Seeding Methods in Homology Search

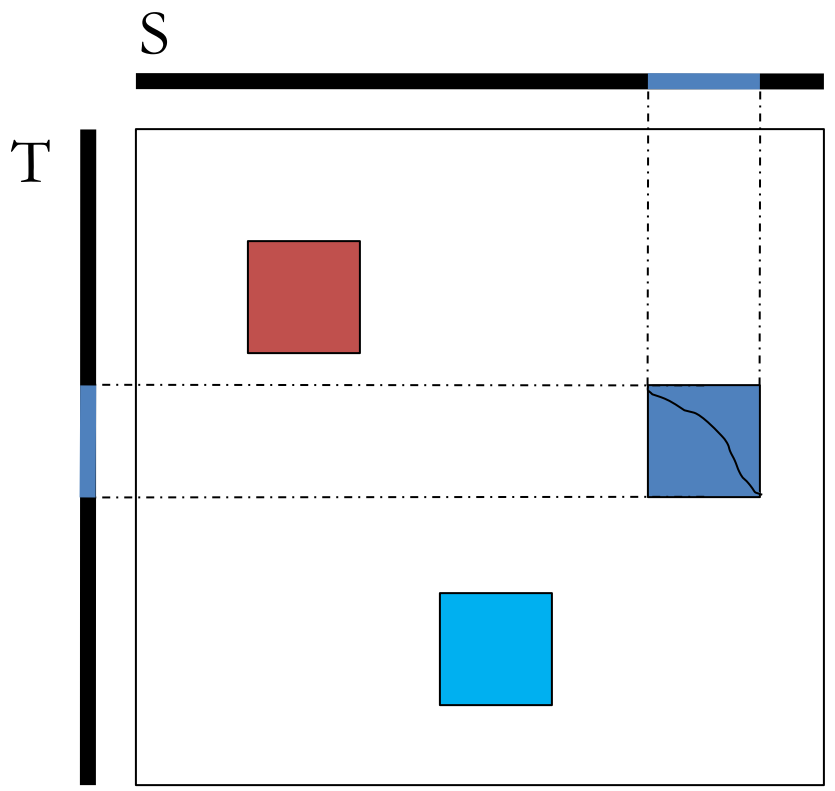
A similarity between mouse and human genomes

```
GCNTACACGTCACCATCTGTGCCACCACNCATGTCTCTAGTGATCCCTCATAAGTTCCAACAAAGTTTGC
|| |||||  | |||  ||||   ||    |||||||||||||||||| | |||||||| |  | |||||
GCCTACACACCGCCAGTTGTG-TTCCTGCTATGTCTCTAGTGATCCCTGAAAAGTTCCAGCGTATTTTGC

GAGTACTCAACACCAACATTGATGGGCAATGGAAAATAGCCTTCGCCATCACACCATTAAGGGTGA----
|| ||||||||| ||||||  | |||||   |||||||| ||| ||||||||  |  | | ||
GAATACTCAACAGCAACATCAACGGGCAGCAGAAAATAGGCTTTGCCATCACTGCCATTAAGGATGTGGG

------------------TGTTGAGGAAAGCAGACATTGACCTCACCGAGAGGGCAGGCGAGCTCAGGTA
                  ||||||||||||| ||| ||||||||||| || ||||||| || ||||   |
TTGACAGTACACTCATAGTGTTGAGGAAAGCTGACGTTGACCTCACCAAGTGGGCAGGAGAACTCACTGA

GGATGAGGTGGAGCATATGATCACCATCATACAGAACTCAC-------CAAGATTCCAGACTGGTTCTTG
||||||| |||| | | |||| ||||| || |||||  ||        |||||| |||||||||||||||
GGATGAGATGGAACGTGTGATGACCATTATGCAGAATCCATGCCAGTACAAGATCCCAGACTGGTTCTTG
```

Smith-Waterman is the most accurate method.

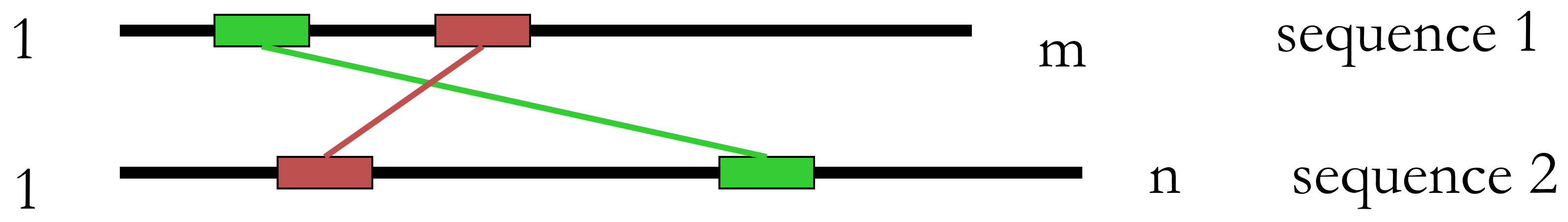
Time complexity：O(mn).

Smith-Waterman Algorithm



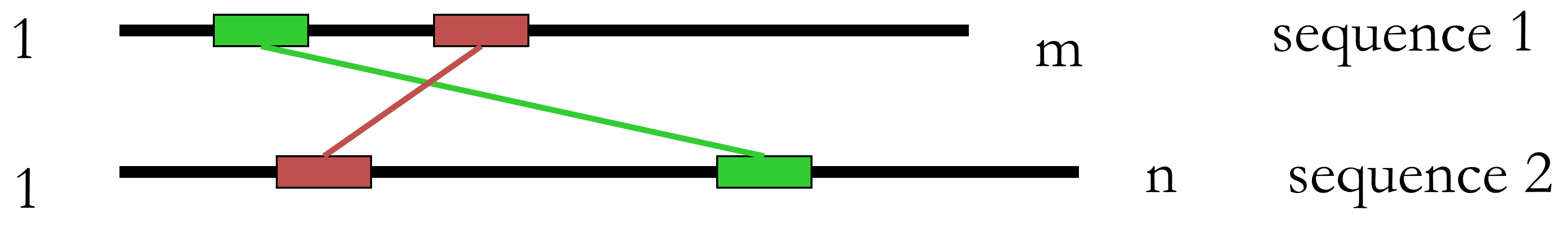
- The old algorithm requires O(mn) and is too slow.
- Human v.s. mouse: $3\text{x}10^9\text{x}3\text{x}10^9=9\text{x}10^{18}$

Similarity Search



- Most similarities (local alignments) are very short relative to the genomes.

