CSCE 471/871 Lecture 2: Pairwise Alignments

Stephen Scott

sscott@cse.unl.edu

Outline

- What is a sequence alignment?
- Why should we care?
- How do we do it?
  - Scoring matrices
  - Algorithms for finding optimal alignments
  - Statistically validating alignments

What is a Sequence Alignment?

- Given two nucleotide or amino acid sequences, determine if they are related (descended from a common ancestor)
- Technically, we can align any two sequences, but not always in a meaningful way
- In this lecture, we'll focus on AA sequences, but same alignment principles hold for DNA sequences

What is a Sequence Alignment? (cont'd)

```
HIGHLY RELATED:
HBA_HUMAN    GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
             G+ +VK+HGKKV  A+++++AH+D++ +++++LS+LH  KL
HBB_HUMAN    GNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKL

RELATED:
HBA_HUMAN    GSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSDLHAHKL
             ++ ++++H+ KV   + +A  ++          +L+ L+++H+ K
LGB2_LUPLU   NNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG

SPURIOUS ALIGNMENT:
HBA_HUMAN    GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSD----LHAHKL
             GS+ + G +   +D L  ++ H+ D+  A +AL D     ++AH+
F11G11.2     GSGYLVGDSLTFVDLL--VAQHTADLLAANAALLDEFPQFKAHQE
```

How to filter out the last one & pick up the second?

Why Should We Care?

- Fragment assembly in DNA sequencing
  - Experimental determination of nucleotide sequences is only reliable up to about 500-800 base pairs (bp) at a time
  - But a genome can be millions of bp long!
  - If fragments overlap, they can be assembled:

```
...AAGTACAATCA
        CAATTACTCGGA...
```

  - Need to align to detect overlap

Why Should We Care? (cont'd)

- Finding homologous proteins and genes
  - I.e., evolutionarily related (common ancestor)
  - Structure and function are often similar, but this is reliable only if they are evolutionarily related
  - Thus want to avoid the spurious alignment of Slide 4

How do we do it?

- Choose a scoring scheme
- Choose an algorithm to find optimal alignment wrt scoring scheme
- Statistically validate alignment

Scoring Schemes

- Since goal is to find related sequences, want evolution-based scoring scheme
  - Mutations occur often at the genomic level, but their rates of acceptance by natural selection vary depending on the mutation
  - E.g., changing an AA to one with similar properties is more likely to be accepted
- Assume that all changes occur independently of each other and are Markovian

  ⇒ Changes occuring now are independent of those in the past

  ⇒ Makes working with probabilities easier

Scoring Schemes (cont'd)

- If AA a_i is aligned with a_j, then a_j was substituted for a_i

  `...KALM...`

  `...KVLM...`

- Was this due to an accepted mutation or simply by chance?
  - If A or V is likely in general, then there is less evidence that this is a mutation
- Want the score s_{ij} to be higher if mutation more likely
  - Take ratio of mutation prob. to prob. of AA appearing at random
- Generally, if a_j is similar to a_i in property, then accepted mutation more likely and s_{ij} higher

Scoring Schemes (cont'd)

- Only consider immediate mutations $a_i \to a_j$, not $a_i \to a_k \to a_j$
- Mutations are undirected
  $\Rightarrow$ scoring matrix is symmetric

The PAM Transition Matrices

- Dayhoff et al. started with several hundred manual alignments between very closely related proteins ($\geq 85\%$ similar in sequence), and manually-generated evolutionary trees
- Computed the frequency with which each AA is changed into each other AA over a short evolutionary distance (short enough where only 1% AAs change)
- 1 PAM = 1% point accepted mutation
- Becomes our measure of evolutionary "time"

The PAM Transition Matrices (cont'd)

