acacag, cacag, acag, cag, ag, g

- Suffix 1: (score=0)
 - -
 - -
- Suffix 2: (score=3)
 - cac
 - ctc

- Suffix 3: (score=0)
 - -
 - -
- Suffix 4: (score=2)
 - c
 - c
- Suffix 5: (score=0)
 - -
 - -
- Suffix 6: (score=0)
 - -
 - -

- Best local alignment score is 3.

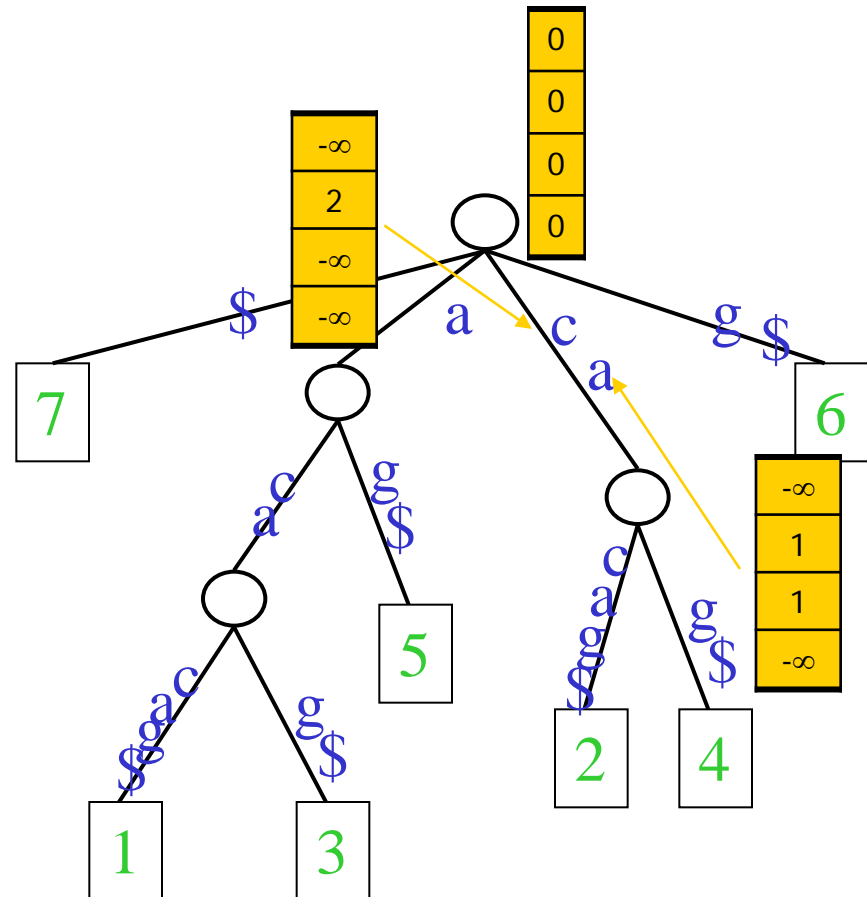


Observation

- First, we can find meaningful alignments on suffix tree T of S . This avoid redundant computations.
- Second, in practice, most of the entries in the dynamic programming are either zero or $-\infty$.
- This allows us to have two pruning strategies.

Pruning strategy 2

- When we go down the tree, we don't need $O(m)$ time to compute a column when the column has many non-positive entries.





How about gap penalty?

- The algorithm can be extended to handle affine gap penalty.

- Not discuss!



Suffix tree is too big!

- For human genome, we need to put the suffix tree in disk which slowed down the dynamic programming.
- Simulating suffix trie using BWT.
 - We reverse the sequence S .
 - Then, we simulate the suffix trie using backward search.
- Hence, we can store the suffix trie for human genome using less than 4G RAM.
- We can traverse the suffix trie in DFS order using the same time complexity since backward search takes $O(1)$ time.



Experimental result

- Based on the above discussion, BWT-SW is developed to find optimal local alignment.
- Using a Pentium D 3.0GHz PC with 4G RAM, the running time of BTW-SW to align a pattern with the human genome is summarized as follows.
- Note that Smith-Waterman algorithm takes more than 15 hours to align a pattern of length 1K against the human genome.

Query length	100	200	500	1K	2K	5K
Average time (seconds)	1.91	4.02	9.89	18.86	35.93	81.60

Query length	10K	100K	1M	10M	100M
Average time (seconds)	161.04	1.4K	8.9K	34.4K	218.2K



Completeness of BLAST (I)

- BLAST is the most popular solution for finding local alignments. It is well-known that BLAST is heuristics and it will miss solution.
- Since BWT-SW can find all optimal local alignments, we would like to check how many good alignments are missed by BLAST.
- We extracted 2000 mRNA sequences from each of the 4 different species. We aligned them on human genome. Then, we checked how many significant alignments are missed by BLAST.



Completeness of BLAST (II)

	Chimpanzee	Mouse	Chicken	Zebrafish	All 4 species
E-value (\leq)	Missing %	Missing %	Missing %	Missing %	Missing %
1.0×10^{-16}	0.00	0.03	0.05	0.06	0.01
1.0×10^{-15}	0.00	0.03	0.05	0.06	0.02
1.0×10^{-14}	0.00	0.04	0.06	0.06	0.02
1.0×10^{-13}	0.00	0.03	0.07	0.14	0.02
1.0×10^{-12}	0.01	0.04	0.10	0.17	0.03
1.0×10^{-11}	0.02	0.05	0.11	0.28	0.05
1.0×10^{-10}	0.02	0.07	0.13	0.39	0.06
1.0×10^{-9}	0.03	0.09	0.16	0.60	0.08
1.0×10^{-8}	0.05	0.11	0.25	0.77	0.12
1.0×10^{-7}	0.10	0.19	0.31	0.81	0.18
1.0×10^{-6}	0.17	0.31	0.45	1.08	0.28
1.0×10^{-5}	0.32	0.47	0.70	1.45	0.45
1.0×10^{-4}	0.57	0.88	0.99	1.81	0.75
1.0×10^{-3}	0.99	1.36	1.25	2.25	1.17
1.0×10^{-2}	1.69	2.11	1.68	2.61	1.84
1.0×10^{-1}	2.70	2.97	2.33	2.86	2.76

- BLAST only missed 0.06% of those 8000 queries (with E-value smaller than 1.0×10^{-10}).
- In conclusion, BLAST is accurate enough in most cases, yet the few alignments missed could be critical for biological research.



Conclusion

- This presentation discusses some database searching methods.
- Due to the advance in short tag sequencing, a number of new methods are proposed. For examples:
 - BWA, Bowtie, RMAP, SOAP2



More information

- The list of database used by blast
 - http://www.ncbi.nlm.nih.gov/blast/blast_databases.shtml
 - <ftp://ftp.ncbi.nlm.nih.gov/blast/db/>