Similarity Search



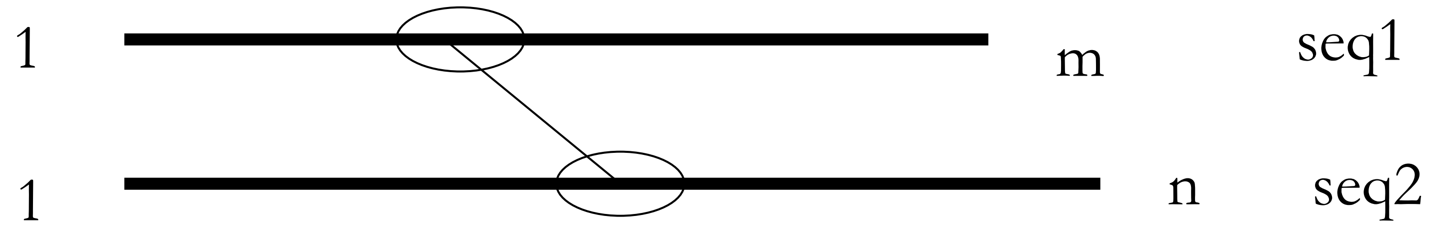
- For every pairs of (i, j), build a local alignment around it.
  - O(mnT)
  - Not better than Smith-Waterman.
- But this leads to an important idea…

Main Idea

- Most pairs of (i, j) are useless. We only want to try local alignments on the "promising" pairs of (i, j).
- In the context of sequence similarity search in bioinformatics, these "promising" pairs are called "seeds" or "hits".
- We need
    - a proper definition of hits.
    - some efficient way to enumerate the hits faster than trying every pair of (i, j).

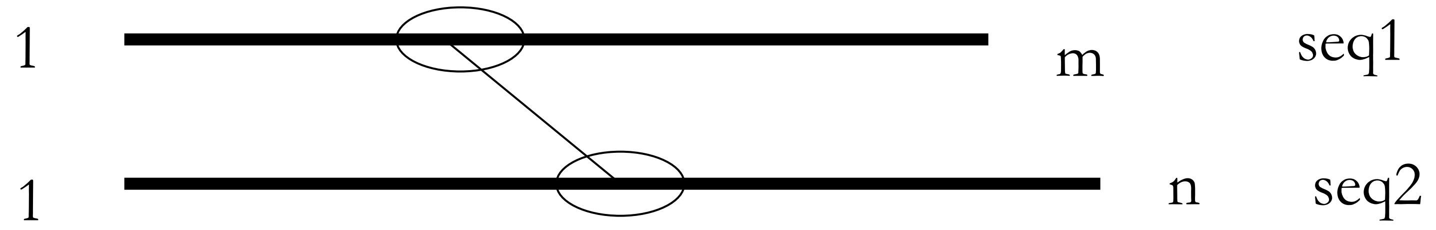
BLAST Uses Short Consecutive Match as Hits

```
GCNTACACGTCACCATCTGTGCCACCACNCATGTCTCTAGTGATCCCTCATAAGTTCCAACAAAGTTTGC
|| |||||  | |||  ||||   ||    ||||||||||||||||||| | |||||||| |  | |||||
GCCTACACACCGCCAGTTGTG-TTCCTGCTATGTCTCTAGTGATCCCTGAAAAGTTCCAGCGTATTTTGC
```



BLAST Uses Short Consecutive Match as Hits

```
GCNTACACGTCACCATCTGTGCCACCACNCATGTCTCTAGTGATCCCTCATAAGTTCCAACAAAGTTTGC
|| |||||  | |||  ||||   ||    |||||||||||||||||| | |||||||| |  | |||||
GCCTACACACCGCCAGTTGTG-TTCCTGCTATGTCTCTAGTGATCCCTGAAAAGTTCCAGCGTATTTTGC
```



- Majority of (i,j) are random and probability of generating a random hit is small.
- For length-k seed, time complexity becomes $O(4^{-k}mnT)$
- By default, BLAST used k=11.
- What's the speed up factor for k=11?

The Idea behind Seeding

- A true similarity has a high chance of being hit.
- A random pair (i, j) has low chance of being hit.
- Thus, if we use hit to filter (i, j), we will
    - Detect most true similarities.
    - Not wasting time on random pairs of (i, j).

