Seeding Methods in Homology Search
Heuristic algorithms

• Heuristic algorithms are basically “algorithms for the real world”.

• They acknowledge that the real world is complicated, and you can’t always get what you want.

• A heuristic algorithm promises to probably run fast, and to probably solve the problem pretty close to right.

• This is much vaguer than a theorem can guarantee.
A homology between mouse and human genomes

Smith-Waterman is the most accurate method.
Time complexity: $O(mn)$.  

Smith-Waterman is the most accurate method. 
Time complexity: $O(mn)$. 

Smith-Waterman is the most accurate method. 
Time complexity: $O(mn)$. 

Smith-Waterman is the most accurate method. 
Time complexity: $O(mn)$.
Similarity Search

• Needs to compare every pair of positions
  • $O(mnT)$
  • Human v.s. mouse: $3 \times 10^9 \times 3 \times 10^9 = 9 \times 10^{18}$
  • Not feasible!
• Modified Smith-Waterman still $O(mn)$. 
Local Alignment Algorithm

Diagram showing the alignment of sequences S and T.
BLAST finds “hits” then extends

- Time complexity becomes $O(4^{-11}mnT)$
- The homology grows from the 11 consecutive matches therefore they are called the seed
- More precisely these are called hits. A seed often is used to refer to the way these 11 positions are selected.
- What if we drop seed length by 1?
After Finding Hits

• Instead of trying to build an 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.

• Build a local alignment around HSPs.
  • More heuristics can be used. Not to be discussed here.
HSP extension

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.
The Data Structure for Finding Hit?

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

```
AAA  → List of occurrences of AAA in S
AAC  → ....
AAG  
AAT  
ACA  → ....
....  
....  
```

Space complexity?
Overall runtime

• Build the index: $O(n)$ time.

• Find matches between the index and sequence $T$: $O(m)$ time to scan $T$, plus we need to extend all of the $N$ hits found: $O(m + Nt)$.

• Overall runtime: $O(n + m + Nt)$.

• The term $Nt$ is the most expensive part. Indexing overhead is small.
Example of missing a target

• Fail:
  GAGTACTCAACACACATTAGTGGAACATGGAAAAT
  || |||||||| | |||||   |||||
  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 \( \text{weight}=11, \text{length} = 18 \)
  - Seven “don’t care” positions

---

<table>
<thead>
<tr>
<th>GAGTACT</th>
<th>CAACACCACGCGACTTAGGCGCAATGGGAAAAT...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>GAATACT</td>
<td>CAACAGCGACGACACTAATGGCGCAGCAGAAAAT...</td>
</tr>
<tr>
<td></td>
<td>111<em>1**1</em>1**11*111</td>
</tr>
</tbody>
</table>

- 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.
A important lemma

- 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.

- What about the sensitivity?
A trivial comparison between spaced and consecutive seed

- Consider weight-3 seeds 111 and 11*1.
- To fail seed 111, we can use
  - 110110110110…
  - 66.66% similarity
- But we can prove, seed 11*1 will hit every sufficiently long region with >61% similarity.
Proof

• Suppose there is a sufficiently (infinitely) long region not hit by 11*1.

• We can break the region into blocks of $1^a0^b$. Besides the last block, the other blocks have the following few cases:
  • $10^b$ for $b>=1$
  • $110^b$ for $b>=2$
  • $1110^b$ for $b>=2$

• In each block, similarity $\leq 3/5$.

• So the long region’s similarity is $< 61\%$. 
Simulated sensitivity curves
Why spaced seeds are better?

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

- Indeed, under the condition that there is one hit in a length 64, 70% similar homology, the average number of hits in that region is
  - 2.0 for PH’s weight-11 seed
  - 3.6 for contiguous weight-11 seed.
PH’s seed does not overlap heavily

• 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
111*1**1*1**11*111

... ...

• The hits at different positions are independent.
• The probability of having the second hit is 3*p^6 + …
  • compare to BLAST’s model p + p^2 + p^3 + p^4 + …
A dynamic programming algorithm to compute sensitivity

- R[1..n]: A random similarity, Pr(R[i]=1) = p;
- We want Pr(R is hit by a seed x), |x| = k
- DP[i,s] denotes Pr(R[1..i] is hit | R[1..i] ends with s), here s is a length |x| binary string.
  
  
  \[
  DP[i,s] = \begin{cases} 
  1; & s \text{ is hit by } x. \\
  p \times DP[i-1,(1s')] + (1-p) \times DP[i-1,(0s')]; & \text{otherwise.}
  \end{cases}
  \]

  Here s’ is the first k-1 characters of s.

- Finally, the hit probability is
  \[
  \sum_s p(s) \times DP[n,s] = \sum_s p^{\# \text{of 1 ins}} (1-p)^{\# \text{of 0 ins}} \times DP[n,s]
  \]

- Time complexity O(n*2^{|x|}). Better algorithms exist.
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:
Observations re. spaced seeds

• Seeds with different shapes can detect different homologies.

• Two consequences:
  • Some models *may* detect more homologies than others
    • More sensitive homology search
    • PatternHunter I
  • Can use several seed models simultaneously to hit more homologies
    • Approaching 100% sensitive homology search
    • PatternHunter II
The seeds obtained under a simple homology distribution

(homology identity = 0.7, homology length = 64)

111*11**1*11*1*111,
1111***1***1**1*111*111,
11**11*1**1*1***11*111,
111*1***1111**1***11*1,

......

......
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”.
Coding region seeds

• The first two bases of a codon is more conserved than the third base.

• Coding regions matches have patterns like 110110…….

• The seeds trained under a coding region homology distribution are called the coding region seeds.
  • PHII’s default seeds were trained under a simple distribution (0.8, 0.8, 0.5).
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 $\Rightarrow$ 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?
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
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.
• The research for better seeds is still going on.