Computational Systems Biology

… Biology X – Lecture 6…

Bud Mishra

Professor of Computer Science, Mathematics, & Cell Biology

Evolution by Mutation

Evolution by Mutation

- Mutant gene or DNA sequence:
  - Substitution, Insertion/Deletion, Recombination, Duplication, Gene Conversion
  - Spread through a population by genetic drift and/or natural selection and eventually is fixed in a species
- Nucleotide substitution can be divided into two classes:
  - **Transitions**: Substitution of a purine by purine (A, G) or a pyrimidine by a pyrimidine (C,T)
  - **Transversions**: The other types.
- More specific properties:
  - Frameshift mutation, nonsense mutation, synonymous or silent substitutions, non-synonymous or amino acid replacement substitution
  - Transposons, gene conversions, horizontal gene transfer.

Jukes-Cantor

- The nucleotide substitution occurs at any site with equal frequency, and that at each site a nucleotide changes to of the three remaining nucleotides with a probability of α per year.
- Probability of a change of a nucleotide to another is $r = 3\alpha$
- q_t = Proportion of identical nucleotides between two sequences

$$q_{t+1} = (1-2r)\, q_t + (2/3)\, r\, (1-q_t)$$

$$q(t) = 1 - \tfrac{3}{4}(1-e^{-8rt/3})$$

Kimura 2 Parameters

- The rate of transitional substitution per site per year = α
- The rate of transversional substitution per site per year = 2β
- Total substitution rate, $r = \alpha + 2\beta$
- R_t is the proportion of identical transition-type pairs AG, GA, CT, TC
- Q_t is the proportion of identical transversion-type pairs AG, GA, CT, TC

$$R(t) = (1/4)(1 - 2e^{-4(\alpha+\beta)t} + e^{-8\beta t})$$

$$Q(t) = (1/2)(1 - e^{-8\beta t})$$

Other Models

- Tajima & Nei:
  - Substitutions that seem to be rather insesnsitive to various disturbing factors.
- Tamura:
  - Takes into account varying GC content
- Hasegawa et al. (HKY)
- Rzhetsky & Nei
- Tamura & Nei

Matching and Alignment

Inexact Matching

- Example: Edit Distance Problem:
  - Edit distance between two biological sequences
  - May correspond to:
    - Evolutionary Distance
    - Functional Distance
    - Structural Distance

Edit Distance

- Simplest distance function corresponds to:

  EDIT DISTANCE

- Atomic Edit Functions:
  - Insertion AATCGG $\mapsto$ AATACGG
  - Deletion AATACGG $\mapsto$ AATCGG
  - Substitution AATCGG $\mapsto$ AATAGG
- A composite edit function

  $\simeq$ Function Composition of Atomic Edit Functions

Cost of a Composite Edit Function

- (Based on the cost or distance for Atomic Edit Functions)
- Given: Two strings S_1 and S_2

$$\text{Distance}(S_1, S_2) = \min \{ \text{cost}(E) \mid E(S_1) = S_2 \}$$

Where

E = composite edit function mapping S_1 to S_2.

Some Properties of Distance Function

- Assume:

  ($\forall$ e= Atomic Edit Function) cost(e) = cost(e^{-1})

  - Distance(S_1, S_2) = Distance(S_2, S_1) *Symmetric*
  - Distance(S_1, S_1) = 0
  - Distance(S_1, S_2) + Distance(S_2, S_3) $\geq$ Distance(S_1, S_3)

    Triangle Inequality

- Simplest Cost Function:

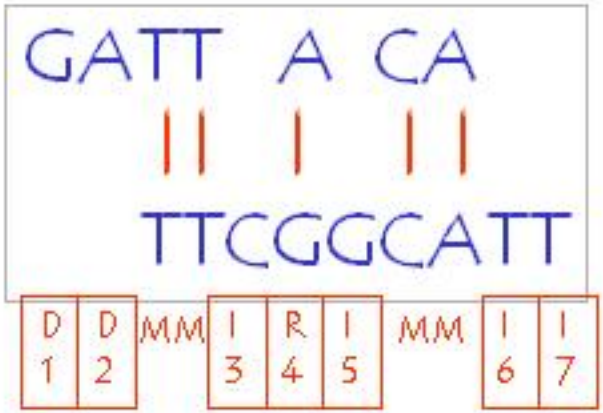
  - *Each atomic edit function is of <u>unit cost</u>.*

Edit Operations

- I: Insertion of a character into the first string S_1
- D: Deletion of a character from the first string S_1
- R: Replacement (or Substitution) of a character in the first string S_1 with a character in the second string S_2
- M: Matching (Identity)

Edit Transcript

Example:



- The complete edit function is described by an "edit transcript"
- EDIT TRANSCRIPT

  $= \sigma \in \{D, M, R, I\}^*$
- Example (in left):
  - Edit transcript = DDMIRIMMII
  - Edit Distance = 1+1+0+1+1+1+0+0+1+1=7

Levenshtein (or Edit) Distance

- Edit Distance between two strings S_1 and S_2 is defined as the minimum number of atomic edit operations – insertions, deletions (indels), and substitutions – needed to transform the first string into the second
- Optimal Transcript = An edit transcript corresponding to the minimum number of atomic edit operations of unit cost.

EDP

The Edit Distance Problem

is to compute

- the edit distance between two given strings, along with
- an optimal edit transcript that describes the [illegible] ment)

Dynamic Programming Calculation of Edit Distance

- Define:

$D(i,j) \equiv$ Min number of atomic edit operations needed to transform the first i characters of S_1 into the first j characters of S_2

$\equiv$ EditDistance($S_1[1..i]$, $S_2[1..j]$)

- $|S_1| = n \quad |S_2| = m$

$$\text{Distance}(S_1, S_2) = D(n,m)$$

- Dynamic Programming: 3 components:
  - Recurrence Relation
  - Tabular Computation
  - Traceback

Recurrence

- Base Relation:
  - $D(0,0) = 0$
  - $EditDistance(\lambda,\lambda) = 0$
- Recurrence Relations:
  - <u>In 1 coordinate:</u>

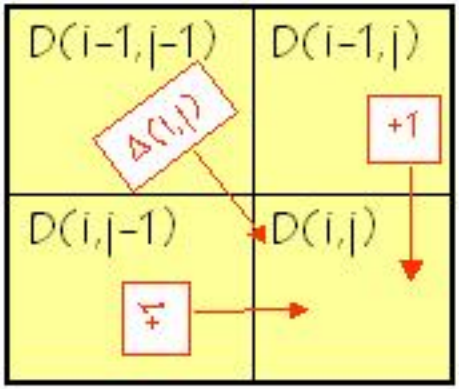
    $D(i,0) = D(i-1,0)+1$ ($S_1[i]$ deleted)

    $D(0,j) = D(0,j-1)+1$ ($S_2[j]$ inserted)
  - $EditDistance(S_1[1..i], \lambda) = i$ (i deletions)
  - $EditDistance(\lambda, S_2[1..j]) = j$ (j insertions)
  - <u>In both coordiates:</u>

    $D(i,j) = \min \{D(i-1,j) + 1,$ ($S_1[i]$ deleted)

    $\{D(i,j-1)+1,$ ($S_2[j]$ inserted)

    $\{D(i-1,j-1)+ \Delta(i,j)\}$ (substn or match)
  - $\Delta(i,j) = 1$, if $S_1[i] \neq S_2[j]$; 0 otherwise.



Efficient Tabular Computation of Edit Distance

- Recursive Implementation $\mapsto 2^{O(n+m)}$-time computation
- Bottom-up computation
  $(n+1)\times(m+1)$ distinct values for $D(i,j)$ to be computed
- Dynamic Programming Table of size $(n+1)\times(m+1)$
  - String S_1 corresponds to the rows (Vertical Axis)
  - String S_2 corresponds to the columns (Horizontal Axis)
- Fill out $D(i,0) \leftarrow$ First Column
- Fill out $D(0,j) \leftarrow$ First Row
- Fill out rows $D(i,j) \leftarrow$ Left-to-Right (increasing i)

The Algorithm

- for i=0 to n do
  - $D(i,0) \leftarrow i;$

  for j=0 to m do
  - $D(0,j) \leftarrow j;$

  for i=1 to n do
  - for j=1 to m do
    - $D(i,j) \leftarrow \min[\ D(i-1,j)+1,$
      $D(i,j-1)+1,$
      $D(i-1,j-1) + \Delta(i,j)]$

  □
- Time complexity = $O(nm)$

Example

$S_2 \rightarrow$

$\downarrow S_1$

		A	A	T	C	G	G
	0	1	2	3	4	5	6
T	1	1	2	2	3	4	5
A	2	1	1	2	3	4	5
G	3	3	2	2	3	3	4
G	4	4	4	3	3	3	3
C	5	5	5	4	3	4	4

Solutions with minimal edit distance =4

(sol.1) TA-GGC
AATCGG
1 23 4
(Edit Script= RMIRMR)

(sol.2) - -TAGGC
AATCGG -
1 2 3 4
(Edit Script = IIMRMMD)

Trace Back

- Extracting Optimal Edit Transcript:
- Set a pointer from:
  - Cell(i,j) → Cell(i,j-1), if $D(i,j) = D(i,j-1)+1$
    Horizontal Edge ⇒ I, Insertion
  - Cell(i,j) → Cell(i-1,j), if $D(i,j) = D(i-1,j)+1$
    Vertical Edge ⇒ D, Deletion
  - Cell(i,j) → Cell(i-1,j-1), if $D(i,j) = D(i-1,j-1)+\Delta(i,j)$
    Diagonal Edge ⇒ R, Substitution, if $\Delta(i,j)=1$
    M. Match, if $\Delta(i,j) = 0$.
- Optimal Edit Transcript can be computed in $O(n+m)$ additional time.