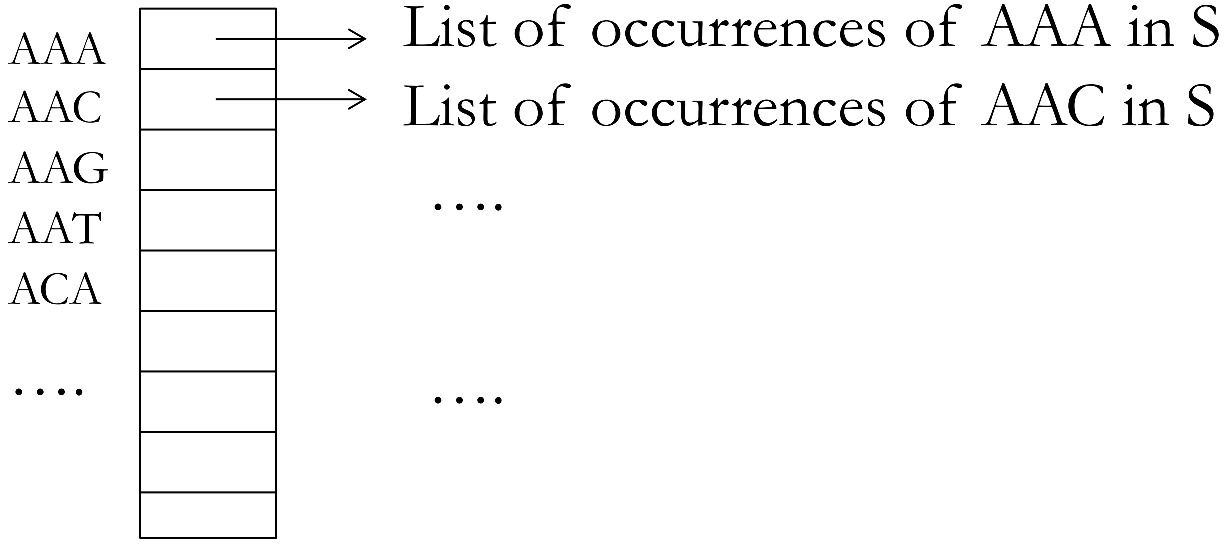
The Data Structure for Finding Hit?

- for each *k*-mer, index table to remember all its occurrences in S.
- for each *k*-mer of T, find its hits in the index table.
- The index table can be a trie or a hash table.

AA	0, 6
AC	
AG	
AT	1
CA	
CC	
CG	
CT	3
GA	
GC	
GG	
GT	
TA	5
TC	2
TG	
TT	4

```
S:    AATCTTAA
      01234567

T:    GAACTTA
```

The Data Structure for Finding Hit?


AAA
AAC
AAG
AAT
ACA
….
List of occurrences of AAA in S
List of occurrences of AAC in S
….
….

Space complexity?

Overall runtime

- Build the index using S: O(n) time.
- Find matches between the index and sequence T: O(m) time to scan T, plus we need to examine all of the N hits found. Let t be the examination time. Then
  O(m+Nt).
- Overall runtime: O(n+m+Nt).
- The term Nt is the most expensive part. Indexing overhead is small.
- In practice, most of the hits encountered are random hits.

Filtration can have multiple rounds

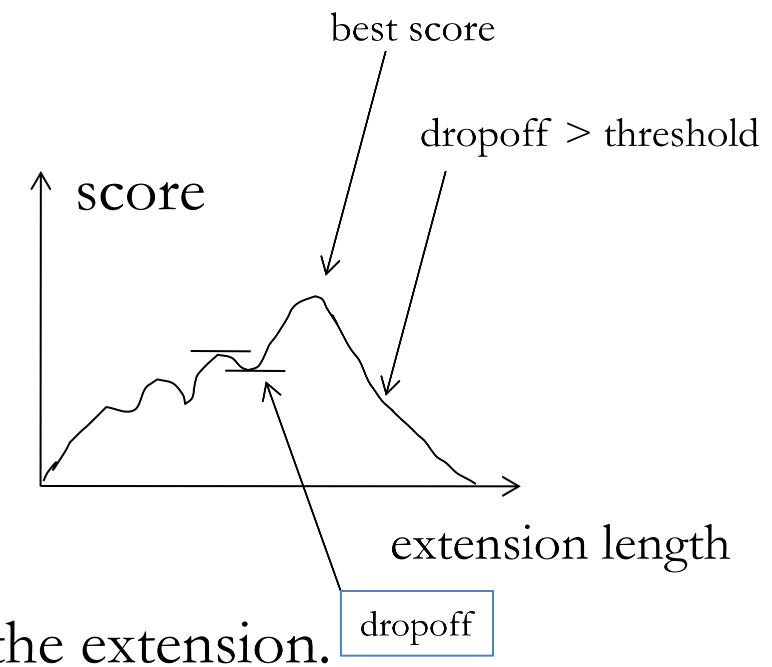
```
GCNTACACGTCACCATCTGTGCCACCAGCCATGTCTCTAGTGATCCCTCATGGTGGCCAACAAAGTTTGC
          |  |   |      || || ||||||||||| |||||| |      ||| |  | |||||
TGCCTACACACCGCCAGTTGTGTTCCTGCTATGTCTCTAGTTATCCCTGAAAAGTTCCAGCGTATTTTGC
```

- After finding a hit, instead of trying to build a local alignment directly, BLAST uses another round of filtration to determine if a hit is a "good" or "bad" hit.
- Quick search in both directions; if most symbols match, it's a good hit. Otherwise it's bad.
  - More precisely, use ungapped extension to find HSPs.
- If an HSP is above a certain score threshold, build a local alignment around it.

HSP extension

```
GCNTACACGTCACCATCTGTGCCACCAGCCATGTCTCTAGTGATCCCTCATGGTGGCCAACAAAGTTTGC
          |  |   |      || || ||||||||||| |||||| |      ||| |  | |||||
TGCCTACACACCGCCAGTTGTGTTCCTGCTATGTCTCTAGTTATCCCTGAAAAGTTCCAGCGTATTTTGC
```

```
for k from 0 to …
    score += sc(S[i+k],T[j+k])
for k from 1 to …
    score += sc(S[i-k],T[j-k])
```



- But when to stop?
- Score will increase and decrease during the extension.
- Extension stops when drop off greater than threshold.

HSP Extension

- How long will the extension continue after reaching best score?
- Assumptions:
  - After reaching best score, sequence becomes random.
  - match=1 and mismatch=-1
- Expected score on each additional base is -0.5.
- If dropoff=k, then after 2k bases, the expected dropoff will reach k.
- Conclusion: Not too long.

Example of missing a target

- Fail:

```
GAGTACTCAACACCAACATTAGTGGGCAATGGAAAAT
|| ||||||||| |||||| | ||||||   ||||||
GAATACTCAACAGCAACATCAATGGGCAGCAGAAAAT
```

- Dilemma
  - **Sensitivity** – needs shorter seeds
    - the success rate of finding a homology
  - **Speed** – needs longer seeds
    - Mega-BLAST uses seeds of length 28.

