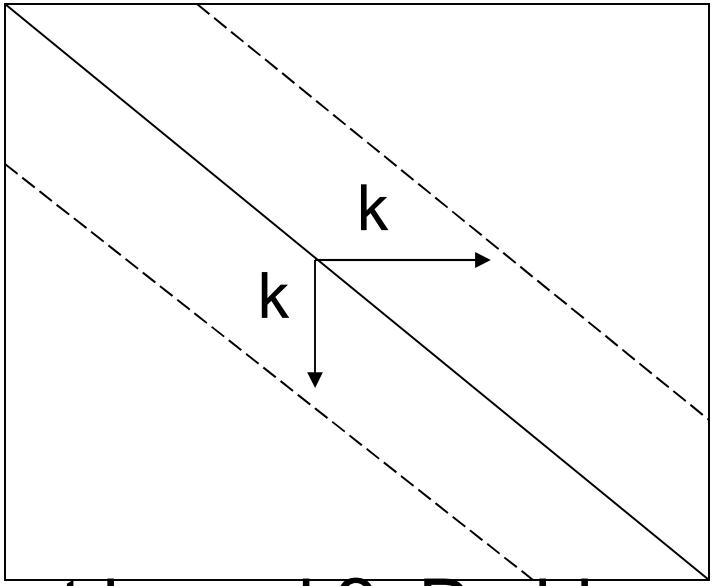
CMSC423: Bioinformatic Algorithms, Databases and Tools

Alignment heuristics

Heuristics

- What if limit the # of differences allowed? E.g. we expect the sequences to be very similar.
- Compute 'banded' alignment – stay within # of differences (k) from the diagonal.
- Optimal alignment cannot stray too far from diagonal