GAPS: The Scoring Model

- Basic operations:
  - Sequencing Errors or Evolutionary processes of Mutations and Selections
  - Substitution: Changes one base to another.
  - Gaps: Insertions or Deletions:

    Adds or removes a base.
- Total Score Assigned to an Alignment=
  - Sum of terms for each aligned pair of bases plus terms for each gap.

Total Score of an Alignment with Gaps

- Total Score Assigned to an Alignment
  - Corresponds to log of the
  - Relative likelihood that the two sequences are related compared to being unrelated.
- Assumptions:
  - Mutations or Sequencing Errors at different sites in a sequence occur independently.

Substitution Matrices

- Notation: x and $y \equiv$ Pairs of sequences,

$$|x| = n \text{ and } |y| = m.$$

$$x, y \in (A+G+C+T)^*$$

  - x_i = i^{th} symbol in x
  - y_j = j^{th} symbol in y
- Random Model, R:

$$P(x,y|R) = \prod q_{xi} \prod q_{yj}$$

  - q_a = probability that the letter "a" occurs independently at a given site.

Random Model vs. Alternative Model

- Alternative Model, M:
- $P(x,y \mid M) = \prod p_{x_i, y_i}$
  - p_{ab} = Probability that the letters "a" and "b" have each been derived independently from some common letter.
- Log-Odds Ratio (LOD):

$$s(a,b) = \ln (p_{ab}/q_a q_b)$$

- $P(x,y|M)/P(x,y|R) = \prod (p_{x_i y_i}/ q_{x_i} q_{y_i})$

$$= \prod \exp[s(x_i,y_i)] = \exp[\sum s(x_i, y_i)]$$

Score

- Score = ln [P(x,y|M)/P(x,y|R)]

  $= \sum s(x_i, y_i) = s(x,y)$
- Score Matrix or Substitution Matrix:

	A	T	C	G
A	2	-1	-1	-1
T	-1	2	-1	-1
C	-1	-1	2	-1
G	-1	-1	-1	2

- Blossum 50
- PAM
  - (Point Accepted Mutation)

1-PAM Matrix

- Let M be a probability transition matrix. $M_{ab} = Pr(a \Leftrightarrow b)$,
  - a, b = chracters
- p_a = Pr("a" occurs in a string)
- f_{ab} = The number of times the mutation a $\Leftrightarrow$ b was observed to occur.
- $f_a = \sum_{a \neq b} f_{ab}$ & $f = \sum f_a$.
- K = 1-PAM Evolutionary distance
  - "The amount of evolution that will change 1 in K characters on average."

1-PAM Matrix

- $m_a = f_a / (Kf\, p_a)$,
  $M_{aa} = 1 - m_a$, $M_{ab} = f_{ab} / (Kf\, p_a) = (f_{ab}/f_a)\, m_a$
- α-PAM Matrix = M^α
- $M^* = \lim_{\alpha \leftarrow \infty} M^\alpha$
- $\text{Score}_\alpha(a,b) = 10 \log_{10} M^\alpha_{ab}/p_b$
- Sequence comparison with 40 PAM, 120 PAM & 250 PAM score functions…

Gaps:

- g = length of a gap,

  exponentially distributed $\sim \mathrm{Exp}(\lambda)$

$$f(g) = \lambda e^{-\lambda g}$$

- $P(g) = f(g) \prod q_{xi}$

$$\ln P(g) = -\lambda g + \ln \lambda + \text{sum} \ln q_{xi}$$

$$= -d - (g-1)\, e$$ (Affine Score Model)

  - d = Gap-open Penalty
  - e = Gap-Extension Penalty

Multiple Sequence Alignment

Multiple Sequence Alignment

- Defn: Given strings $S_1, S_2, \ldots S_k$ a multiple (global) alignment maps them to strings $S'_1, S'_2, \ldots, S'_k$ (by inserting chosen spaces) such that

1. $|S'_1| = |S'_2| = \cdots = |S'_k|$, and
2. Removal of spaces from S'_i contracts it to S_i, for $1 \leq i \leq k$.

Value of a Multiple Global Alignment

- The sum of pairs (SP) value for a multiple global alignment $\mathcal{A}$ of k strings is the sum of the values of all $C_{k,2}$ pairwise alignments induced by $\mathcal{A}$.
- Given: Two strings S_1 and S_2. The expanded strings S'_1 and S'_2 correspond to a pairwise alignment.
- $\delta(x, y)$ = distance between two characters x and y

  $= 1$, if $x \neq y$ and 0, if $x = y$.
- $\delta(x,-) = \delta(-,y) = 1$.
- $\text{Distance}(S'_1, S'_2) = \sum_{i=1}^{l} \delta(S'_1[i], S'_2[i])$,

  where $l = |S'_1| = |S'_2|$.

Optimal Global Alignment

- An optimal SP(global) alignment of strings $S_1, S_2, \ldots, S_k$ is an alignment that has a minimum possible sum-of-pairs value for these strings among all possible multiple sequence alignments.

Generalization of DP

- Assume $|S_1| = |S_2| = \cdots = |S_k| = n$.
- The generalized k-dimensional DP table has $(n+1)^k$ entries.
- Each entry depends on $2^k - 1$ adjacent entries.
  - $D(i_1, 0, \ldots, 0) = i_1$
  - $D(0, i_2, \ldots, 0) = i_2$

    $\vdots$
  - $D(0, 0, \ldots, i_k) = i_k$
  - $D(i_1, i_2, \ldots i_k) = \min_{\emptyset \neq \mathcal{S} \subseteq \{1..k\}} [$

    $D[\ldots, i_l - 1, \ldots]_{l \in \mathcal{S}}$

    $+ \sum_{l \neq m \in \mathcal{S}} \delta(i_l, i_m) + |\mathcal{S}| \times (n - |\mathcal{S}|)$

    $]$

Complexity

- The time and space complexity of the generalized DP solution of the multiple alignment problem is = $O((2n)^k)$
- Theorem: *The optimal SP alignment problem is NP-complete.*
- In the worst-case, one cannot expect to do much better unless *P=NP*.

P-Time Heuristics

- A Polynomial Time Approximate Algorithm for Multiple String Alignment:
- Assumption about the distance function:
  - Triangle Inequality:

    $\forall_{\text{chars}, x, y, z}\ \delta(x,z) \leq \delta(x,y) + \delta(y,z)$
  - $\forall_{\text{char}, x}\ \delta(x, x) = 0$
- $D(S_1, S_2)$

  $\triangleq$ Value of the min. global alignment of S_1 & S_2.

Algorithm

- Input: $\mathcal{T} = \{S_1, S_2, \ldots, S_k\}$
- Step 1: Find $S_1 \in \mathcal{T}$ that minimizes

$$\sum_{S \in \mathcal{T} \setminus \{S1\}} D(S_1, S)$$

  - Time Complexity = $O(k^2 n^2)$
    - $\mapsto C_{k,2}$ DP each taking $O(n^2)$ time
  - Call the remaining strings $S_2, \ldots, S_k$

The ith Step

- Step i: Assume $S_1, \ldots S_{i-1}$ have been aligned as $S'_1, \ldots, S'_{i-1}$
- Add S_i: Run DP to align S'_1 & $S_i \mapsto S''_i$ and S'_i
  - Adjust $S'_2, \ldots, S'_{i-1}$ by adding spaces where spaces were added in S'_1
- $S_1, S_2, \ldots, S_i \Rightarrow_{aligned} S'_1, S'_2, \ldots, S'_i$.
  - Length(S'_1) in step i $\leq$ i n.
  - DP(S'_1, S_i) takes $O(i\, n^2)$ time
- Total time Complexity

$$= O(k^2 n^2) + \sum_{i=1}^{k} O(i\, n^2) = O(k^2 n^2)$$

Competitiveness

- $\mathcal{M}$ = Alignment induced by the algorithm
- $d(i,j)$ = Distance $\mathcal{M}$ induces on pair S_i, S_j
- $\mathcal{M}^*$ = Optimal alignment
- $2\,SP(\mathcal{M}) = \sum_{i=1}^{k} \sum_{j=1,\, j \neq i}^{k} d(i,j)$

  $\leq \sum_{i=1}^{k} \sum_{j=1,\, j \neq i}^{k} d(i,1) + d(1,j)$

  (Triangle Inequality)

  $= \sum_{i=1}^{k} \sum_{j=1,\, j \neq i}^{k} d(1,i) + \sum_{i=1}^{k} \sum_{j=1,\, j \neq i}^{k} d(1,j)$

  (Symmetry)

  $= \sum_{i=2}^{k} (k-1)\, d(1,i) + \sum_{j=2}^{k} (k-1)\, d(1,j)$

  $= 2(k-1) \sum_{i=2}^{k} d(1,i)$

Competitiveness

- $2SP(\mathcal{M}') \geqq \sum_{i=1}^{k} \sum_{j=1,\, j \neq i}^{k} D(S_i, S_j)$

  $= \sum_{j=2}^{k} D(S_1, S_j) + \sum_{j=1,\, j \neq 2}^{k} D(S_2, S_j) + \cdots + \sum_{j=1}^{k-1} D(S_k, S_j)$

  $\geqq k \sum_{i=2}^{k} d(1,i)$
- $SP(\mathcal{M}') \geqq (k/2) \sum_{i=2}^{k} d(1,i)$

  $\geq (k/2)\, [SP(\mathcal{M})/(k-1)]$
- $SP(\mathcal{M}) \leqq 2\,(1 - 1/k)\, SP(\mathcal{M}^*)$

Local Alignment Problem

Local Alignment Problem

- Finding substrings of high similarity:
- Given two strings, S_1 and S_2: They may have regions that are locally highly similar.