PatternHunter uses “spaced seeds”

- 111*1**1*1**11*111 (called a spaced seed)
  - Eleven required matches (**weight**=11, **length** = 18)
  - Seven “don’t care” positions

```
GAGTACTCAACACCAACATTAGTGGCAATGGAAAAT…
|| ||||||||| ||||| || |||||   ||||||
GAATACTCAACAGCAACACTAATGGCAGCAGAAAAT…
       111*1**1*1**11*111
```

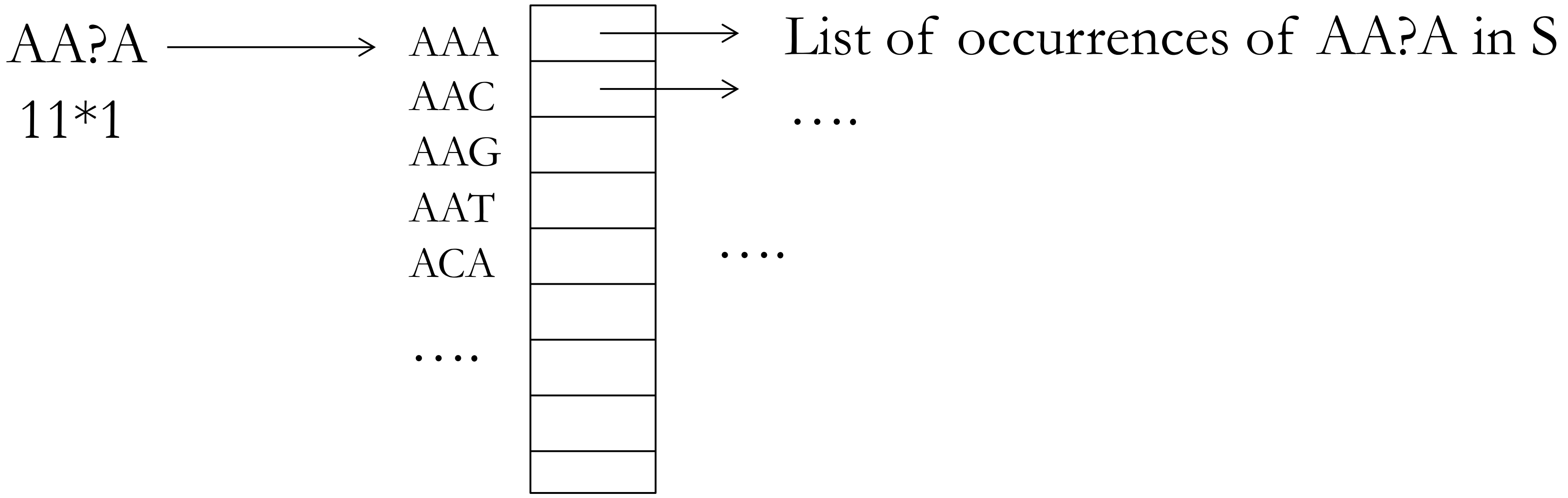
- Hit = all the required matches are satisfied.
- BLAST’s seed = 11111111111

Notes about the notation

- A homology/similarity region's actual sequences do not matter, the match/mismatch matters.
- Therefore, a region is often denoted by a binary 0-1 sequence, `11011111001110111011111`
- A hit is then as follows:
- `11011111001110111011111`
- `   111*1**1*1**11*111`

The Data Structure for Finding Hit

- The same as consecutive seed. Except that now we have a length *l*, weight *w* seed. E.g. 11*1.
  - Each *l*-mer, take the *w* letters out and put in index table.
- The index table can be a hash table.



