Lecture 12

- More on WDAGs:
  - Inverted WDAGs, fwd/backwd algorithm
  - Finding *multiple* high-scoring paths
- Multiple paths in edit graphs
  - Internal repeats
- Multiple paths in WLLs
- “D-segments”

Inverted WDAGs

- Can "invert" any WDAG: create graph with
  - same vertices & edge weights
  - direction of each edge reversed
  - is still acyclic!
- inverted WDAG has same paths (& path weights), but in reverse direction
  - "*forward*" path in inverted WDAG = "*backward*" path in original WDAG (& vice versa)

Forward/backward algorithm

- Order vertices (v_1, v_2, ..., v_n) with parents preceding children.
  - Reverse order (v_n, v_{n-1}, ..., v_1) has parents before children in *inverted* graph
- (Forward direction) Find $w(v)$

  = highest weight of all paths ending at v in ***original*** (non-inverted) graph
- (Backward direction) Using inverted graph, find $w'(v)$

  = highest weight of all paths ending at v in ***inverted*** graph

  = highest weight of all paths *beginning* at v in ***original*** graph
- joining path ending at v, to path beginning at v (in *original* graph),

  see that $w(v) + w'(v)$ = highest weight of any path going *through* v.

Finding *multiple* high-scoring paths

- If high-weight paths are important, we'll want more than one!
  - But *not* slight perturbations of highest-weight path
- 'Brute force' algorithm:
  - Find highest-weight path
  - 'Mask it' (remove its edges from graph)
  - Repeat above two steps until scores 'uninteresting'
    < some threshold value S
  - can be $O(N^2)$, but often acceptable

Improving on ‘brute force’ by graph reduction

- Use forwd/backwd to find $w(v)$, $w'(v)$
- Eliminate v (& all its edges) if $w(v) + w'(v) < S$
- Eliminate all edges into v if $w(v) \leq 0$
- Eliminate all edges out of v if $w'(v) \leq 0$
- Remaining graph is often much smaller & splits into ‘connected components’ which can be processed separately
  - v, v' in same c.c. if a chain of edges connected them
- ***But*** no guarantee that < $O(N^2)$

- Is there an $O(N)$ algorithm?
  - Yes, for WLLs (Ruzzo & Tompa)

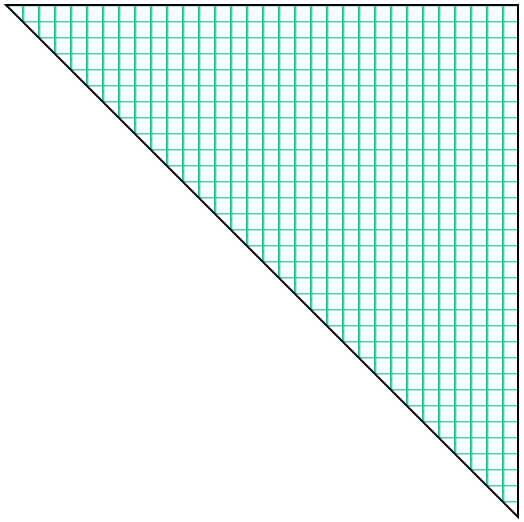
Finding (imperfect) internal repeats

- Search edit graph of *sequence against itself*
  - i.e. the same sequence labels columns and rows

above (& not including) the main diagonal:

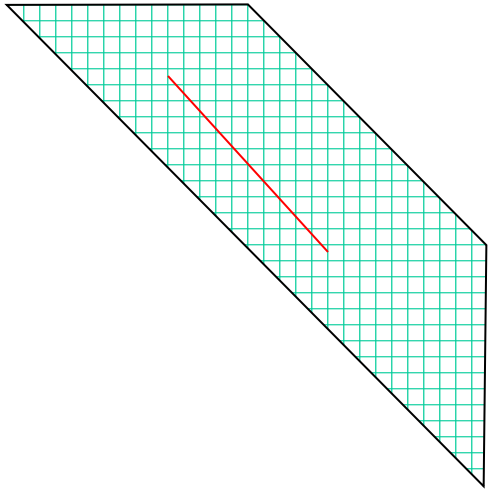
  - if include main diagonal, best path will be identity match to self
  - complexity = $O(N^2)$ where N = sequence length.