LAP: Local Alignment Problem

- Given: Two strings S_1 and S_2
- Find: Substrings $\alpha \sqsubseteq S_1$ and $\beta \sqsubseteq S_2$ whose similarity (in terms of an object function—e.g., optimal global alignment value) is maximum over all pairs of substrings from S_1 and S_2

$$v^* = \max_{\alpha \sqsubseteq S1,\ \beta \sqsubseteq S2} \text{distance}(\alpha, \beta)$$

Example

- $d(x,x) = 2, d(x,y) = -2$
- $d(x,-) = d(-,x) = -1.$
- $\alpha = a\,x\,a\,b\,c\,s \sqsubseteq S_1$
- $\beta = a\,x\,b\,a\,c\,s \sqsubseteq S_2$

```
S_1 = p q r a x a b c s t v q
            |--- α----|
S_2 =   x y a x b a c s l l
            |--- β----|
```

Local Alignment:

```
a  x  a  b  -  c  s
|  |     |     |  |
a  x  -  b  a  c  s
2  2 -1  2 -1  2  2
```

distance(α, β) = 8

Naïve Complexity

- Note: (1) Let $|S_1| = n$ and $|S_2| = m$.
  - Total number of substrings of $S_1 = C_{n+1,2} = O(n^2)$
  - Total number of substrings of $S_2 = C_{m+1,2} = O(m^2)$
  - Naïvely, $O(n^2m^2)$candidate substrings need to be globally aligned by a DP algorithm of complexity $O(|\alpha|\,|\beta|)$
- Complexity of the resulting algorithm

$$= O(n^3\, m^3)$$

- (2) An improved algorithm (SWAT, Smith-Waterman) reduces the time complexity to $O(nm)$

LSAP: Local Suffix Alignment Problem

- A restricted version of the LAP.
- Given: Two strings S_1 and S_2 and two indices $i \leq |S_1|$ and $j \leq |S_2|$
  - $A_i = S_1[1..i]$ prefix of S_1
  - $B_j = S_2[1..j]$ prefix of S_2
- Find: A suffix (possibly empty, λ) of A_i ($\alpha = S_1[k..i]$) and a suffix of B_j (possibly empty, λ) of B_j ($\beta = S_2[l..j]$) that maximizes a linear objective function $V(\alpha, \beta)$ over all pairs of suffixes of A_i and B_j. □

Objective Function

- $v(i, j) = \max_{\alpha = \text{suf } S1[1..i],\ \beta = \text{suf } S2[1..j]} V(\alpha, \beta)$
  = Value of the optimal local suffix alignment for the given index pair i, j.
- $v^* = \max_{i \leq n,\ j \leq m} v(i, j)$
  = Value of the optimal local alignment.
- $n = |S_1|$, $m = |S_2|$

Optimal Local Alignment: Rec. Eqns.

- $v^* = \max_{i \leqq n,\, j \leqq m} V(i,j)$
- $\alpha = \text{suf}\, S_1[1..i]$, $\beta = \text{suf}\, S_2[1..j]$
- $v^* = v(i', j') = V(\alpha, \beta)$
- Consider an optimal suffix alignment with $\alpha = \text{suf}\, S_1[1..i]$ and $\beta = \text{suf}\, S_2[1..j]$
- Case 1: $\alpha = \beta = \lambda$ (= empty string)
  - Base: $V(\alpha, \beta) = 0$

Optimal Local Alignment: Rec. Eqns.

- Case 2: $\alpha \neq \lambda$, $\alpha = \alpha' \circ S_1[i]$ and $S_1[i]$ matches "–"
  - Ind(A): $V(\alpha,\beta) = V(\alpha', \beta) + d(S_1[i], -)$
- ...or $S_1[i]$ matches $S_2[j]$ ($\beta = \beta' \circ S_2[j]$)
  - Ind(C): $V(\alpha,\beta) = V(\alpha', \beta') + d(S_1[i], S_2[j])$

Optimal Local Alignment: Rec. Eqns.

- Case 3: $\beta \neq \lambda$, $\beta = \beta' \circ S_2[j]$ and $S_2[j]$ matches "–"
  - Ind(B): $V(\alpha,\beta) = V(\alpha, \beta') + d(-, S_2[j])$
- ...or $S_1[i]$ matches $S_2[j]$ ($\alpha = \alpha' \circ S_1[i]$)
  - Ind(C): $V(\alpha,\beta) = V(\alpha', \beta') + d(S_1[i], S_2[j])$

Recurrence Equation

- $V(i,j) = \max_{\alpha = \text{suf}\, S1[1..i],\ \beta = \text{suf}\, S2[1..j]} V(\alpha, \beta)$
- Base: $v(i,j)|_{i=0 \vee j=0} = 0$

  $(v(0,0) = v(i,0) = v(0,j) = 0)$
- Induction: $v(i,j)|_{i=0 \wedge j=0} = \max[0,$

  $v(i-1,j) + d(S_1[i],-),$

  $v(i,j-1) + d(-, S_2[j]),$

  $v(i-1,j-1), d(S_1[i], S_2[j])\ ]$

Dynamic Programming Table

- (with Traceback)
- Compute all $v(i,j)$ entries: Complexity = $O(nm)$
- Find $v^* = v(i^*, j^*)$ by finding the largest value in any cell: Complexity = $O(nm)$
- Trace the pointer back from from $v(i^*, j^*)$ until a cell is reached with value $v(i',j') = 0$:

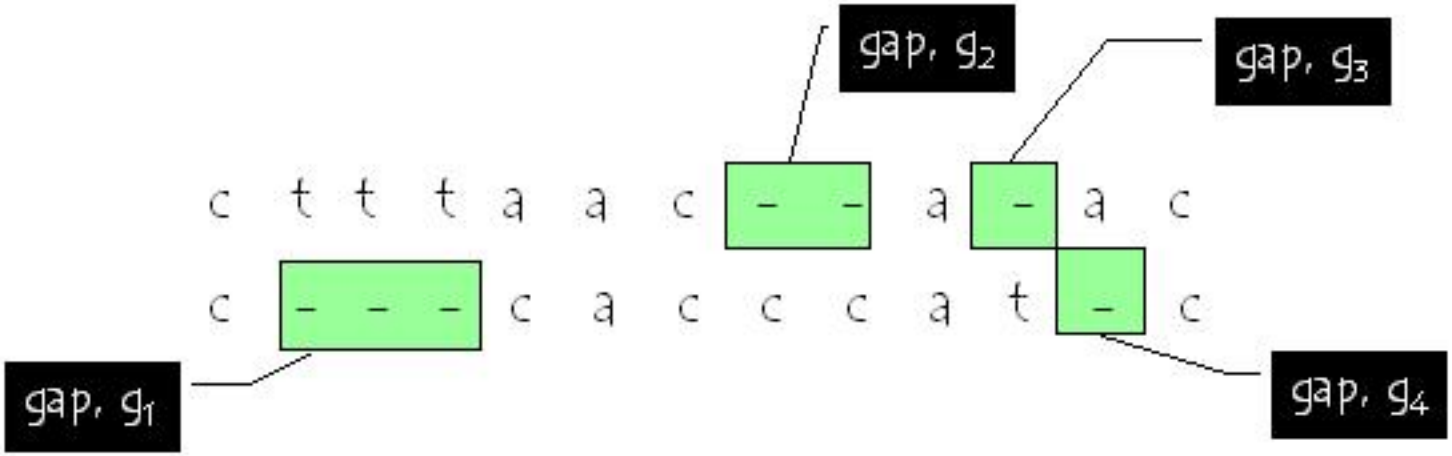
  Complexity = $O(n+m)$
- Results: $\alpha = S_1[i'..i^*] \sqsubseteq S_1$ and $\beta = S_2[j'..j^*] \sqsubseteq S_2$
- Total Complexity = $O(nm) = O(|S_1|, |S_2|)$

Example

	.	x	y	a	x	b	a	c	s	l	l
.	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0
r	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	↖2	←1	0	↖2	←1	0	0	0
x	0	↖2	←1	↑1	↖4	←3	←2	←1	0	0	0
a	0	↑1	0	↖3	↑3	↑2	↖5	←4	←3	←2	←1
b	0	0	0	↑2	↑2	↖5	←4	←3	←2	←1	←0
c	0	0	0	↑1	↑1	↑4	↑3	↖6	←5	←4	←3
s	0	0	0	0	0	↑3	↑2	↑5	↖8	←7	←7
t	0	0	0	0	0	↑2	↑1	↑4	↑7	←6	←6
v	0	0	0	0	0	↑1	0	↑3	↑6	←5	←5
q	0	0	0	0	0	0	0	↑2	↑5	←4	←4

Dealing with Gaps

- A gap is any "maximal consecutive run of spaces" in a single string of a given alignment.