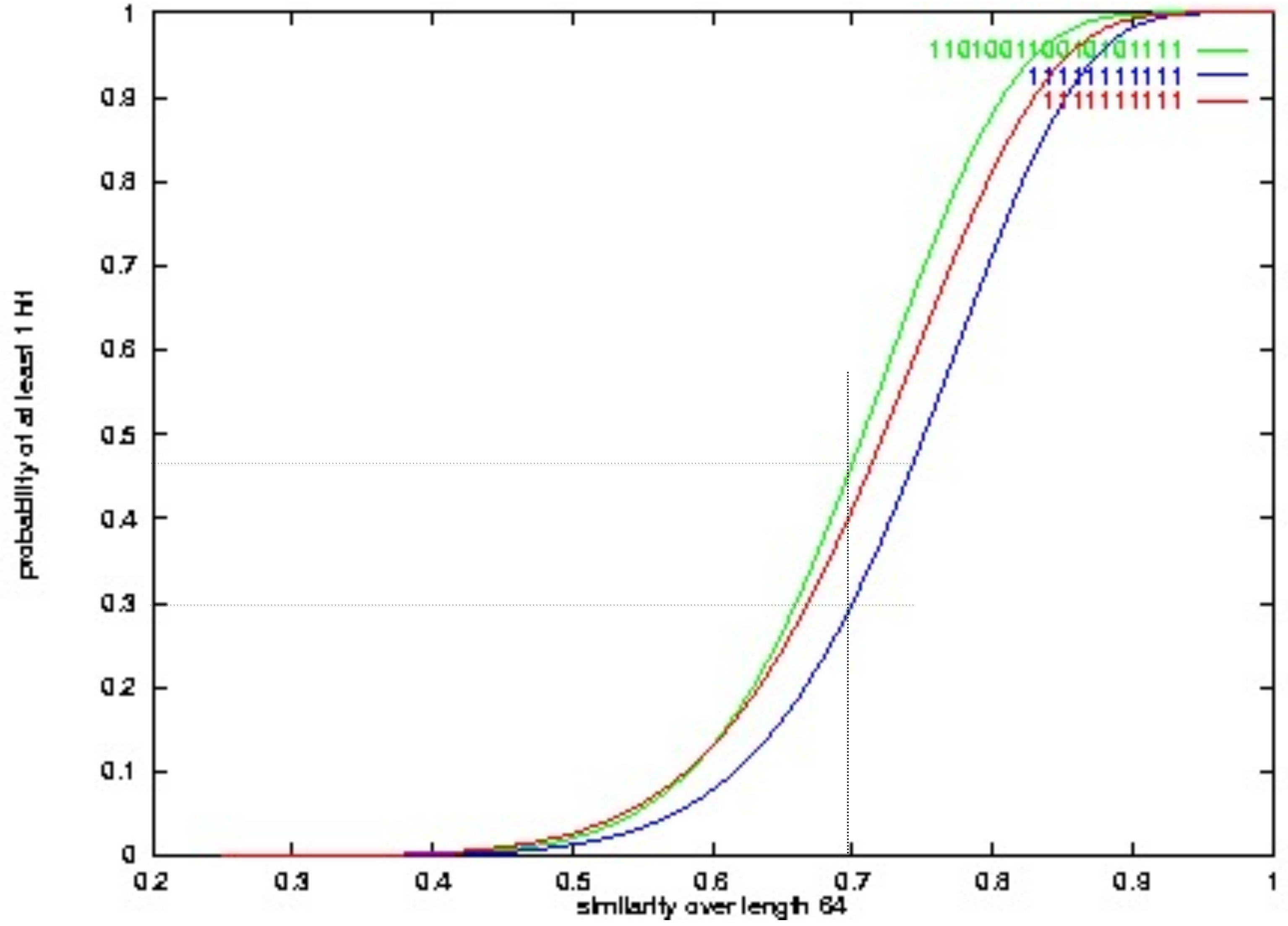
Time Complexity Comparison

- Lemma: for random sequence S and T with lengths m and n, the expected number of hits for weight w, length l seed is

$$(m-l+1)(n-l+1)4^{-w}$$

- Because usually l is much shorter than S and T, this is approximately $4^{-w}mn$
- That is, the expected number of hits in random regions only depends on the weight, but not the shape of the seed. So does the running time.
- So, speed-wise, spaced seed is similar to consecutive seed.
- What about the sensitivity?

Simulated sensitivity curves

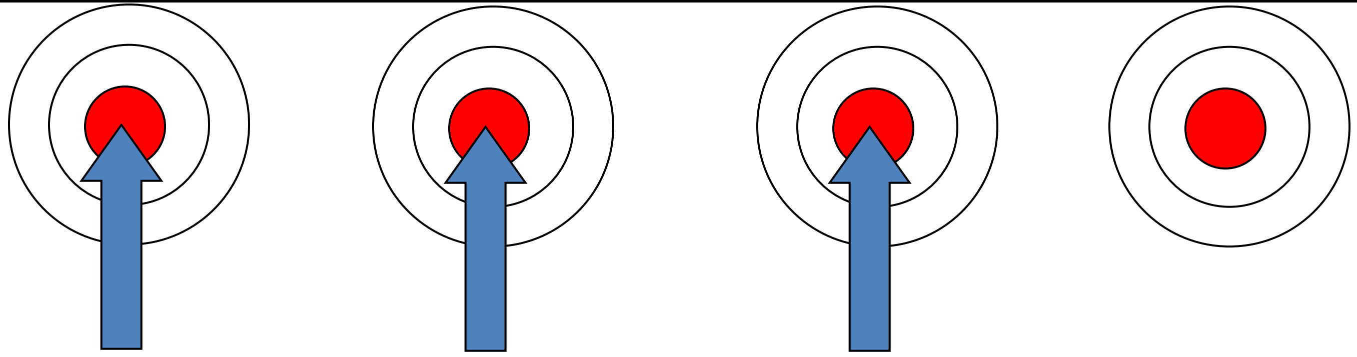


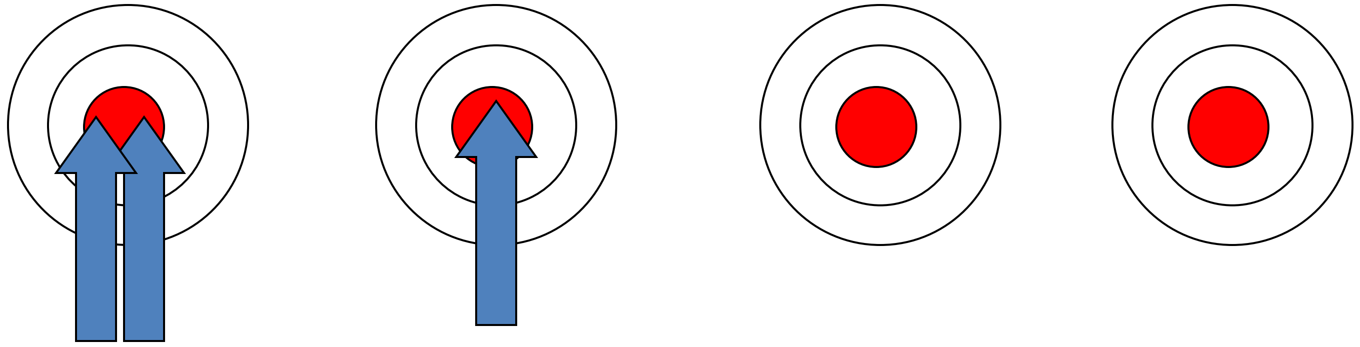
Why spaced seeds are better?

```
TTGACCTCACC?
|||||||||||?
TTGACCTCACC?
11111111111
 11111111111
```

```
CAA?A??A?C??TA?TGG?
|||?|??|?|??||?|||?
CAA?A??A?C??TA?TGG?
111*1**1*1**11*111
 111*1**1*1**11*111
```

- BLAST’s seed usually uses more than one hits to detect one homology (redundant)
- Spaced seeds uses fewer hits to detect one homology (efficient)





PH’s seed does not overlap much

- PH’s seed do not overlap heavily when shifts:

```
111*1**1*1**11*111
 111*1**1*1**11*111
  111*1**1*1**11*111
   111*1**1*1**11*111
    111*1**1*1**11*111
     111*1**1*1**11*111
      111*1**1*1**11*111
         ......
```

- The hits at different positions are independent.
- The probability of having the second hit is $3*p^6 + \dots$
  - compare to BLAST’s seed $p + p^2 + p^3 + p^4 + \dots$

Lossless Filtration

- When seeds are short enough and HSP similarity is high enough, lossless filtration is also possible.
- For example, seed 111 can guarantee to match when a sufficiently long HSP has similarity 66.7%.
- Proof: To fail being hit by 111, the HSP must have a mismatch in every 3 adjacent positions.
- On the other hand, 110110110…, which has 66.6% similarity, will fail the seed 111.