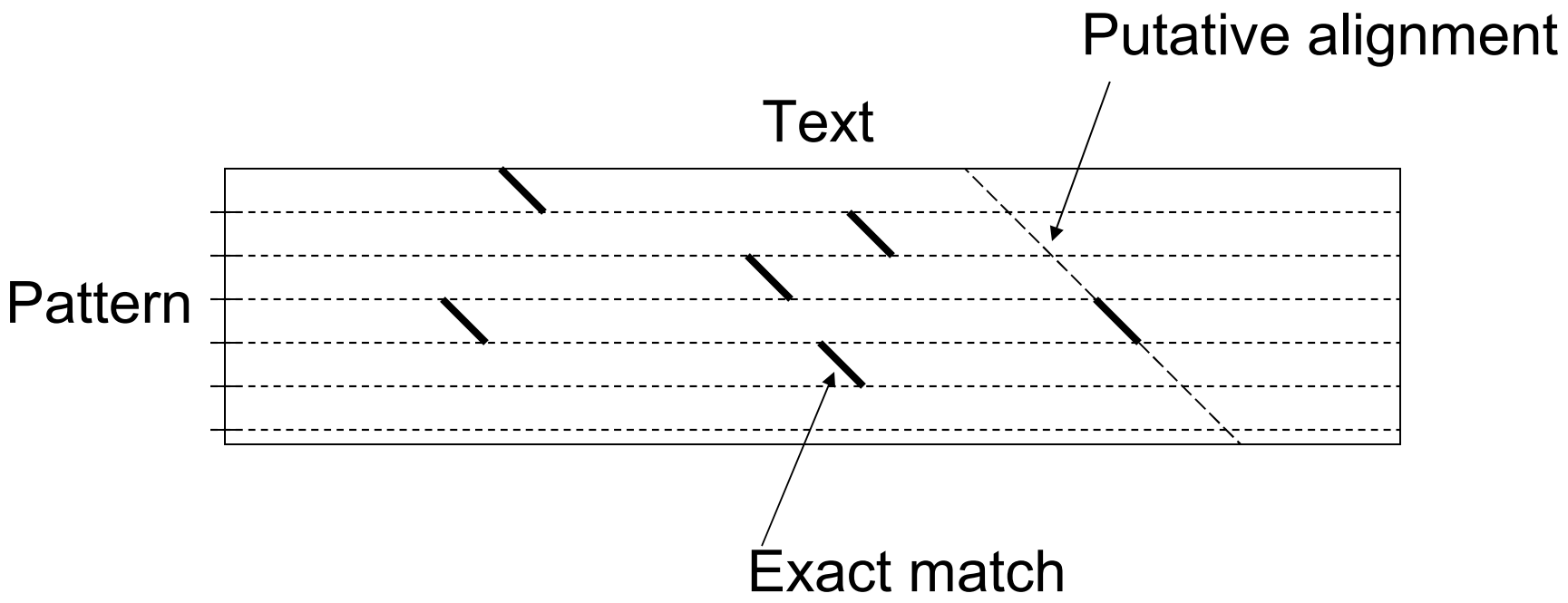
O(km) running time and space

- What if we do not know k? Do binary search to find it

Exclusion methods

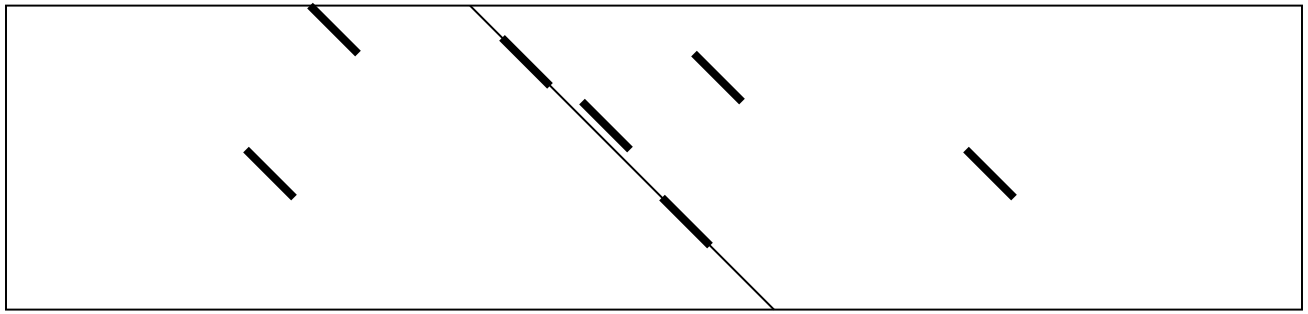
- Assume P must match T with at most k errors. Find places in T where P cannot match.
- Split P into **floor(n/k+1)**-sized chunks.
- If P matches T with less than k errors => at least one chunk matches with no errors
- Use any exact matching algorithm to find places where a chunk matches T, then run dynamic programming in that vicinity.
- Running time, on average O(m)