Gaps

- Initial Gap
  - A gap may be bordered on the right by the first character of a string.
- Final Gap
  - A gap may be bordered on the left by the last character of a string.
- Internal Gap
  - A gap may be bordered on both left and right
- **Simple Gap Penalty Model $\mapsto$ Constant Wt, W_g**
  - Each gap contributes a constant penalty = W_g
  - $d(x,x) = 2$, $d(x,y) = -2$, $d(x,-) = d(-,y) = 0$
  - # gaps = k. Then
  - Value of an alignment = $\sum_{i=1}^{l} d(S'_1[i], S'_2[i]) - k\,W_g$

Biological Motivations for Gap Models

- Unequal Crossing-over in Meiosis
- DNA slippage during replication
- Insertion of transposable elements ("Jumping Genes")
- Insertion by retroviruses
- Translocation between chromosomes

◇ Examples of Alignment with gaps:

- *cDNA matching problem*
- *Processed Pseudo-gene Problem*

Gap Weights

- Constant:
  - Each gap has a penalty of W_g
  - Each space is free: $d(x,-) = d(-,x) = 0$.
- Affine:
  - Gap initiation weight = W_g
  - Gap Extension weight = W_s
  - Each gap of length q has a penalty of $W_g + q\ W_s$
- Convex:
  - Each gap of length q has a penalty of $W_g + \ln q\ W_s$
- Arbitrary:
  - Each gap of length q has a penalty of $W_g + \omega(q)\ W_s$, where $\omega(q)$ = arbitrary function

General Model

- **Arbitrary**:
  - Each gap of length q has a penalty of $W_g + \omega(q)\, W_s$, where $\omega(q)$ = arbitrary function
  - $\omega(q) = 0 \mapsto$ constant
  - $\omega(q) = q \mapsto$ linear/affine
  - $\omega(q) = \ln q \mapsto$ convex
- Total Cost under constant model

$$\sum_{i=1}^{l} d(S'_1[i], S'_2[i]) - (\#\text{gaps})\, W_g$$

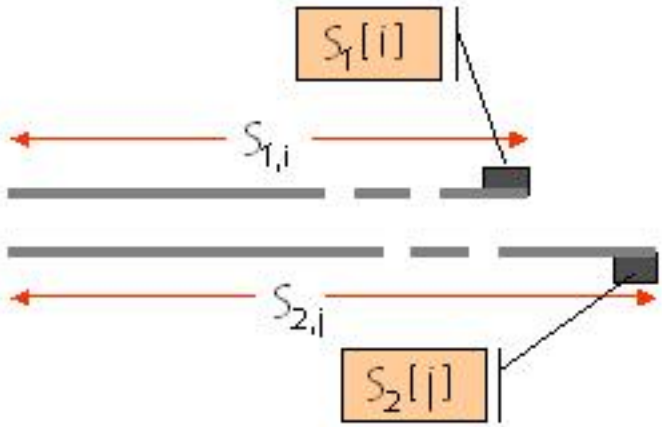
- Total Cost under affine model

$$\sum_{i=1}^{l} d(S'_1[i], S'_2[i]) - (\#\text{gaps})\, W_g - (\#\text{spaces})\, W_s$$

Local Alignment
Arbitrary Gap Weight Model

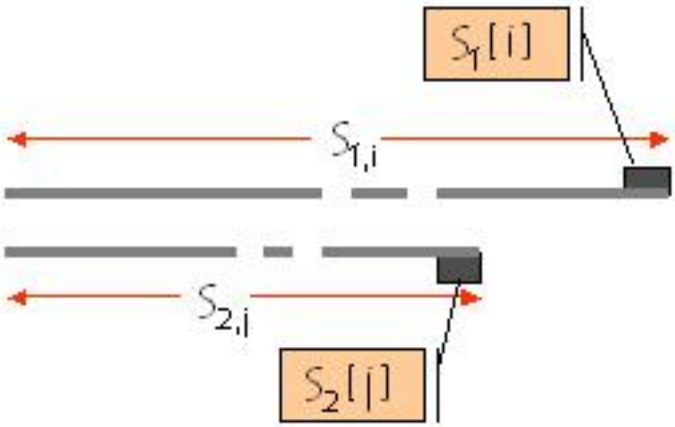
- Dynamic Programming
  (Needleman & Wunsch)
- Given two strings S_1 and S_2 start by aligning the prefixes
  - $S_{1,i} = S_1[1..i]$ and
  - $S_{2,j} = S_2[1..j]$
- There are three different cases to consider…

Case 1



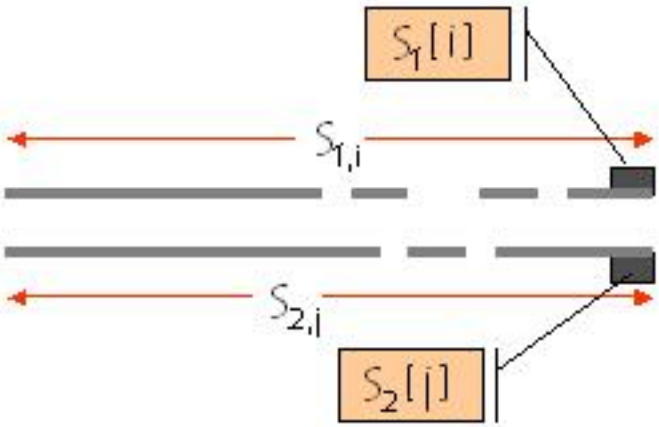
- $S_1[i]$ is aligned to a character <u>strictly to the left</u> of a character $S_2[j]$

Case 2



- $S_1[i]$ is aligned to a character <u>strictly to the right</u> of a character $S_2[j]$

Case 3



- $S_1[i]$ and $S_2[j]$ are aligned opposite each other:
  - Subcase A

    $S_1[i] = S_2[j]$
  - Subcase B

    $S_1[i] \neq S_2[j]$

Auxiliary Vaiables

- $X_L(i,j) =$

  $\max_{\text{alignments for case 1}} \text{distance}(S_1[1..i], S_2[1..j])$
- $X_R(i,j) =$

  $\max_{\text{alignments for case 2}} \text{distance}(S_1[1..i], S_2[1..j])$
- $X_S(i,j) =$

  $\max_{\text{alignments for case 3}} \text{distance}(S_1[1..i], S_2[1..j])$
- $V(i,j) = \max(X_L(i,j), X_R(i,j), X_S(i,j))$

Recurrence: Base

- Notation: $\perp \triangleq$ "undefined"
- $X_S(0,0) = 0$, $X_S(i,0) = \perp$, $X_S(0,j) = \perp$
- $X_L(0,0) = \perp$, $X_L(i,0) = -\omega(i)$, $X_L(0,j) = \perp$
- $X_R(0,0) = \perp$, $X_R(i,0) = \perp$, $X_R(0,j) = -\omega(j)$
- $V(0,0) = 0$, $V(i,0) = -\omega(i)$, $V(0,j) = -\omega(j)$

Recurrence: Induction

$i > 0$ and $j > 0$:

- $X_S(i,j) = V(i-1,j-1) + d(S_1[i], S_2[j])$
- $X_L(i,j) = \max_{0 \leq k \leq j-1} (V(i,k) - \omega(j-k))$
- $X_R(i,j) = \max_{0 \leq l \leq i-1} (V(l,j) - \omega(i-l))$
- $V(i,j) = \max(X_L(i,j), X_R(i,j), X_S(i,j))$
- Each $V(i,j)$ can be computed in time $O(i+j)$

Total Time Complexity

- Let $|S_1| = n$ and $|S_2| = m$.
- The recurrence can be evaluated with a Dynamic Programming Table of space complexity = $O(nm)$ and in time complexity = $O(n^2m+m^2n)$

Affine Gap Model-Recurrence

- SWAT : Smith-Waterman
- Modifying the recurrence equations for the affine case:
  - $X_S(0,0) = 0$, $X_S(i,0) = \perp$, $X_S(0,j) = \perp$
  - $X_L(0,0) = \perp$, $X_L(i,0) = -W_g - i\,W_s$, $X_L(0,j) = \perp$
  - $X_R(0,0) = \perp$, $X_R(i,0) = \perp$, $X_R(0,j) = -W_g - j\,W_s$
  - $V(0,0) = 0$, $V(i,0) = -W_g - i\,W_s$, $V(0,j) = -W_g - j\,W_s$

Recurrence: Induction

$i > 0$ and $j > 0$:

- $X_S(i,j) = V(i-1,j-1) + d(S_1[i], S_2[j])$
- $X_L(i,j) = \max(X_L(i, j-1) - W_s, \perp, X_S(i,j-1) - W_g - W_s, V(i,j-1) - W_g - W_s)$
  $= \max[X_L(i, j-1), V(i,j-1) - W_g] - W_s$
- $X_R(i,j) = \max(\perp, X_R(i-1, j) - W_s, X_S(i-1,j) - W_g - W_s, V(i-1,j) - W_g - W_s)$
  $= \max[X_R(i-1, j), V(i-1,j) - W_g] - W_s$
- $V(i,j) = \max(X_L(i,j), X_R(i,j), X_S(i,j))$
- Each $V(i,j)$ can be computed in $O(1)$ time. The optimal alignment with affine gap weights can be computed with a DP table of space and time complexity = $O(nm)$.