Graph for finding imperfect internal repeats:



- Find *short tandem repeats* (e.g. microsatellites, minisatellites):
  - scan a *band* just above main diagonal.
  - Complexity = $O(kN)$ where k is width of the band.
  - Manageable even for large N, if k small.

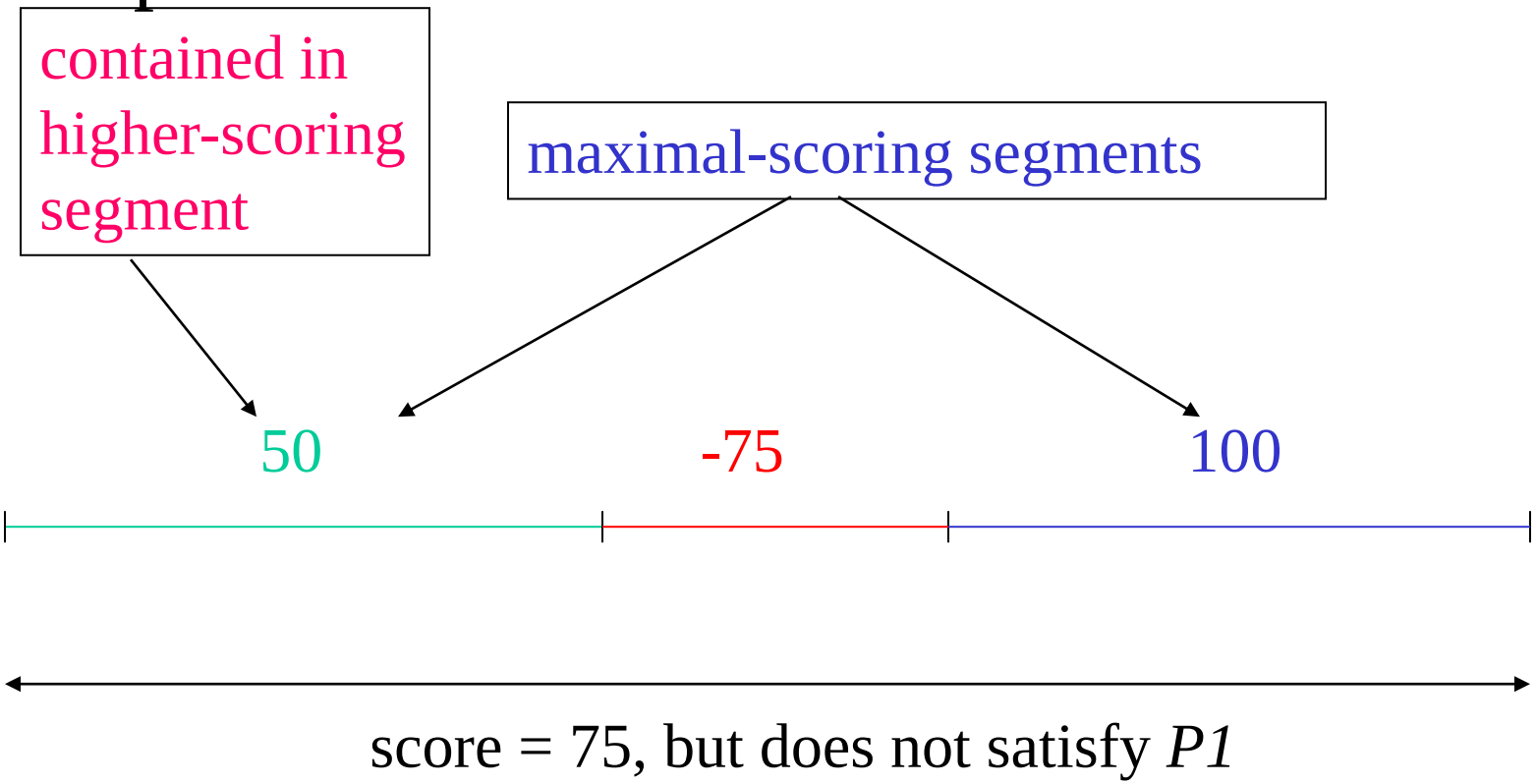
Graph for finding short tandem repeats:



ACACACACACACACAC
ACACACACACACACAC

Finding multiple high-scoring segments in WLLs

- A (*locally*-)*maximal*(*-scoring*) *segment* I is one such that
  - *P1:* no subsegment of I has a higher score than I
  - *P2:* no segment properly containing I satisfies *P1*
- Example:



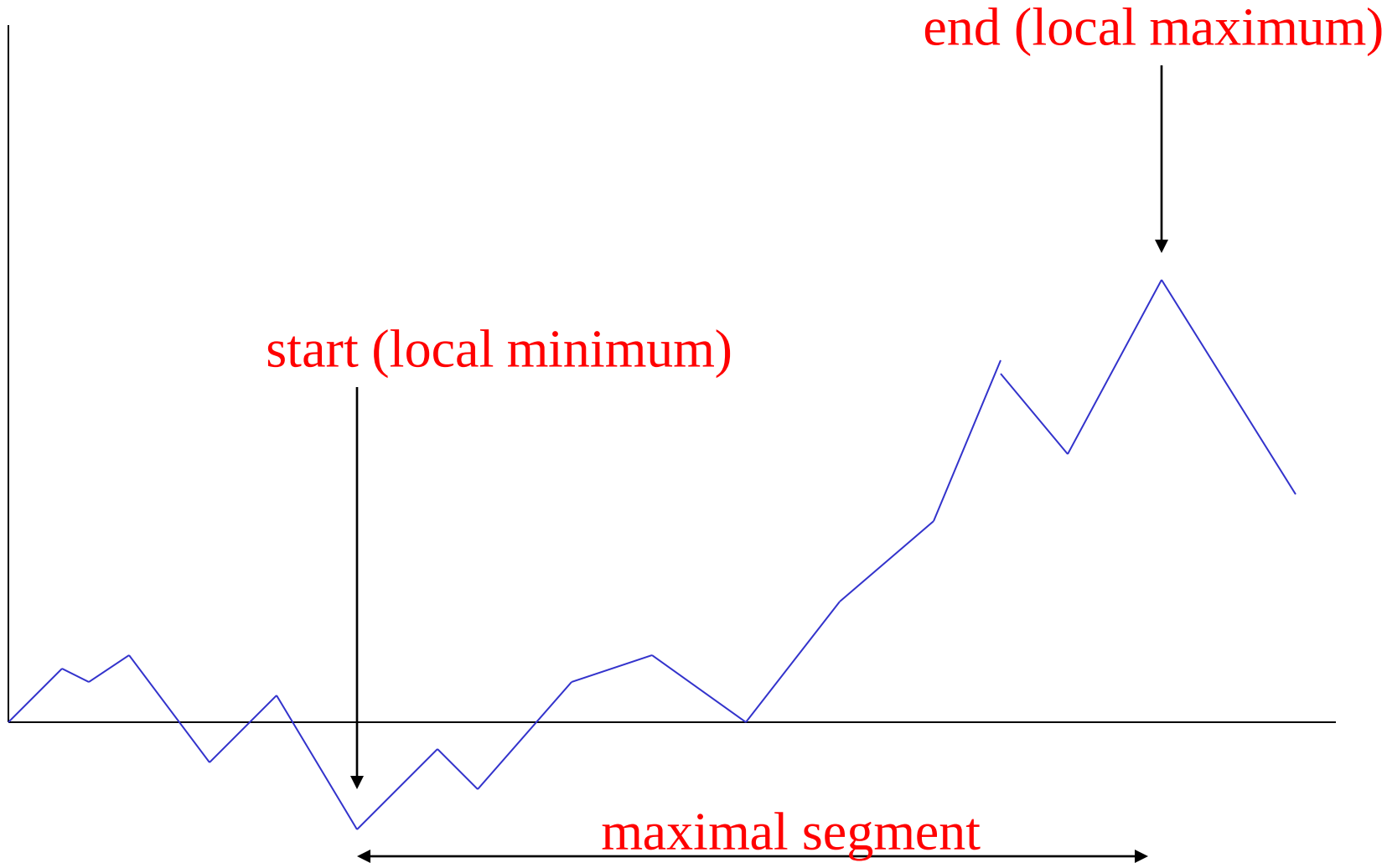
- ***Highest weight path*** via dynamic programming (no explicit graph):