- Estimate p_i with the frequency of AA a_i over both sequences, i.e., number of a_i's/number of AAs
- Let $f_{ij} = f_{ji} =$ number of $a_i \leftrightarrow a_j$ changes in data set, $f_i = \sum_{j \neq i} f_{ij} =$ number of changes involving a_i, and $f = \sum_i f_i =$ number of changes
- Define the scale to be the amount of evolution to change 1 in 100 AAs (on average) [1 PAM dist]
- Relative mutability of a_i is the ratio of number of mutations to total exposure to mutation: $m_i = f_i / (100 f \, p_i)$

The PAM Transition Matrices (cont'd)

- If m_i is probability of a mutation for a_i, then $M_{ii} = 1 - m_i$ is prob. of no change
- $a_i \to a_j$ if and only if a_i changes and $a_i \to a_j$ given that a_i changes, so

$$\begin{aligned} M_{ij} &= Pr(a_i \to a_j) \\ &= Pr(a_i \to a_j \mid a_i \text{ changed}) Pr(a_i \text{ changed}) \\ &= (f_{ij}/f_i)\, m_i = f_{ij}/(100 f\, p_i) \end{aligned}$$

- The 1 PAM transition matrix consists of the M_{ij} and gives the probabilities of mutations from a_i to a_j

Properties of PAM Transition Matrices

$$\begin{aligned}
\sum_j M_{ij} &= \sum_{j \neq i} M_{ij} + M_{ii} \\
&= 1/(100 f\, p_i) \sum_{j \neq i} f_{ij} + (1 - f_i/(100 f\, p_i)) \\
&= f_i/(100 f\, p_i) + 1 - f_i/(100 f\, p_i) = 1
\end{aligned}$$

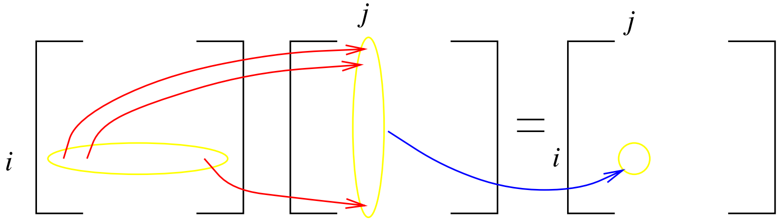
[sum of probabilities of changes to an AA + prob of no change = 1]

$$\sum_i p_i M_{ii} = \sum_i p_i - \sum_i f_i/(100 f) = 1 - f/(100 f) = 0.99$$

[prob of no change to any AA is 99/100]

What About 2 PAM?

- How about the probability that $a_i \rightarrow a_j$ in two evolutionary steps?
- It's the prob that $a_i \rightarrow a_k$ (for any k) in step 1, and $a_k \rightarrow a_j$ in step 2. This is $\sum_k M_{ik} M_{kj} = M_{ij}^2$



k PAM Transition Matrix

- In general, the probability that $a_i \rightarrow a_j$ in k evolutionary steps is M_{ij}^k
- As $k \rightarrow \infty$, the rows of M^k tend to be identical with the ith entry of each row equal to p_i
  - A result of our Markovian assumption of mutation

Building a Scoring Matrix

- When aligning different AAs in two sequences, want to differentiate mutations and random events
- Thus, interested in ratio of transition probability to prob. of randomly seeing new AA

$$\frac{M_{ij}}{p_j} = \frac{f_{ij}}{100 f\, p_i\, p_j} = \frac{M_{ji}}{p_i} \qquad \text{(symmetric)}$$

- Ratio > 1 if and only if mutation more likely than random event

Building a Scoring Matrix (cont'd)

When aligning multiple AAs, ratio of probs for multiple alignment = product of ratios:

$$\begin{array}{cccc} a_i & a_k & a_n & \cdots \\ a_j & a_\ell & a_m & \cdots \end{array} \longrightarrow \left(\frac{M_{ij}}{p_j}\right)\left(\frac{M_{k\ell}}{p_\ell}\right)\left(\frac{M_{nm}}{p_m}\right)\cdots$$

Taking logs will let us use sums rather than products

$\Rightarrow$ "Log odds"