Exclusion methods



"Famous" approaches

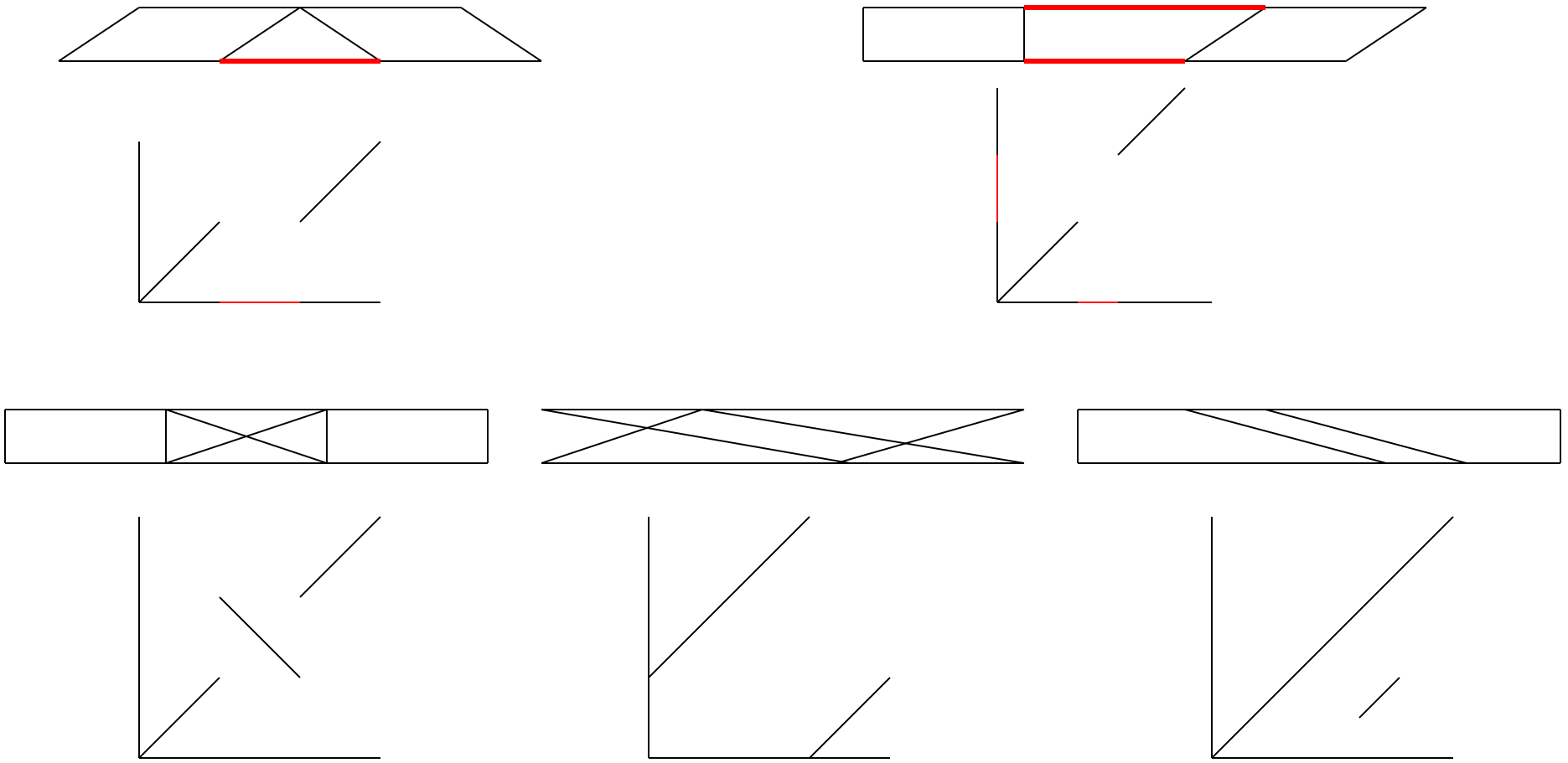
- FASTA (Pearson et al.)
  - Take all k-mers (substrings of length k) from Pattern and identify whether and where they match in the Text
  - Assume the k-mer starting at pos'n i in Pattern matches at position j in Text, remember (j – i) – the diagonal on which the match occured
  - Identify "heavy" diagonals – diagonals where many k-mers match, then refine the diagonals with Smith Waterman
  - Also look for off-diagonal matches to account for gaps



"Famous" approaches

- BLAST (Altschul et al.)
  - Find short k-mer matches
  - Also search for possible inexact matches, e.g. all k-mers within 1 difference from current one.
  - Extend exact matches with Smith-Waterman algorithm
  - Assign probabilistic scores to matches: what is the probability of finding a match with the same S-W alignment score just by chance (e.g. matching a random string)?