Lossless Filtration

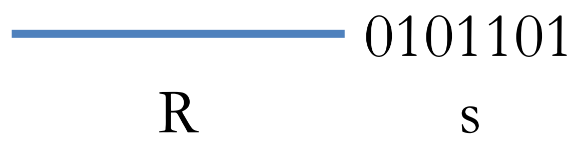
- Now consider spaced seed 11*1.
- Claim: For any $\epsilon > 0$, seed 11*1 will hit every sufficiently long region with similarity $0.6 + \epsilon$.

Proof

- Suppose there is a sufficiently long region not hit by 11*1.
- We can break the region into blocks of 1^a0^b. Besides the last block that can have at most three 1s, each of the other blocks is one of the following three cases:
  - 10^b for b>=1
  - 110^b for b>=2
  - 1110^b for b>=2
- In each block, similarity <= 0.6.
- So the long region's similarity is $< 0.6 + \epsilon$.

Compute a Seed's Sensitivity

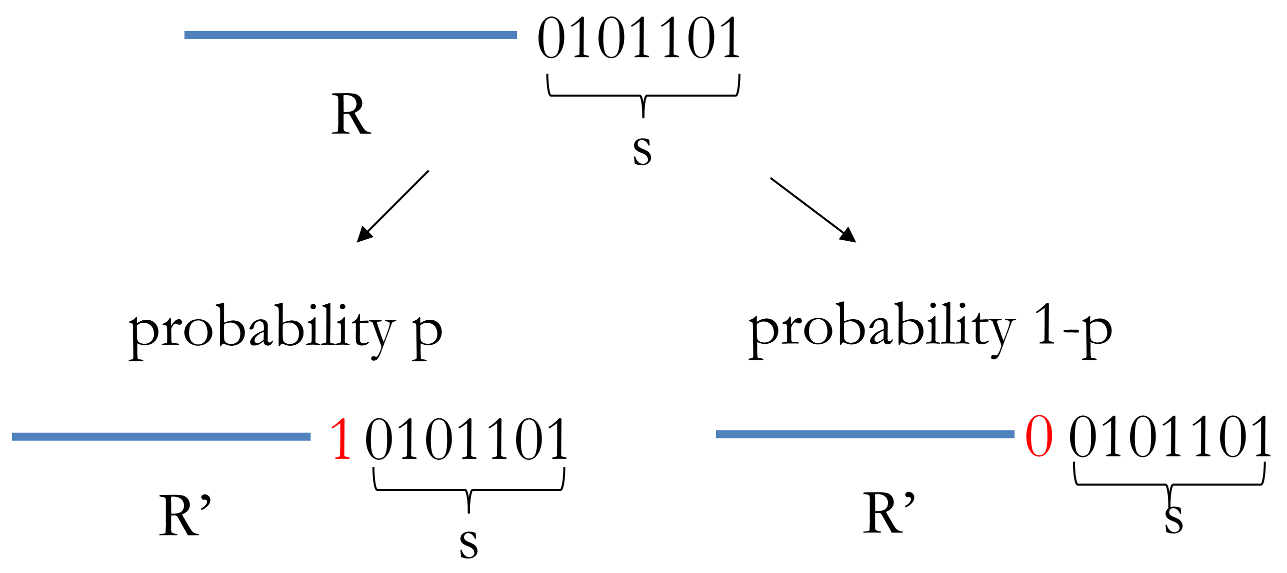
- R: A probabilistic distribution of HSP, Pr(R[i]=1) = p;
- We want Pr(length-n R is hit by a seed x). |x|=k
- s: A length-k 0-1 string.
- Rs: The concatenation of R and s.
- Let $D[i,s]$ be the probability Rs is hit by x for a length-i R.



- By total probability law, answer is $\sum_s (p(s) \cdot D[n-k,s])$. Note the summation is over all length k binary string s, and $p(s) = p^{\#1\ in\ s}(1-p)^{\#0\ in\ s}$

Dynamic Programming

- Case I: s is hit by x. Then $D[i, s] = 1$.
- Case II: s is not hit by x:



R' is the length-(i-1) distribution. s' is the length-(k-1) prefix of s.

$$D[i, s] = p \cdot D[i-1, 1s'] + (1-p) \cdot D[i-1, 0s']$$

Dynamic Programming

```
Initialize D[0,s]
For i from 1 to n
    for every binary string s
        if s is hit by x
            $D[i,s] = 1$
        else
            $D[i,s] = p \cdot D[i-1,1s'] + (1-p) \cdot D[i-1,0s']$
Return $\sum_s p(s) \cdot D[n-k,s]$
```

Here $p(s) = p^{\#1\ in\ s}(1-p)^{\#0\ in\ s}$.

Time complexity $O(2^k n)$

More efficient algorithm exists (not lectured here). $O(2^{\#0\ in\ s} n)$.