  in (pseudo-)pseudocode:

```
cumul = max = 0;  start = 1;
for (i = 1; i ≤ N; i++)  {
    cumul += s[i];
    if (cumul ≤ 0)
        {cumul = 0;  start = i + 1;}    /* NOTE RESET TO ZERO */
    else if (cumul ≥ max)
        {max = cumul;  best_end = i;  best_start = start;}
}
if (max ≥ S) print best_start, best_end, max
```

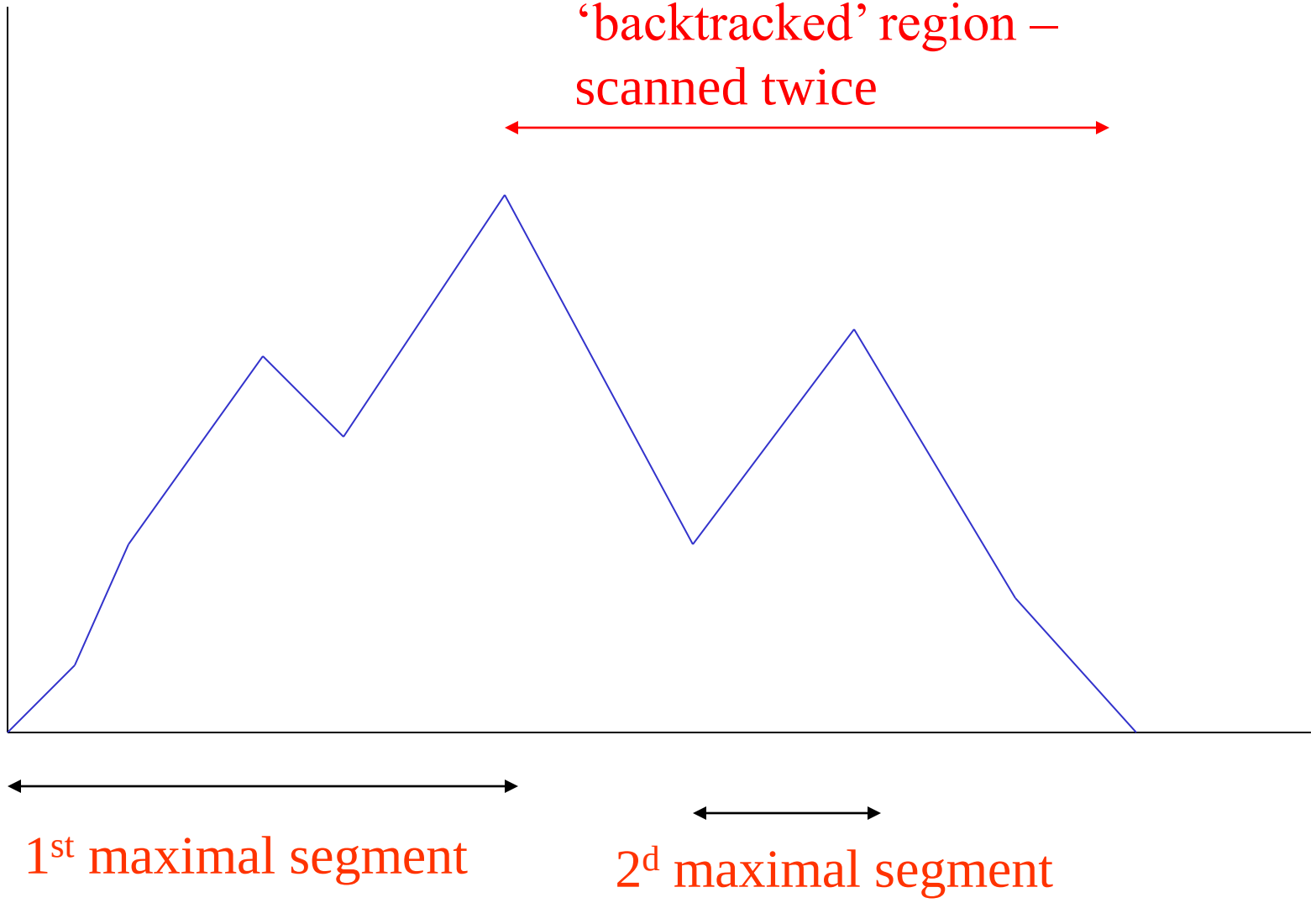
- Correspondence to (implicit) WLL
  - i labels *edges*
  - cumul = w(v) (where v is vertex at end of edge i)
  - max = best w(v) so far
  - best_end = i corresponding to edge ending at best w(v) so far
  - start = edge following B(v)