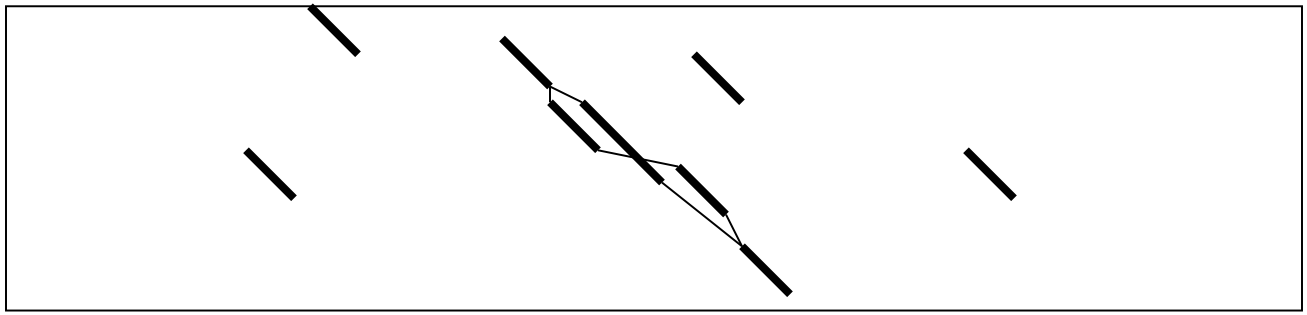
Aside: viewing alignments with dot-plots



axes – two sequences/genomes, 'dots' – regions that match in the two genomes

Chaining approach

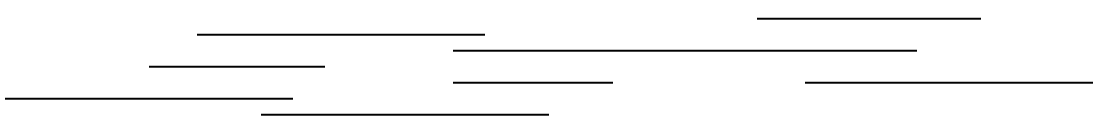
- Extends the FASTA idea
- Search for exact matches
- Find the longest consistent chain of exact matches
- Fill in the gaps in the chain using Smith-Waterman



- This is the approach used by MUMmer (Delcher et al.)
- MUM – maximally unique match (see mummer.sourceforge.net)

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.



Chaining in 1D

- Basic idea – dynamic programming
- V[j] – weight of best chain ending with interval j
- $V[j] = \max_{k < j,\ \text{intervals } k \text{ \& } j \text{ do not overlap}}(V[k] + \text{weight}(j))$
- i.e. find all possible ways of building a chain ending at j and pick the best one (the key to all dynamic programming algorithms)
- Where do we find the answer? largest value in V array
- How do we find the actual chain? backtracking
- What is the running time? O(n^2)

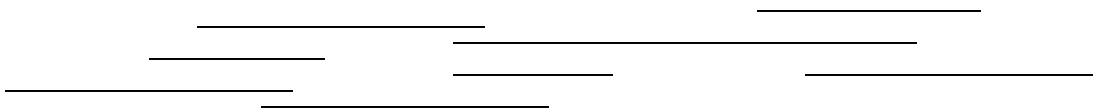
Chaining in 1D

- 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?

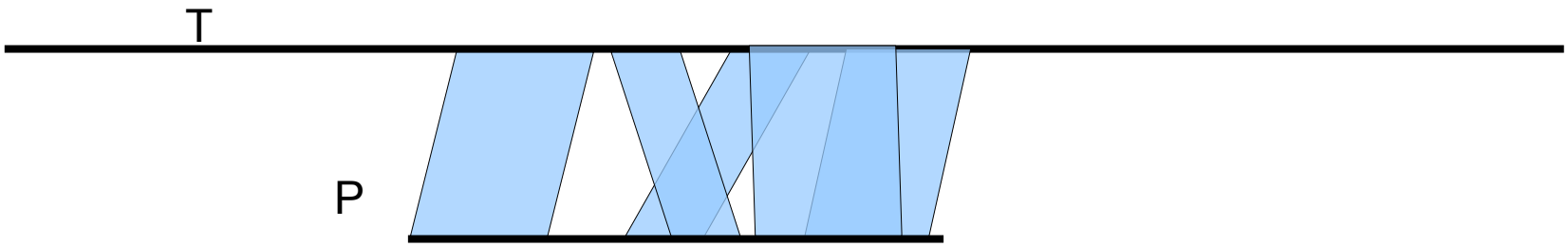
O(n log n) – from sorting

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.