$\Rightarrow$ Avoid underflow issues

Building a Scoring Matrix (cont'd)

- Final step: Computation faster with integers than with reals, so scale up (to increase precision) and round:

$$s_{ij} = C \log_2 \left(\frac{M_{ij}}{p_j} \right)$$

- C is a scaling constant
- For k PAM, use M_{ij}^k

Building a Scoring Matrix (cont'd)

Table 1 – The log odds matrix for 250 PAMs (multiplied by 10)

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	2	-2	0	0	-4	1	-1	-1	-1	-2	-1	0	1	0	-2	1	1	0	-6	-3
C		12	-5	-5	-4	-3	-3	-2	-5	-6	-5	-4	-3	-5	-4	0	-2	-2	-8	0
D			4	3	-6	1	1	-2	0	-4	-3	2	-1	2	-1	0	0	-2	-7	-4
E				4	-5	0	1	-2	0	-3	-2	1	-1	2	-1	0	0	-2	-7	-4
F					9	-5	-2	1	-5	2	0	-4	-5	-5	-4	-3	-3	-1	0	7
G						5	-2	-3	-2	-4	-3	0	-1	-1	-3	1	0	-1	-7	-5
H							6	-2	0	-2	-2	2	0	3	2	-1	-1	-2	-3	0
I								5	-2	2	2	-2	-2	-2	-2	-1	0	4	-5	-1
K									5	-3	0	1	-1	1	3	0	0	-2	-3	-4
L										6	4	-3	-3	-2	-3	-3	-2	2	-2	-1
M											6	-2	-2	-1	0	-2	-1	2	-4	-2
N												2	-1	1	0	1	0	-2	-4	-2
P													6	0	0	1	0	-1	-6	-5
Q														4	1	-1	-1	-2	-5	-4
R															6	0	-1	-2	2	-4
S																2	1	-1	-2	-3
T																	3	0	-5	-3
V																		4	-6	-2
W																			17	0
Y																				10

PAM Scoring Matrix Miscellany

- Pairs of AAs with similar properties (e.g., hydrophobicity) have high pairwise scores, since similar AAs are more likely to be accepted mutations
- In general, low PAM numbers find short, strong local similarities and high PAM numbers find long, weak ones
- Often multiple searches will be run, using e.g., 40 PAM, 120 PAM, 250 PAM
- Altschul (*JMB*, 219:555–565, 1991) gives discussion of PAM choice

BLOSUM Scoring Matrices

- Based on multiple alignments, not pairwise
- Direct derivation of scores for more distantly related proteins
- Only possible because of new data: Multiple alignments of known related proteins

BLOSUM Scoring Matrices (cont'd)

- Started with ungapped alignments from BLOCKS database
- Sequences clustered at L% sequence identity
- This time, f_{ij} = # of $a_i \leftrightarrow a_j$ changes between pairs of sequences from different clusters, normalizing by dividing by $(n_1 n_2)$ = product of sizes of clusters 1 and 2
- $f_i = \sum_j f_{ij}, \quad f = \sum_i f_i$ (different from PAM)
- Then the scoring matrix entry is

$$s_{ij} = C \log_2 \left(\frac{f_{ij} / f}{p_i \, p_j} \right)$$

BLOSUM 50 Scoring Matrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2

Gap Penalties

- A gap can be inserted in a sequence to better align downstream residues, e.g., alignments 2 & 3 on slide 4
- Two widely-used types of scoring functions:
  - Linear: $\gamma(g) = -gd$, where g is gap length and d is gap-open penalty (often choose $d = 8$)
  - Affine: $\gamma(g) = -d - (g-1)e$, where e is gap-extension penalty (often choose $d = 12$, $e = 2$)
- Vingron & Waterman (*JMB*, 235:1–12, 1994) discuss penalty function choice in more detail

How do we do it?