Heuristic Alignment

Heuristic Alignment Motivation

- O(m n) time complexity:
  - too slow for large databases with high query traffic
  - heuristic methods do fast approximation to dynamic programming
- FASTA [Pearson & Lipman, 1988]
- BLAST [Altschul et al., 1990]
  - BLAST heuristically finds high scoring segment pairs (HSPs):
    - identical length segments from 2 sequences with statistically significant match scores
    - i.e. ungapped local alignments
    - key tradeoff: sensitivity vs. speed

BLAST Overview

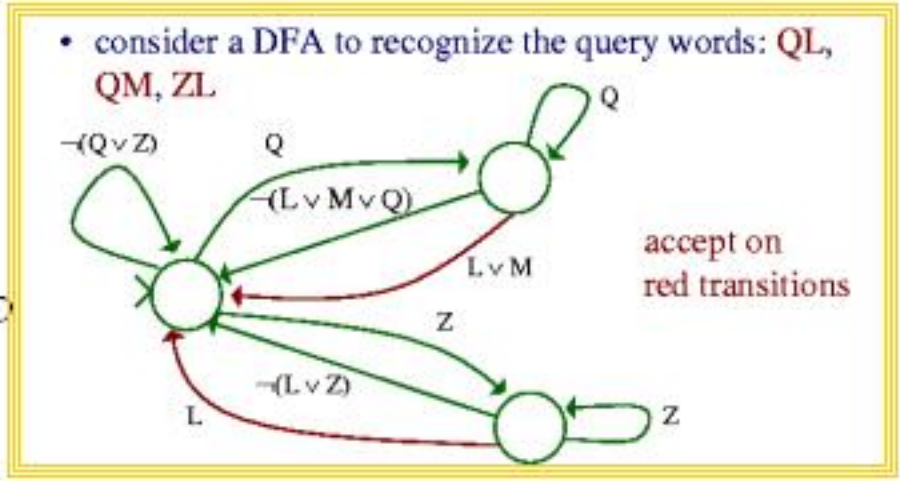
- Basic Local Alignment Search Tool
- BLAST heuristically finds high scoring segment pairs (HSPs):
  - identical length segments from 2 sequences with statistically significant match scores
  - i.e. ungapped local alignments
  - key tradeoff: sensitivity vs. speed
- Sensitivity is just the ratio of
  - # significant matches in DB
  - # significant matches detected

BLAST Overview

- given: query sequence q, word length w, word score threshold T, segment score threshold S
  - compile a list of "words" that score at least T when compared to words from q
  - scan database for matches to words in list
  - extend all matches to seek high-scoring segment pairs
- return: segment pairs scoring at least S

Scanning the Database

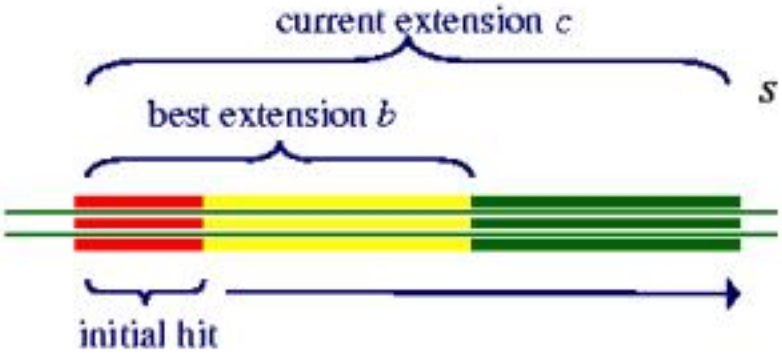
- search database for all occurrences of query words approach:
  - build a DFA that recognizes all query words
  - run DB sequences through DFA
  - remember hits
- use Mealy paradigm (accept on transitions) to save space and time
  - consider a DFA to recognize the query words: QL, QM, ZL



Extending Hits

- extend hits in both directions (without allowing gaps)
- terminate extension in one direction when score falls certain distance below best score for shorter extensions
- return segment pairs scoring at least S

$Score(c) \geq score(b) - \varepsilon?$



Is BLAST the right tool?

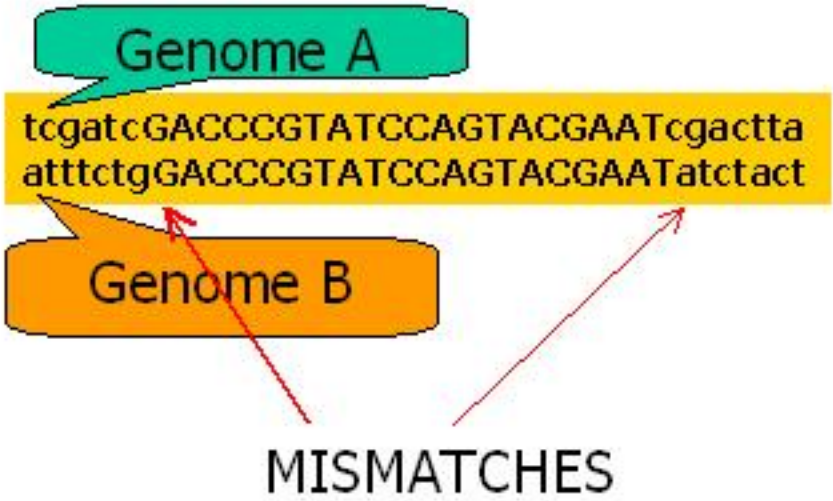
- may fail to find all HSPs
  - may miss seeds if threshold, T is too stringent
  - extension is greedy
- empirically, 10 to 50 times faster than Smith-Waterman (Swat)
- large impact:
  - NCBI's BLAST server handles more than 50,000 queries a day
  - The ultimate low-lying fruit: most used bioinformatics program; most cited paper in science!!
- the two-hit method
  - gapped BLAST
  - PSI-BLAST
    - all are aimed at increasing sensitivity while limiting run-time
  - Altschul et al., Nucleic Acids Research 1997

Whole Genome Alignment

The MUMmer System

- Delcher et al., Nucleic Acids Research, 1999
- given genomes A and B
  - find all maximal, unique, matching subsequences (MUMs)
  - extract the longest possible set of matches that occur in the same order in both genomes
  - close the gaps
  - output the alignment

Step 1: MUM Decomposition



- maximal unique match (MUM):
  - occurs exactly once in both genomes A and B
  - not contained in any longer MUM
- key insight:
  - a significantly long MUM is certain to be part of the global alignment

MUM with Suffix Tree

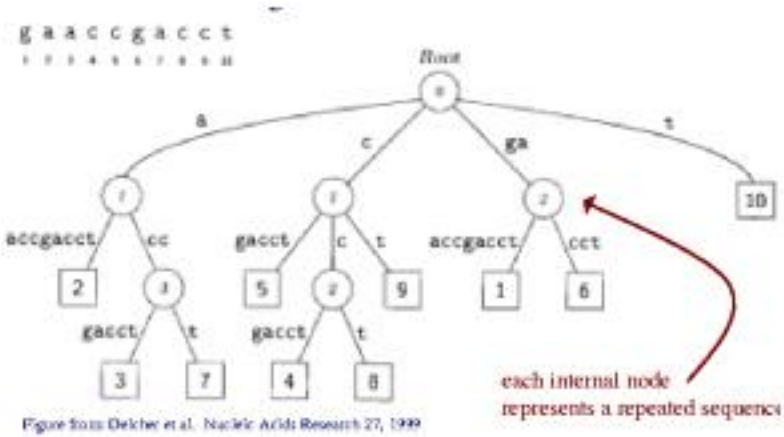
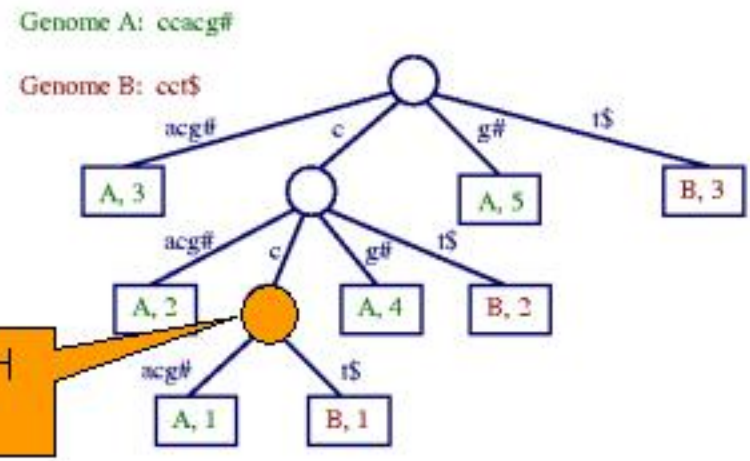


