chaining algorithms
multiple alignment
Jobs

- Applied Predictive Technologies – looking for the best students – focus on databases (forwarded by Daniel Hackner) -not bioinformatics
Chaining in 1-D

• Input: multiple overlapping intervals on a line
• Output: highest weight set of non-overlapping intervals
• Weight could be length of interval, or Smith-Waterman score, etc.

• Rationale? The pattern can have multiple inconsistent exact matches to the text – we want to pick a longest consistent set
Path “planning” and dynamic programming

• One intuitive way to think about dynamic programming
  – similar to finding shortest path between two points
  – at each “point” ask – what are all possible ways to get here?
  – pick the best (shortest, fastest, etc.)
Chaining in 1D

- Sort the endpoints (starts, ends) of the intervals
- For every interval $j$, store $V[j]$ – best score of a chain ending in $j$
- $MAX$ – store highest $V[j]$ seen sofar
- Process endpoints in increasing order of $x$ coordinate
- If we encounter left end (start) of interval $j$
  - $V[j] = weight(j) + MAX$
- If we encounter right end (end) of interval $j$
  - $MAX = \max\{V[j], MAX\}$

Running time?
Chaining in 2-D

• Easy to do in $O(n^2)$ (n - # of intervals)
• View alignments as "boxes"
• All boxes in a chain must follow each other in a "diagonal" order, i.e. the range of the x coordinates and y coordinates of any two boxes in a chain cannot overlap
• Similar to 1-D approach except at each step we must check if current box can extend any of the previously built chains
• $V[j] = \max_{\text{all previous boxes } k} \{V[k] + \text{weight}(j)\}$
• More complex algorithm leads to $O(n \log n)$ running time
Multiple sequence alignment
Multiple sequence alignment

- Simultaneously identify relationship between multiple sequences

Note: multiple alignment implies (not necessarily optimal) pairwise alignment between the individual sequences
Multiple alignment – formal definition

- $M$ – multiple sequence alignment for $s_1, \ldots, s_k$
- $D(s_i, s_j)$ – optimal score of alignment between $s_i$, $s_j$
- $d(s_i, s_j)$ – score of alignment between $s_i$, $s_j$ induced by $M$
- score of $M$ $d(M) = \sum_{\text{all pairs } s_i, s_j} d(s_i, s_j)$
- also called sum-of-pairs

- Optimal multiple alignment minimizes $d(M)$

- Computing optimal $d(M)$ is NP hard
- Note: in multiple alignment we think of "distance" rather than "similarity"
But....here's a solution

- Dynamic programming solution. e.g. 3 sequences

- Score(i, j, k) – optimal alignment between s1[1..i], s2[1..j], s3[1..k] – do DP as usual

- $s(i,j,k) = \max \{ s(i-1, j-1, k-1) + \text{match}(s1[i], s2[j], s3[k]), ... \}$
But... it's expensive

- 3 sequences – need to fill in the cube $O(n^3)$
- $k$ sequences – $k$-dimensional cube $O(n^k)$ time/space

- There are tricks that can help – similar to AI techniques for reducing the search space

- Basic idea – if we can estimate optimal score, we can prune the search space.

- Note – these are just heuristics – not guaranteed to work faster
Alternative – approximation algorithm

• Can we efficiently compute a multiple alignment with a score that's not too bad?

• The Star method:
  – build all $k^2$ pairwise alignments ($O(k^2n^2)$)
  – pick sequence $sc$ that is closest to all other sequences: $\sum_{s_i} D(sc, s_i)$ is minimal over all choices of $sc$
  – iteratively align each sequence to $sc$

• Theorem: sum-of-pairs score of star alignment is at most twice as big as optimal multiple alignment score
Iterative alignment

• Take sequences $s_i$ in order:
  – align $s_1$ with $sc$ - results in gaps being inserted in both sequences
    
    SC  YFPHFDLSHGSAQVKAHGKKVGDALTLAVGHLDDLP GAL
    S1  YFP HFDLSHG-AQVKG--KKVADALTNAV AHDVDD MPNAL

  – align $s_2$ with $sc$ - if gaps must be inserted – insert in previously aligned sequences
    
    SC  YFPHF-DLS-----HGSAQVKAHGKKVGDALTLAVGHLDDLP GAL
    S1  YFP HF-DLS-----HG-AQVKG--GKKVA-----DALTN AVAHVD MPNAL
    S2  FFPFKFKGLTTADQLKSADVRWHAERI I-----NAV NDAVASMDD T EKMS

  – and so on (note: if gaps coincide with previously introduced gaps no need to change previously aligned sequences)
    
    SC  YFPHF-DLS-----HGSAQVKAHGKKVGDALTLAVGHLDDLP GAL
    S1  YFP HF-DLS-----HG-AQVKG--GKKVA-----DALTN AVAHVD MPNAL
    S2  FFPFKFKGLTTADQLKSADVRWHAERI I-----NAV NDAVASMDD T EKMS
    S3  LFSFLKGTSEVP--QNNPELQAHAGKVF KLAV YE AA IQLQVTV GVVTVDATL
Theorem proof

- Theorem: star alignment is 2-optimal
- Assumption: distances obey triangle inequality

\[ \text{OPT} = \sum_{s_i,s_j} d^*(s_i,s_j) \geq \sum_{s_i,s_j} D(s_i,s_j) \geq k \sum_{s_i} D(s_i, sc) \]

\[ \text{STAR} = \sum_{s_i,s_j} d(s_i,s_j) \leq \sum_{s_i,s_j} (D(s_i, sc) + D(s_j, sc)) \text{ # triangle ineq.} \]
\[ = \sum_{s_j,s_j} D(s_j, sc) + \sum_{s_j,s_j} D(s_i, sc) \]
\[ = 2k \sum_{s_i} D(s_i, sc) \]

\[ \Rightarrow \text{STAR/OPT} \leq 2 \quad \text{Q.E.D} \]

Note: \( \sum_{s_i} D(s_i, sc) \) – is score optimized by choice of sc

\( d^*(si,sj) \) – score of alignment b/w \( si, sj \) within optimal alignment

\( d(si,sj) \) – score of alignment b/w \( si, sj \) within star alignment

\( D(si,sj) \) – score of optimal alignment b/w \( si, sj \)