- Choose a scoring scheme
- Choose an algorithm to find optimal alignment wrt scoring scheme
- Statistically validate alignment

Optimal Alignment Algorithms

- To find the best alignment, we can simply try all possible alignments of the two sequences, score them, and choose the best
- Will this work?

Optimal Alignment Algorithms

NO!

- The number of alignments grows with $\binom{2n}{n}$, e.g., $n = 100$ residues/sequence $\Rightarrow > 9 \times 10^{58}$ alignments!
- So now what do we do?
  - Pull dynamic programming out of our algorithm toolbox
  - We'll see that optimal alignments of substrings are part of an optimal alignment of the larger strings

Types of Alignments

- Will discuss DP algs for these types of alignments between seqs. x and y:
  - Global: Align all of x with all of y
    - $\Rightarrow$ Useful when testing homology between two similarly-sized sequences
  - Local: Align a substring of x with a substring of y
    - $\Rightarrow$ Useful when finding shared subsequences between proteins
  - Semiglobal ("Overlap"): Same as global, but ignore leading and/or trailing blanks
    - $\Rightarrow$ Useful when doing fragment assembly
- For now, assume linear gap penalty

Global Alignment

- Let $F(i,j)$ = score of best alignment between $x_{1\ldots i}$ and $y_{1\ldots j}$
- Given $F(i-1,j-1)$, $F(i-1,j)$, and $F(i,j-1)$, what is $F(i,j)$?
- Three possibilities:

1. x_i aligned with y_j, e.g.,

I	G	A	x_i
L	G	V	y_j

$\Rightarrow F(i,j) = F(i-1,j-1) + s(x_i, y_j)$

2. x_i aligned with gap, e.g.,

A	I	G	A	x_i
L	G	V	y_j	$-$

$\Rightarrow F(i,j) = F(i-1,j) - d$

3. y_j aligned with gap, e.g.,

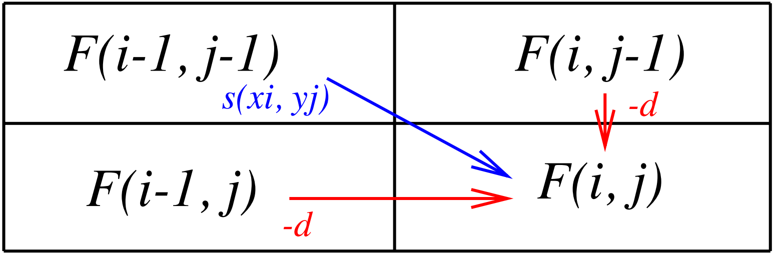
G	A	x_i	$-$	$-$
S	L	G	V	y_j

$\Rightarrow F(i,j) = F(i,j-1) - d$

Global Alignment (cont'd)

- Final update equation:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$



- Boundary conditions: $F(i,0) = -id$, $F(0,j) = -jd$

Global Alignment (cont'd)

- Score of optimal global alignment is in $F(n, m)$
- The alignment itself can be recovered if, for each $F(i, j)$ decision, we kept track of which cell gave the max
  - Follow this path back to origin, and print alignment as we go
  - Figure 2.5, p. 21

Local Alignment

- Similar to global alignment algorithm
- Differences:
  1. If an alignment's score goes negative, it's better to start a new one

$$F(i,j) = \max \begin{cases} 0 \\ F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}, \quad F(i,0) = F(0,j) = 0$$

  2. Score of opt. align. is $\max_{i,j}\{F(i,j)\}$; end traceback at 0 score
- Figure 2.6, p. 23
- Must have expected score < 0 for rand. match and need some $s(a,b) > 0$

Overlap Matches (a.k.a. Semiglobal Alignment)

Which is better?

```
CAGCA-CTTGGATTCTCGG        CAGCACTTGGATTCTCGG
---CAGCGTGG--------        CAGC-----G-T----GG
```

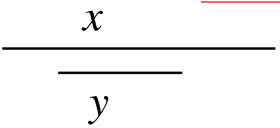
Overlap Matches (a.k.a. Semiglobal Alignment)