Figure from Delcher et al. Nucleic Acids Research 27, 1999

- add suffixes for both genomes A and B to tree
- label each leaf node with genome it represents

In a preprocessing step, build a suffix tree for genomes A and B



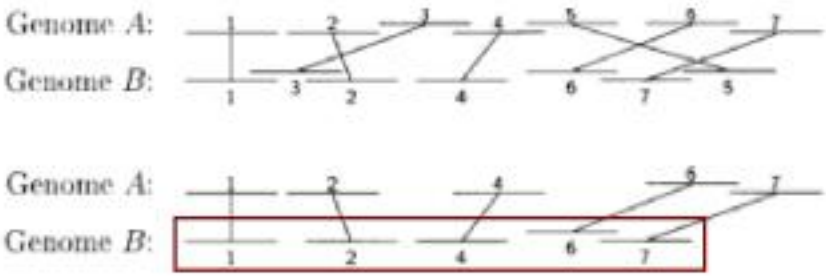
UNIQUE MATCH for MUM

Suffix Trees

- can build in linear time (in lengths of genomes)
- can identify all MUMs in linear time (one scan of tree)
- space complexity is linear (exactly one leaf and at most one internal node for each base)
- main parameter of system: length of shortest MUM that should be identified (20 - 50bp here)

Step 2: Find Longest Subsequence

- sort MUMs according to position in genome A
- solve variation of Longest Increasing Subsequence (LIS) problem to find sequences in ascending order in both genomes
  - requires O(k log k) time where k is number of MUMs



Step 3: Close the Gaps

- polymorphic regions
  - short ones: align them with dynamic programming method
  - long ones: call MUMmer recursively w/ reduced min MUM length
- Handle: SNPs, Repeats, and Indels separately, case-by-case.

Is MUMmer the right tool?

- Problems with low homology regions.
  - Distantly related genomes with very low homologies.

The More the Merrier

Algorithm	Homology Seed	Indexing	Reference
BLAST	exact k-mer	Scanned with DFA*	[31]
WABA	wobble base degenerate k-mer	Array	[32]
LSH-ALL-PAIRS	randomly projected k-mer with <d mismatches	Sorted Array	[33]
BLASTZ	discontinuous exact k-mer	Hash Table	[34][35]
PatternHunter	discontinuous exact k-mer	Hash Table	[36]
BLAT	exact or inexact k-mer	Hash Table	[37]
CHAOS	exact or inexact k-mer	T-Trie	[38]
PASH	discontinuous exact k-mer	Hash Table	[39]
REPuter	maximal exact repeat	Suffix Tree	[40]
[illegible]	exact repeat	Factor Oracle	[41]

Summary

- Many innovative sequence alignment tools available for detailed comparative genomics studies.
  - Recent segmental duplications in mammalian genomes (with identity level >90%) can be detected using BLAST and many other tools.
- They use exact or inexact k-mers as homology seeds for local alignment extension. As homology levels become lower, they encounter a dilemma between sensitivity and computational efficiency
  - homologous segments,
  - segmental duplications, or
  - homology-based phylogentic distances.

Summary

- To improve sensitivity they must rely on exhaustive searches of exact matches with short mers or inexact matches with longer mers,
  - and thus encounter too many false-positives, to be later filtered through an expensive post-processing step.
- Or, if more stringent search criteria (longer mers with more exact matches) are used to improve efficiency,
  - then these algorithms fail to detect low-homology regions, such as ancient duplication events.
- In order to detect less-recent duplications, orthologous genes have been used as "anchors" to map out the duplication blocks. But, for obvious reasons, they are unsuitable for identifying duplications that are not subject to strong selection process, e.g., regions containing only non-coding regions.

Prizm

PRIZM

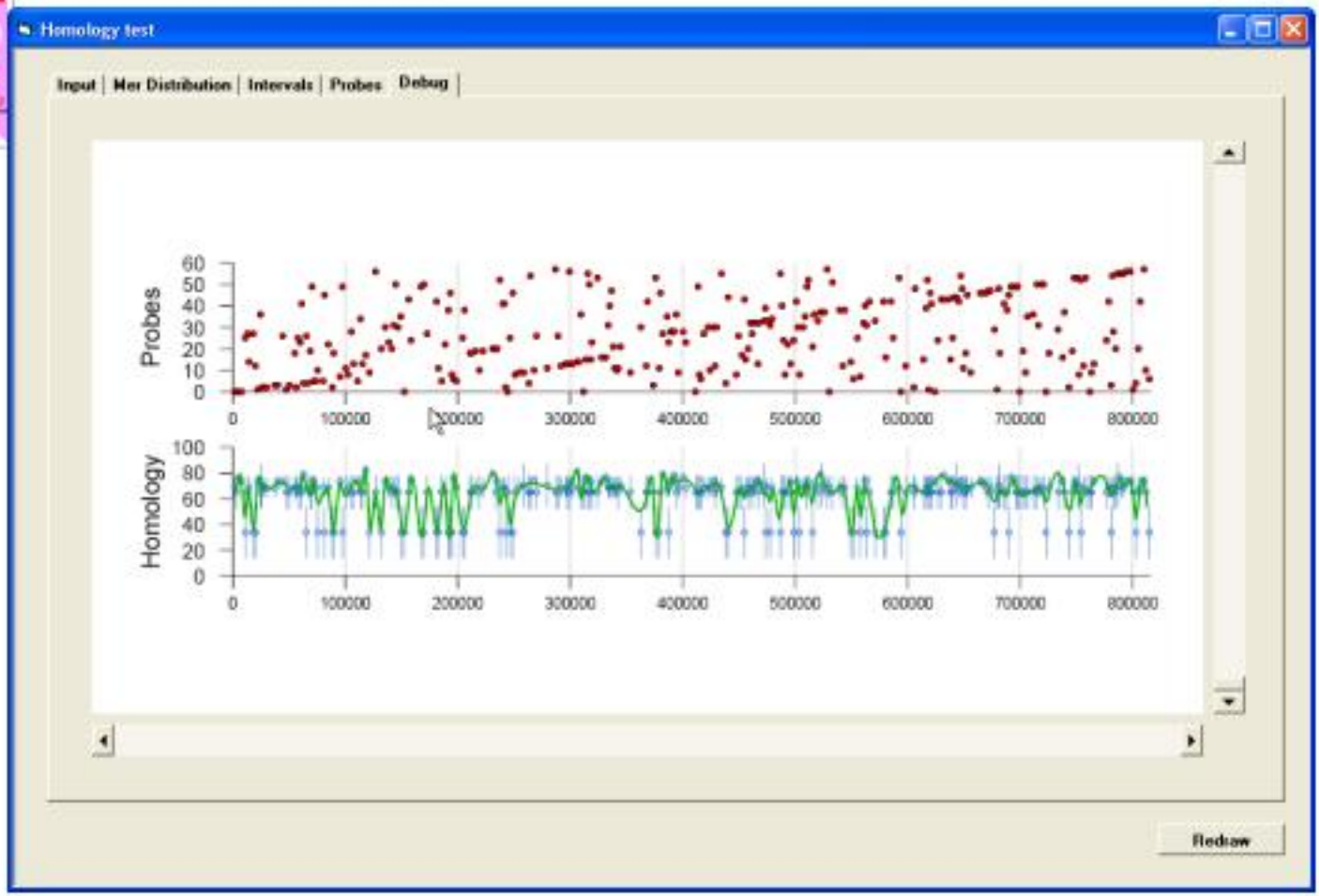
- It uses a Bayesian scheme.
  - It is efficient…Linear time.
  - It computes homologous regions between two genomes even when the homology level drops to a value around 65%.
    - Incorporates background knowledge about genome evolution, by experimenting with several priors (noninformative improper prior, exponential and Gamma priors, and priors based on Juke-Cantor one parameter and Kimura's two parameters models of evolution).
    - The results appear to be unaffected by these choices, while the computational efficiency is only mildly affected by what prior is employed.