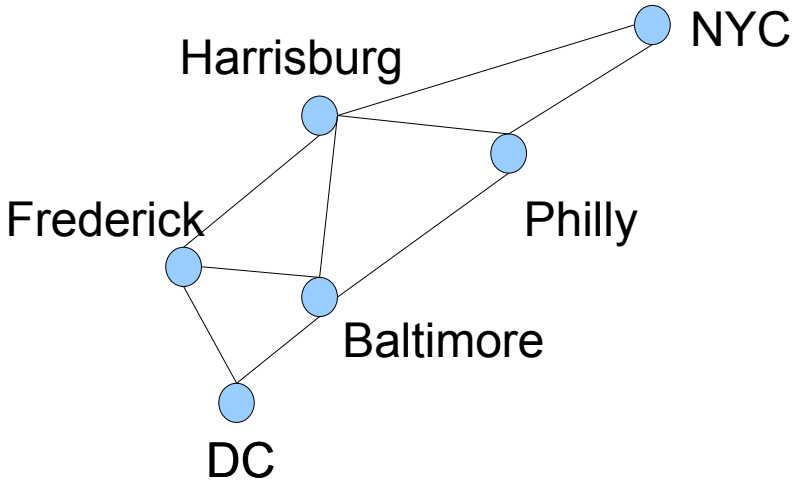


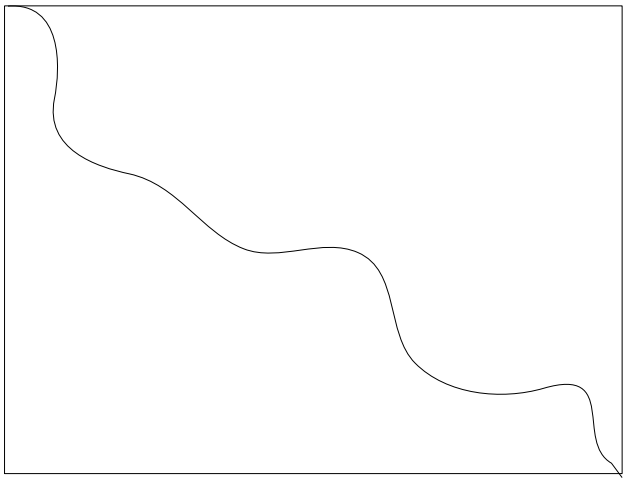
- Rationale? The pattern can have multiple inconsistent exact matches to the text – we want to pick a longest consistent set



Path “planning” and dynamic programming

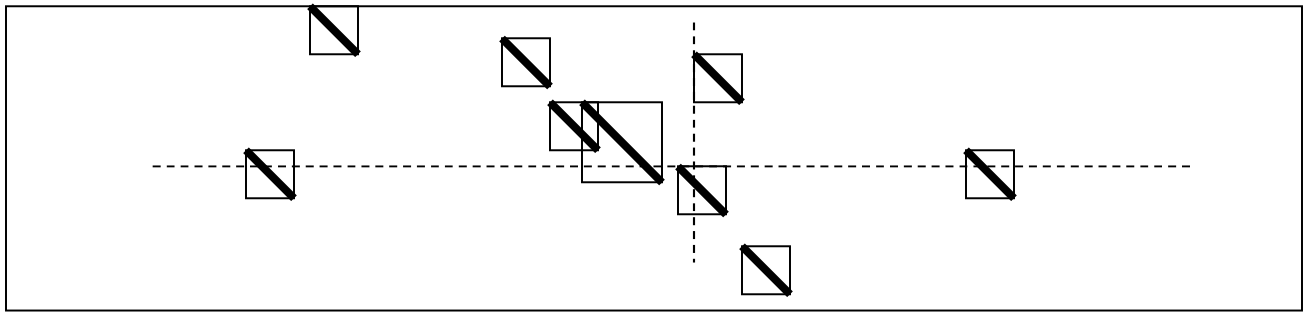
- One intuitive way to think about dynamic programming
  - similar to finding shortest path between two points
  - at each “point” ask – what are all possible ways to get here?
  - pick the best (shortest, fastest, etc.)