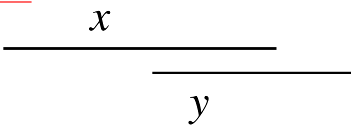
If match $= +1$, mismatch $= -1$ and gap $= -2$,

```
CAGCA-CTTGGATTCTCGG
---CAGCGTGG--------
       -19
```

```
CAGCACTTGGATTCTCGG
CAGC-----G-T----GG
       -12
```

Ignoring end spaces will allow us to constrain alignment to containment or prefix-suffix overlap





Overlap Matches (cont'd)

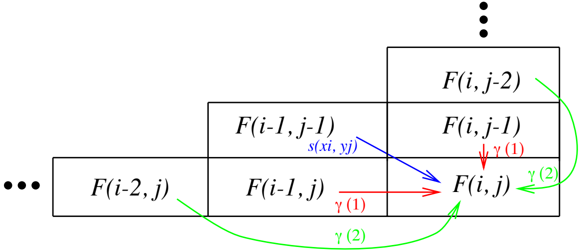
- $F(i, 0) =$ $\qquad F(0, j) =$
- Score of optimal alignment =
- $F(i, j) =$
- Figure 2.8, p. 27

General Gap Penalty Functions

If gap penalty $\gamma(g)$ not linear, can still do optimal alignment:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ \max_{k=0,\ldots,i-1}\{F(k,j) + \gamma(i-k)\} \\ \max_{k=0,\ldots,j-1}\{F(i,k) + \gamma(j-k)\} \end{cases}$$

$$F(0,j) = \gamma(j)$$
$$F(i,0) = \gamma(i)$$



Time complexity now $\Theta(n^3)$, versus $\Theta(n^2)$ for old alg

Affine Gap Penalty Functions

- If gap penalty an affine function, can run in $\Theta\left(n^2\right)$ time
- Use 3 arrays:
  1. $M(i,j)$ = best score to (i,j) when x_i aligns y_j (case 1)
  2. $I_x(i,j)$ = best score when x_i aligns gap (case 2); insert. in x wrt y
  3. $I_y(i,j)$ = best score when y_j aligns gap (case 3)

$$M(i,j) = s(x_i, y_j) + \max \begin{cases} M(i-1,j-1) \\ I_x(i-1,j-1) \\ I_y(i-1,j-1) \end{cases}$$

$$I_x(i,j) = \max \begin{cases} M(i-1,j) - d \\ I_x(i-1,j) - e \end{cases}$$

$$I_y(i,j) = \max \begin{cases} M(i,j-1) - d \\ I_y(i,j-1) - e \end{cases}$$

Affine Gap Penalty Functions (cont'd)

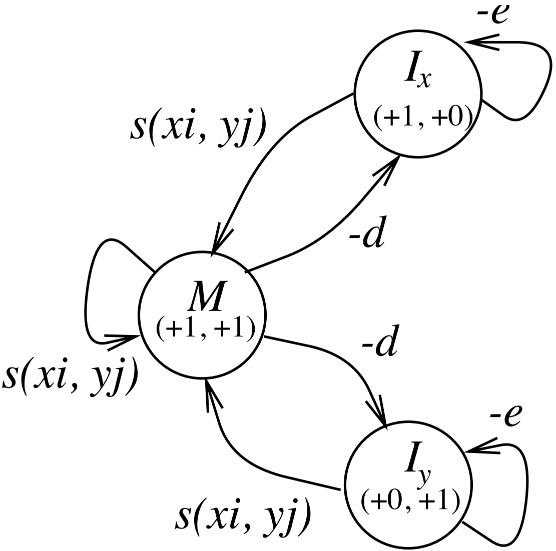
$$M(i,j) = s(x_i, y_j) + \max \begin{cases} M(i-1,j-1) \\ I_x(i-1,j-1) \\ I_y(i-1,j-1) \end{cases}$$