A Simple Observation

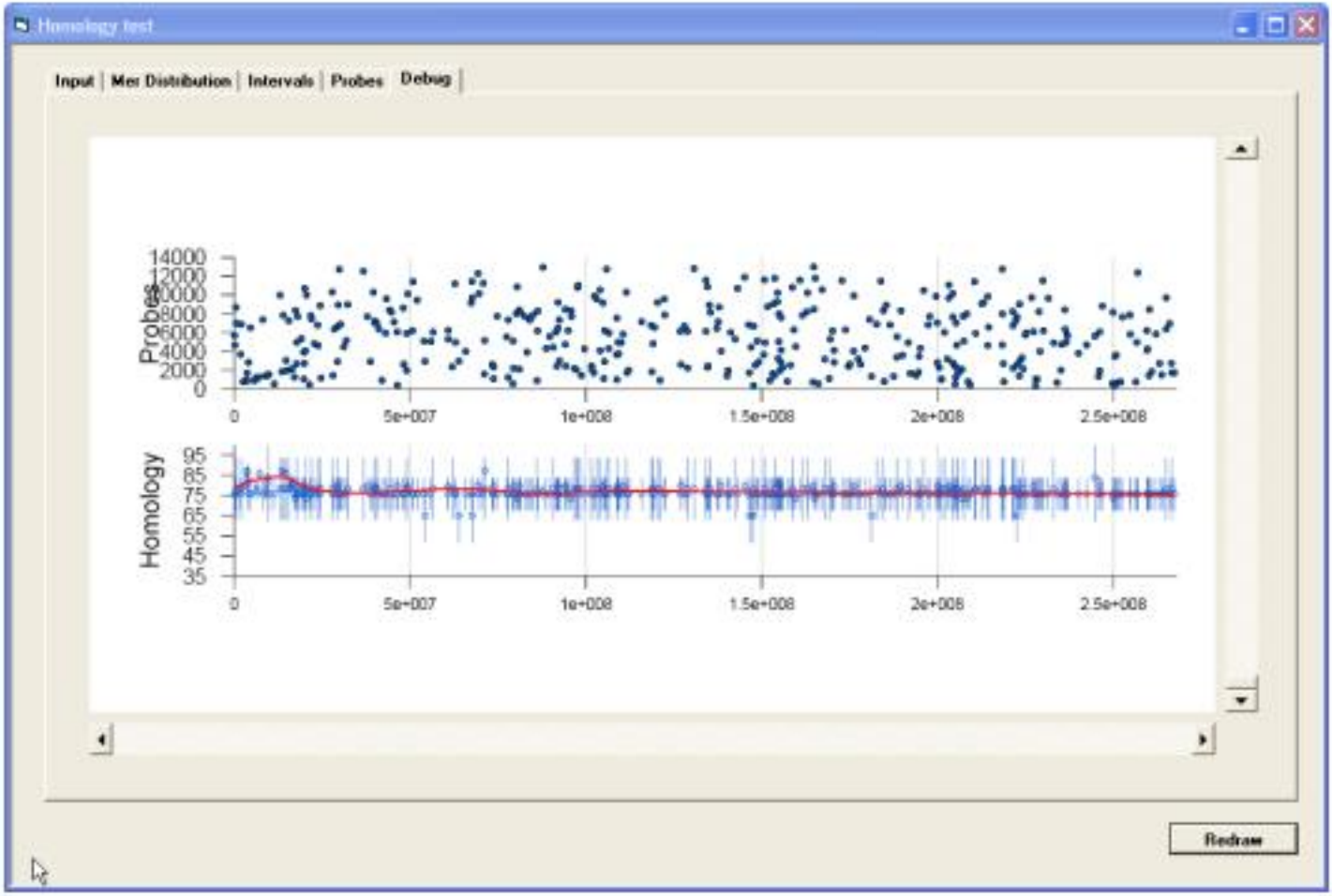
- Homology is hard to compute but easy to verify. Quadratic vs. Linear time.
  - Can a probabilistic approach replace nondeterminism? If so, we can expect a probabilistic linear time algorithm.
  - Unlikely! But if we can use priors based on the underlying distributions, there is hope.
  - Many computational biology problems share this feature!

Homology Curve

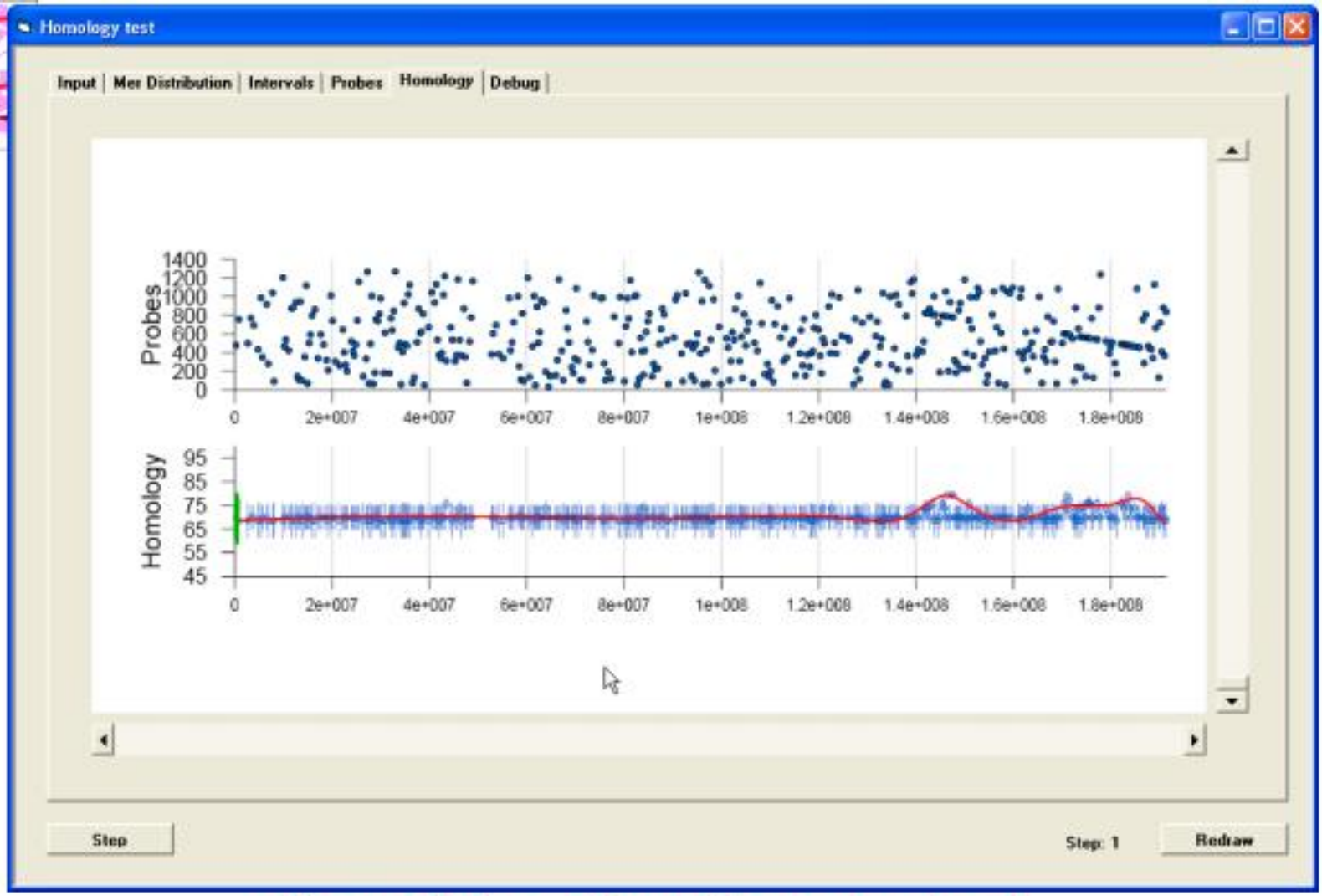
- The algorithm quickly produces a function called "homology curve."
  - The value of this function at a location on the first genome is simply the highest local homology level of a short region straddling that location in the second genome.
  - The structure of this curve, [the distribution of the homologies across the whole genome, its correlation with the composition of the underlying regions (e.g., coding vs. noncoding, CG-content, stability and flexibility), etc.] can tell us how the two genomes are related in terms of
    - evolutionary distances,
    - patterns of conservation, and
    - mechanisms of evolution and selection that acted since their divergence.



The genomic sequences under comparison: A) **M.** *genitalium (Y-axis)* **and M.** *pneumoniae (X-axis), computed using 300 probes from six iterations, taking about 5 seconds using a non-optimized algorithm*



The mouse chromosome 10 and rat chromosome 1 share a syntenic region about 20Mb at the beginning of the two chromosomes (arrow).



Alignment between Mouse Chromosome 8 and human chromosome 4.

(5,105,3), and takes about 45 seconds.