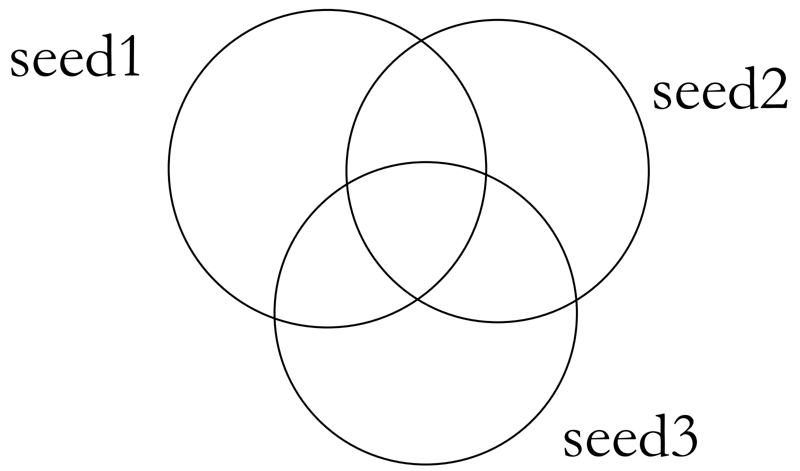
The “algorithm” to select the optimal spaced seed

- Enumerate all spaced seeds with weight 11 and no longer than 18, calculate the sensitivity of each, and output the one with the highest sensitivity.
- This is the ONLY known algorithm that guarantees the finding of optimal seed.
- Many heuristics exist to find suboptimal seeds.

Multiple Seeds – PatternHunter II:

Multiple Spaced Seeds

- Seeds with different shapes can detect different homologies.
  - Some seeds *may* detect more homologies than others. This leads to the use of optimized spaced seed.
  - Can use several seeds simultaneously to hit more homologies
    - Approaching 100% sensitive homology search



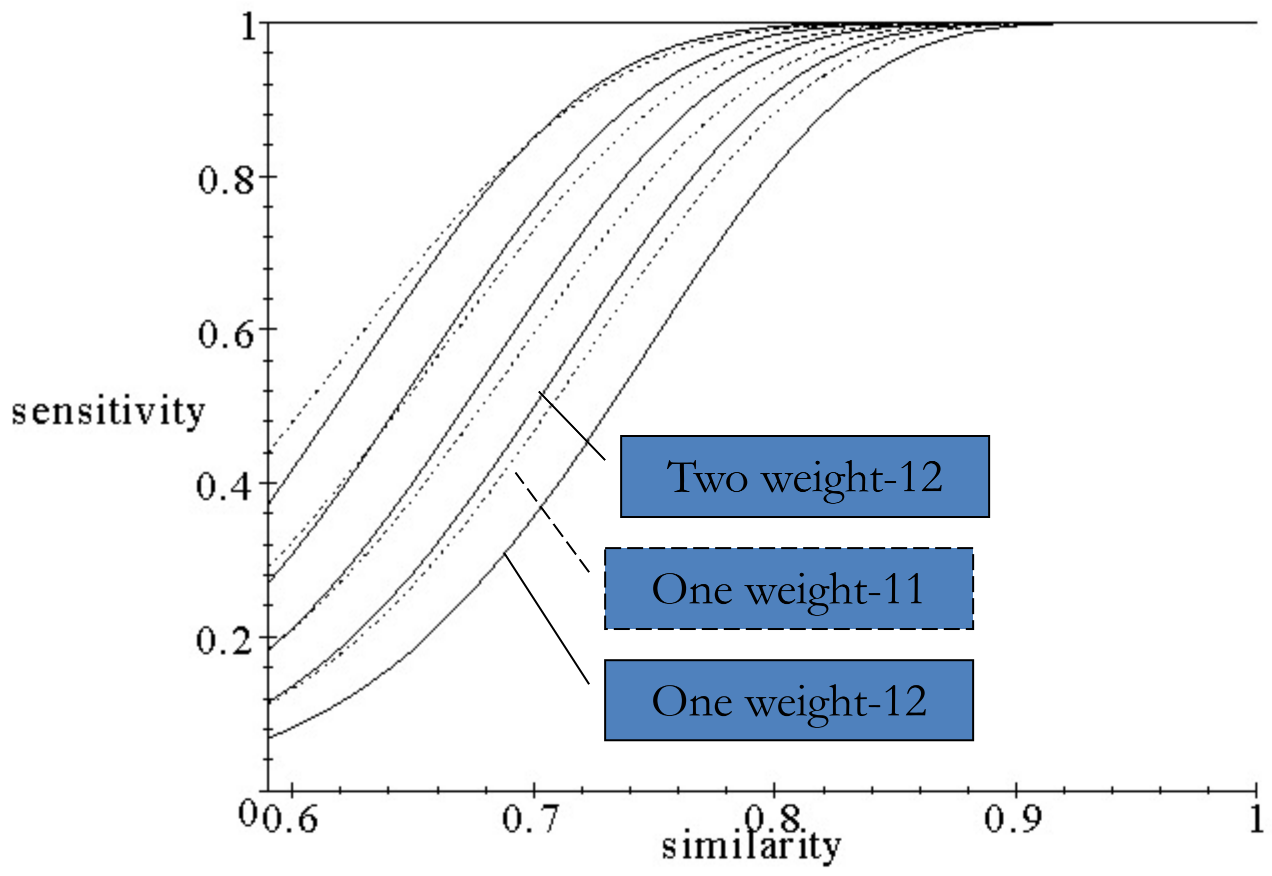
Multiple Seeds Example

(homology identity = 0.7, homology length=64)

```
111*11**1*11*1*111
1111***1***1**11*1*111
11**11*1**1*1***11*111
111*1***1111**1***11*1
```

- To use multiple seeds, one only needs to search multile times with different seeds, and combine results. Of course, you can search with them simultaneously.
- In either case, this slows down approximately k times if k seeds are used.
- Is it worth it? How does it compare with using one shorter seed?

Simulated sensitivity curves:



- Solid curves: Multiple (1, 2, 4, 8, 16) weight-12 spaced seeds.
- Dashed curves: Optimal spaced seeds with weight = 11, 10, 9, 8.

- Typically, "Doubling the seed number" gains better sensitivity than "decreasing the weight by 1".