$$I_x(i,j) = \max \begin{cases} M(i-1,j) - d \\ I_x(i-1,j) - e \end{cases}$$

$$I_y(i,j) = \max \begin{cases} M(i,j-1) - d \\ I_y(i,j-1) - e \end{cases}$$

$$\begin{aligned} M(0,0) &= 0, & M(i,0) = M(0,j) &= -\infty \\ I_x(0,j) &= -\infty, & I_x(i,0) &= -d - (i-1)\,e \\ I_y(i,0) &= -\infty, & I_y(0,j) &= -d - (j-1)\,e \end{aligned}$$

Affine Gap Penalty Functions (cont'd)



Heuristic Alignment Algorithms

- Linear (vs. quadratic) time complexity
  - Important when making several searches in large databases
- Don't guarantee optimality, but very good in practice
- BLAST
- FASTA

BLAST

- Uses e.g., PAM or BLOSUM matrix to score alignments
- Returns substring alignments with strings in database that score higher than threshold S and are longer than min length
- Does not return string if it's a substring of another and scores lower
- Tries to minimize time spent on alignments unlikely to score higher than S

BLAST Steps

1. Find short words (strings) that score high when aligned with query
2. Use these words to search database for hits (each hit will be a seed for next step). Each hit will score $= T < S$ to help avoid fruitless pursuits (lower $T \Rightarrow$ less chance of missing something & higher time complexity)
3. Extend seeds to find matches with maximum score

Find High-Scoring Words

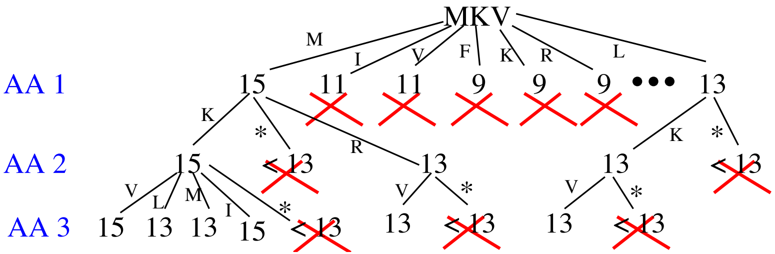
List all words w characters long (w-mers) that score $\geq T$ with some query w-mer

- Pass a width-w window over the query and generate the strings that score $\geq T$ when aligned

```
Query: VTP|MKV|IVFC      T=13, w=3 (PAM 250)
           MKV           score = 6 + 5 + 4 = 15
           LKV           score = 13
           MRV           score = 13
           MKL           score = 13
           MKI           score = 15
           MKM           score = 13
```

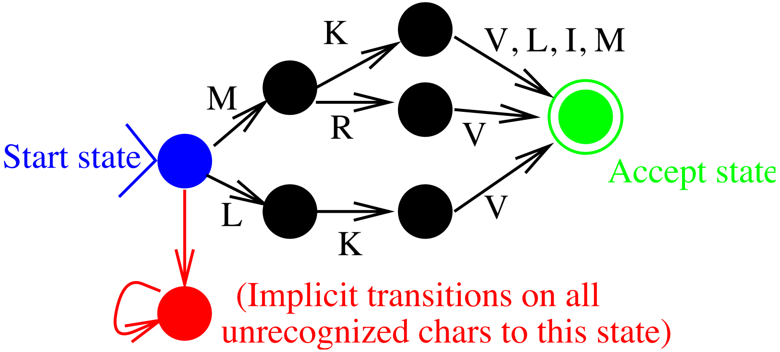
Find High-Scoring Words (cont'd)

- Often use $w = 3$ or 4 characters and $T = 11$
- At most 20^w total w-mers
- $\Rightarrow$ So 160000 w-mers for $w = 4$, 8000 for $w = 3$
- Can quickly find all with brute force, or save time with branch-and-bound (assume $T = 13$):