Homology Curve

- An m-mer is a word of length **m**, selected from either genome.
- Consider a location in the first genome, $G_1[\alpha]$ and a short window, starting at α.

$$W_{1,\alpha} = G_1[\alpha, \alpha+m-1]$$

- Compare this window with a word of equal length from the second genome starting at $G_2[\beta]$:

$$W_{2,\beta} = G_2[\beta, \beta+m-1].$$

- Define the homology level for the locations $G_1[\alpha]$ and $G_2[\beta]$ and a window of size **m** as

$$h^{(2)}(G_1[\alpha], G_2[\beta], m) = (1/m) \sum_{\gamma=1}^{m-1} \mathbb{I}_{G1[\alpha+\gamma] = G2[\alpha+\gamma]}.$$

Homology Curve

- Let us define, $h(\alpha)$ to denote the highest homology level for genome G_1 at position α and computed with respect to G_2:

$$h(\alpha) = \max_{1 \leq \beta < |G2| - m} h^{(2)}(G_1[\alpha], G_2[\beta], m)$$

- The "homology curve" for the first genome, G_1 with the respect to the second genome G_2 is then defined as:

$$h : [1..G_1 - m - 1] \rightarrow [0, 1]$$
$$: \alpha \mapsto h(\alpha)$$

PRIZM

- Replacing non-determinism with a probabilistic guessing scheme.
  - The probability distributions are based on biologically meaningful priors.
  - Using these priors it guesses a local homology curve, and designs and performs an in silico experiment.
  - It uses the results to verify its guess (in linear time) and refines the local homology curve and the probability distributions for the next iteration.

Probabilistic guesses

- Use a Bayesian scheme and a boosting approach to modify the probability distributions of the "guessing experiments" from one iteration to next.
- At each iteration, a sequence of words with a specific distribution is selected from one genome, and is optimally partitioned into groups for "in silico experiments" involving
  - exact-match search,
  - inexact-match search with one error,
  - inexact match search with two errors, etc.

Probabilistic guesses

- These searches can be efficiently conducted over the second genome,
  - Assuming that the other genome has been preprocessed and stored in an efficient data structure (e.g., suffix arrays or hash table).
  - From the results of the experiments, a Bayesian estimator can compute the local homology levels for the genome, and use it to verify and sharpen the probabilistic distributions for the next iteration.
- The algorithm converges to the true local homology levels after a few iterations.

In Silico Experiments

- Let b, $B = |G2|/b$, w, W, m, N_0, N_1, . . ., N_k ($k \leq m$, and in our applications usually $k = 2$) be some pre-specified parameters.
  - Choose $k+1$ random subsets, S_0, S_1, . . ., S_k, of words each of length m, randomly (i.i.d. uniform) from $G_1[\alpha, \alpha+w-1]$, such that

$$|S_0| = N_0,\ |S_1| = N_1,\ \ldots,\ \text{and}\ |S_k| = N_k.$$

  - Consider a block in the second genome of length b: $B_\beta = G_2\ [\beta, \beta+b-1]$. Let X_0 (X_1, . . ., X_k, respectively) be defined as the number of m-mers from set S_0 (S_1, . . ., S_k, respectively) that match exactly (with one, two and so on up to k mismatches, respectively) to an m-mer in $G_2[\beta, \beta+b-1]$.

Experiment Design

- By examining the sensitivity ($\partial(X_i/N_i)/\partial h = a'_i[h]$) we can divide the interval for h into three intervals: $[1/4, \theta_1] \approx [1/4, (m-2)/m]$, $(\theta_1, \theta_2] \approx [(m-2)/m, (m-1)/m]$ and $(\theta_1, 1] \approx ((m-1)/m, 1]$, such that the choices of (N_0, N_1, N_2) are based on the following mixed strategies:

$$N_0 = (K/b) \int_{\theta 2}^{1} p_{i,l}(h)\, dh$$

$$N_1 = (K/3bm) \int_{\theta 1}^{\theta 2} p_{i,l}(h)\, dh$$

$$N_2 = (2K/9b(m^2 - m)) \int_{1/4}^{\theta 1} p_{i,l}(h)\, dh$$

....

In Silico Experiments

- Thus X_i's for **i** in **[0..k]** are binomially distributed random variables whose parameters depend on the homology level **h**.
- We can estimate the local homology by the following robust estimators:

$$\langle h \mid X_0, X_1, \ldots X_k \rangle$$
$$= \int_0^1 h\, p(h \mid X_0, \ldots, X_k)\, dh$$
$$= \int_0^1 h\, p(h)\, p(X_0, \ldots, X_k \mid h)\, dh / \int_0^1 p(h)\, p(X_0, \ldots, X_k \mid h)\, dh$$

  - Similarly, compute the mean, standard deviation and confidence of the homology function over B_β.
  - Let $\beta^* = \arg\max_\beta \text{mean}(B_\beta)$. Then the homology function is estimated at α by $\text{mean}(B_{\beta^*})$.

Conditional Probabilities

$$r_i = b\, p^m \sum_{j=1}^{i} C[m,j]\, 3^j$$

$$s_i = \sum_{j=1}^{i} C[m,j]\, h^{m-j} (1-h)^j$$

$$b_i = (1 - s_i)(1 - r_i)$$

$$a_i = 1 - b_i = s_i + r_i - s_i\, r_i.$$

$$p(X_1, X_2, \ldots, X_K | h) \propto \prod_i a_i^{X_i}\, b_i^{N_i - X_i}$$

Initial Priors

- Using Jukes-Cantor: the random variable r represents the rate of nucleotide substitution per site per year.
  - In this model, it is assumed that nucleotide substitution occurs at any nucleotide site with equal frequency and at each site a nucleotide changes to one of the three remaining nucleotides with a probability $\boldsymbol{\alpha}$ per year: $r = 3\,\boldsymbol{\alpha}$.
  - The substitution rate is often higher at functionally less important sites than at functionally more important sites.
  - Case 1: $r \sim \text{Exponential}(\lambda)$: $f_{exp}(r) = \lambda e^{-\lambda r}$. In that case
    $$p(h) \sim (4h - 1)^{3\lambda/8T - 1}$$
  - Case 2: $r \sim \text{Gamma}(\lambda, \nu)$: $f_{\Gamma}(r) = \lambda^{\nu} e^{-\lambda r} r^{\nu-1}/\Gamma(\nu)$. In that case
    $$\ldots^{3\lambda/8T-1} \ln[3/(4h - 1)/T]^{\nu-1}$$

Initial Priors

- A more complex structures arise as we consider multi-parameter models: e.g., Kimura's Two-Parameter Method. In this model, the rate of transitional substitution per site per year (α) is assumed to be different from that of transversional substitution (2β).

$$h = (1 - P)(1 - Q)$$

$$P = (1/4)\,(1 - 2e^{-4(\alpha+\beta)T} + e^{-8\beta T})$$

$$Q = (1/2)\,(1 - e^{8\beta T})$$

Initial Priors

- Assume that $\alpha \sim$ **Exponential**(λ_α) and $\beta \sim$ **Exponential**(λ_β) and they are independent.

$p(h) = (\lambda_\alpha \lambda_\beta / 8T^2)$

$$\int_0^1 (1-P)^{2+(\lambda\alpha+\lambda\beta)/8T}(2h+P-1)^{(\lambda\alpha-\lambda\beta)/8T}$$
$$(h-2P+P^2)^{-\lambda\alpha/4T}$$
$$/(2h+P-1)(h-2P+P^2)\,dP.$$

Refining Priors

- In iteration **i**, let us consider an interval **I** with **k** homology estimates:

$$\langle \mu_1, \sigma_1 \rangle, \langle \mu_2, \sigma_2 \rangle, \ldots, \langle \mu_k, \sigma_k \rangle$$

- Assume that the homology values $h_1, h_2, \ldots, h_k$ is sampled from a distribution

$$h \sim \mathbb{N}(\mu, \sigma^2).$$

- Furthermore, we assume the following:

$$\mu \sim \mathbb{N}(\xi, \tau^2)$$

$$r = (\sigma^2 + \tau^2)^{-1} \sim \Gamma(\alpha, \beta).$$

New Prior

- Prior = Kummer's hypergeometric function of order 1

$$f(h|\xi, \tau, \alpha, \beta)$$
$$\sim \int_0^{1/\tau 2} r^{\alpha-2}\, 1/\sqrt{(1 - r\,\tau^2)}\; e^{-Br}\, dr$$
$$\sim {}_1F_1(\alpha - 1, \alpha - 1/2, -B/\tau^2)$$
$$\approx ((h-\xi)^2 + 2\,\beta)/2\,\tau^2)^{-\alpha+1}$$

- Estimates

$$\xi = \langle \mu_i \rangle$$
$$\tau^2 = \langle \mu^2 \rangle - \langle \mu_i \rangle^2$$
$$\alpha/\beta = \langle (\sigma^2_i + \tau^2)^{-1} \rangle$$
$$\alpha/\beta^2 = \langle (\sigma^2_i + \tau^2)^{-2} \rangle - \langle (\sigma^2_i + \tau^2)^{-1} \rangle^2$$

Optimizing the parameters

- The parameter choices are as follows:
  - Let the number of blocks (**b**) and the number of windows (**w**) be chosen a priori based on the needed resolution for homology.
  - We may choose these parameters so that $b = O(\sqrt{(G_2)})$ and $w = O(\sqrt{(G_1)})$. We assign $K = O(1)$ amount of work to a region defined by a combination of any single block with any single window.
  - Thus the amount of work is roughly $K(G_1G_2)/(wb) = O(G1+G2)$ per iteration.
  - The mer size parameter '**m**' is chosen so that the probability of a "hit" in a block containing a homologous sequence much higher than in a random block:

$$(b/4^m) \ll \mathbb{E}(h_0,G)^m.$$

To be continued…

…