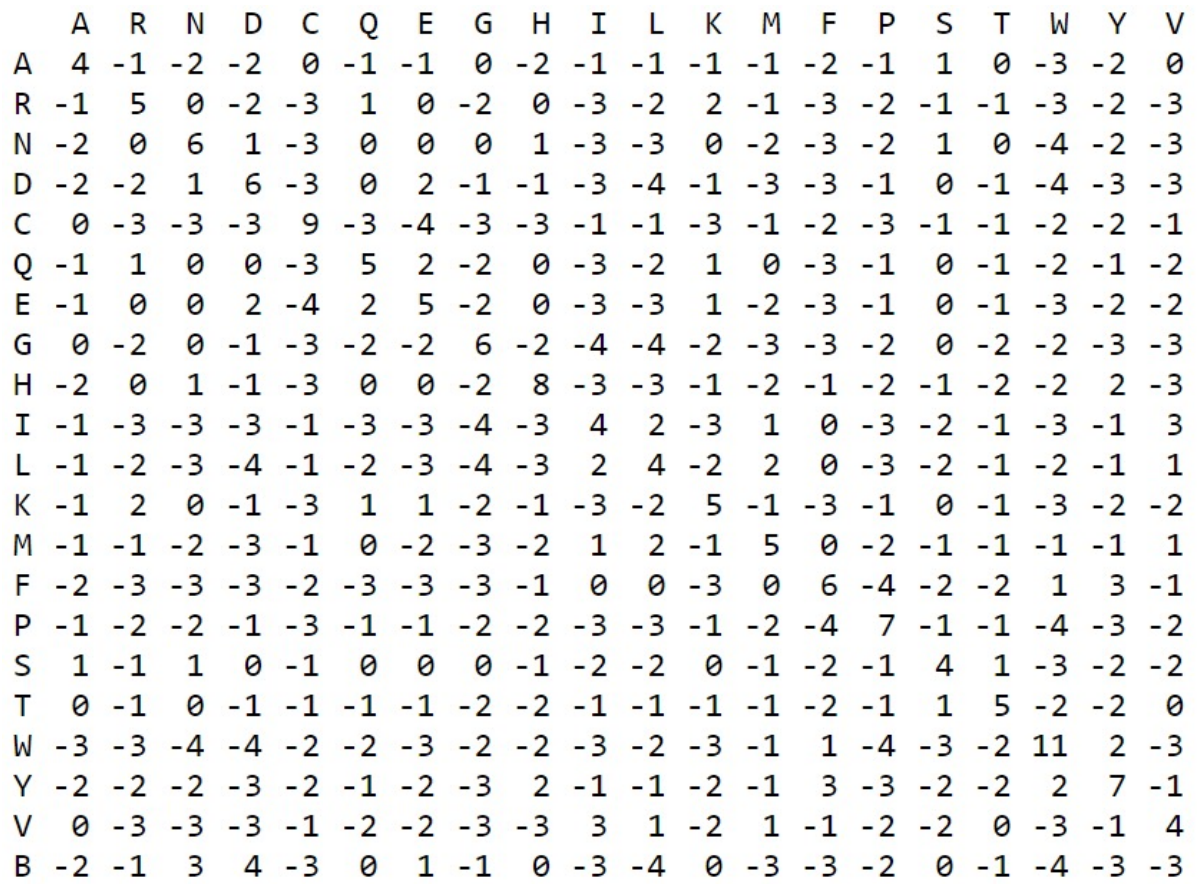
Seeding for Proteins - BLASTP

- With nucleotides, we're requiring *k* positions with exact matches.

- For proteins, that's not really reasonable: some amino acids mutate to another one very often.
- So BLASTP looks for 3- or 4-letter protein sequences that are "very close" to each other, and then builds matches from them.
- Where very close ➔ total BLOSUM score in the short window is at least +13 (or +11 for 3 mer).

Excercise

- For query RRR, threshold 11, what are the other 3-mers that can generate hits?



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3

How to implement that?

- With BLASTP:
  - Build an automaton that reflects all string close to short strings in T (the short sequence)
  - Scan S (the longer sequence), looking for matches.
- We do not study the classic ways to match multiple patterns efficiently. If interested, you can read at https://en.wikipedia.org/wiki/Aho%E2%80%93Corasick_algorithm

A Simpler Way

- There is another way:

1) For every 3-mer, find all "neighboring" 3-mers that, score at least +11 (or whatever). Build these into a data structure NeighborList.
2) Build a hash table H for S of its 3-mers, just like for the nucleotide case
3) For every 3-mer *x* in T, retrieve all neighbors from NeighborList. For each neighbor, query H to find hits in S.

NeighborList is a small structure: there are only 8000 3-mers.

Which sequence to index?

- That's actually a tough question.

- Here's a typical scenario:
- S is the human genome (length n)
- P_1 is a short protein sequence (length m_1)
- P_2 is another short sequence (length m_2)

- If we're smart, build an index for S, *once*, and then look up the short sequences in it.
- Added time for P_2 is more like $O(m_2)$, not $O(n+m_2)$.

More on indexing

- But memory is a concern:
- Indexing the human genome is expensive!
- Oh, wait. No, it isn't, not anymore… you probably should index the longer sequence.
- BLASTN (1990) indexes the query, not the database.
- BLAT (2000) indexes the database, not the query.

- BLASTP also indexes the query.

Extensions to this idea

- Two-hit BLAST:
- Require two seeds (probably shorter) that are nearer than k from each other, and base the alignment on their enclosing box.
- Potentially even fewer false positives, but one has to use shorter seeds. There's quite a tradeoff here.

Wrap-up

- Local alignment slow when sequences are large
- Use 11 consecutive matches as hits
  - How these hits are found efficiently
  - What to do after hits are found
- Spaced seeds better
  - How sensitivity is computed and how optimal seed is found
  - How hits are found for spaced seed
- Multiple spaced seed.
- Protein seeds.
- Two hits.