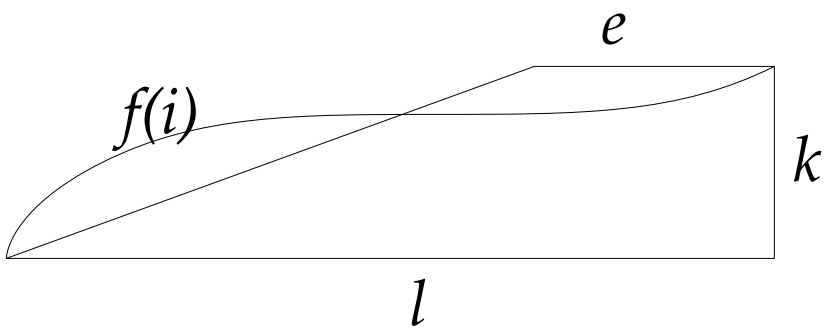
Chaining in 2-D

- 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



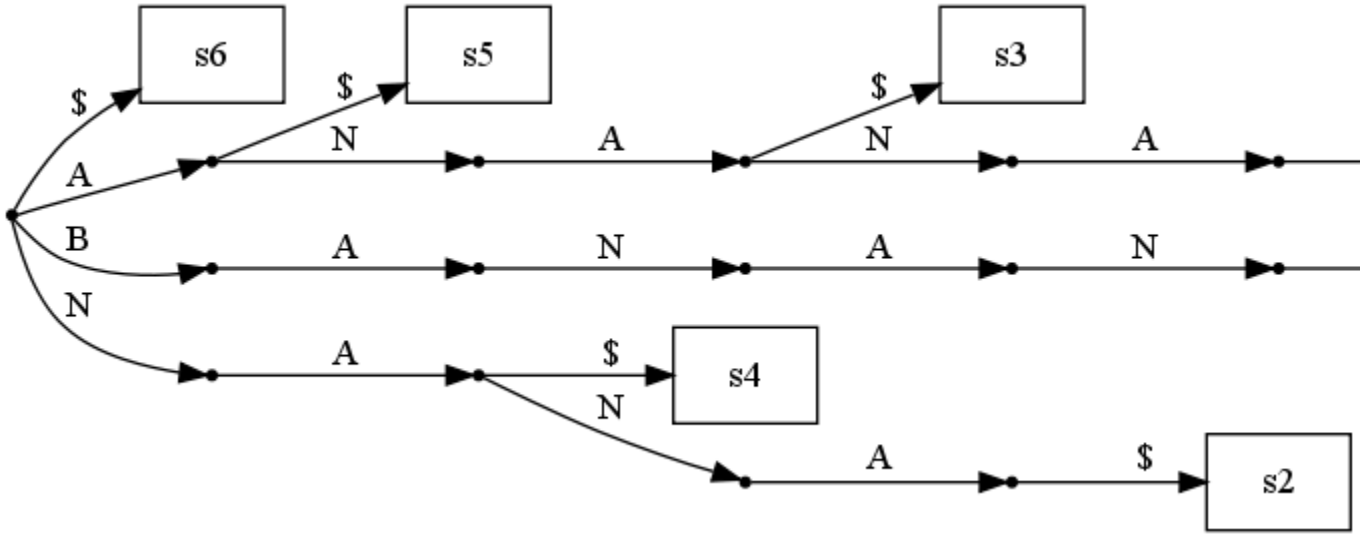
Suffix trees + dynamic programming

- Idea: find inexact seeds (rather than exact matches)
- Observation: if two sequences match within x% identity there must be some short subsequence that also matches with at least x% identity



- Why is this useful? You can backtrack quickly if error rate exceeded (short sequences will have to be almost perfect).

suffix trees + dynamic programming



?	?	?	?	?	?		
A	3	2	1	1	?		
N	2	1	0	1	?		
A	1	0	1	2	?		
-	0	1	2	3	?		
	-	A	N	N	?		