Search for Hits

- Hit = subsequence in data base that matches a high-scoring word from previous step
- To improve efficiency, represent set of high-scoring words with a DFA



Extending the Seeds

- Take each hit (seed) and extend it in both directions until score drops below best score so far minus buffer score
- E.g., if buffer = 4, extend to right, then left:

```
                              13 = original seed score
                              | |
   Query:        VT | PMKVIV | FCW
Database:    ... WW | AMKLKV | GWW ...
                  1   1   1   1
                  1   6   1   0
                            1
                            5
```

So match PMKVIV with AMKLKV for a score of 16

Extending the Seeds (cont'd)

- This is a linear-time greedy heuristic to increase speed
- Can miss better matches, e.g., if W-W or C-C pairs are near:

```
                    stop here
   Query:         VTPMKVIV | FCW | C
Database:    ...  WWAMKLKV | GWW | W ...
                                 I want to get here
```

- Increasing buffer will increase sensitivity, at the cost of increased time
- Choosing good values of parameters makes small the probability of missing a better match

BLAST: Time Complexity

- Expected-time computational complexity: $O(W + Nw + NW/20^w)$ to generate word list, find hits & extend hits
    - $W =$ number of high-scoring words generated and $N =$ number of residues in database ($M =$ query size is embedded in W)
    - Can make Nw into N by replacing DFA with hash table
- Versus $O(NM)$ for dynamic programming, where $M =$ number residues in query

BLAST: Additions

- Gapped BLAST: Allows gaps in local alignments
  - Better reflects biological relationships
  - Less efficient than standard BLAST
- Position-Specific Iterated (PSI) BLAST: Starts with a gapped BLAST search and adapts the results to a new query sequence for more searching
  - Automated "profile" search
  - Less efficient than standard BLAST

FASTA

1. Start by finding k-tuples common to both sequences ($ktup = 1$ or 2)
   - Done with lookup table and offset vector

```
    1 2 3 4 5 6 7 8 9 10 11        1 2 3  4  5  6  7  8
s = H A R F Y A A Q I  V  L    t = V D M  A  A  Q  I  A
LOOKUP TABLE                      +9     -2 -3 +2 +2 -6
A 2,6,7                        OFFSETS   +2 +1       -2
F 4      L 11                            +3 +2       -1
H 1      Q 8
I 9      R 3
V 10     Y 5

OFFSET VECTOR
-7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 +9
 0  1  0  0  1  2  1 0  1  4  1  0  0  0  0  0  1
                           /\
```

FASTA (cont'd)

2. Extend the exact word matches to find maximal scoring ungapped regions (similar to BLAST)
3. Ungapped regions are joined into gapped regions, accounting for gap costs
4. Realign candidate matches using full dynamic programming

- Increasing *ktup* improves speed but increases chance of missing true matches

How do we do it?

- Choose a scoring scheme
- Choose an algorithm to find optimal alignment wrt scoring scheme
- Statistically validate alignment

Statistically Validating Alignments

- Once we take our highest-scoring hits, are we done?
  - What if none of the hits was good enough?
  - What is our threshold (minimum) score?
- Given a particular score, want a bound on the probability that a random sequence would get at least that score
  - Such a probability is given by an extreme value distribution (EVD)

EVD for Sequence Comparisons

[Karlin & Altschul 1990]

- Let λ be the unique positive solution to

$$\sum_{i,j} p_i p_j \exp(\lambda s_{ij}) = 1$$

- If the two aligned sequences are of length m and n, then the probability that a score S can occur with a random match is bounded by

$$P\left(S > \frac{\ln mn}{\lambda} + x\right) \leq K \exp(-\lambda x) ,$$

  where K is given in the paper
- So e.g., if x is such that $K \exp(-\lambda x) = 0.01$, then any score $S \geq x + (\ln mn)/\lambda$ has a 99% chance of being significant
  - Allows us to assess significance of any score and/or to set a threshold on minimum score