Maximal segments – from cumulative score plot (*without* 0 resets)



- Can find *all* maximal segs of score ≥ S using following practical (but ***non-optimal***) algorithm:

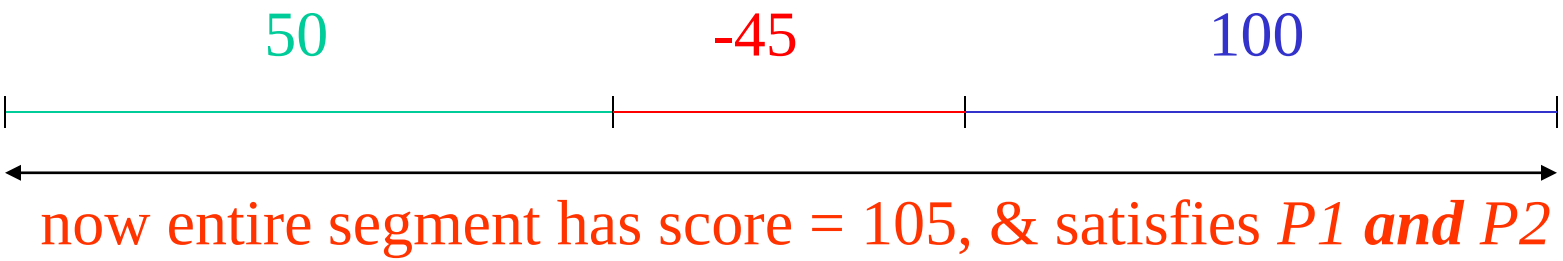
```
cumul = max = 0;  start = 1;
for (i = 1; i ≤ N; i++) {
    cumul += s[i];
    if (cumul ≥ max)
        {max = cumul; end = i;}
    if (cumul ≤ 0 or i == N) {
        if (max ≥ S)
            {print start, end, max;   i = end; }  /* N.B. MUST BACKTRACK! */
        max = cumul = 0;  start = end = i + 1;
    }
}
```


‘backtracked’ region –
scanned twice
1st maximal segment
2d maximal segment

- In worst case this is $O(N^2)$ (because of backtracking),
  - but in practice usually $O(N)$ because a given base is usually traversed only a few times
- Ruzzo-Tompa algorithm ***guarantees*** $O(N)$
  - Basic idea:
    - keep list of *potential* high-scoring segments
      - modify as new local maxima/minima encountered
    - report them when confirmed (at end of a region)

- An undesirable aspect of maximal segments as defined:
  - single maximal seg may contain *two* (or more) high-scoring regions, separated by significant negative-scoring regions
  - i.e. two possibly biologically distinct target occurrences get merged into one maximal segment

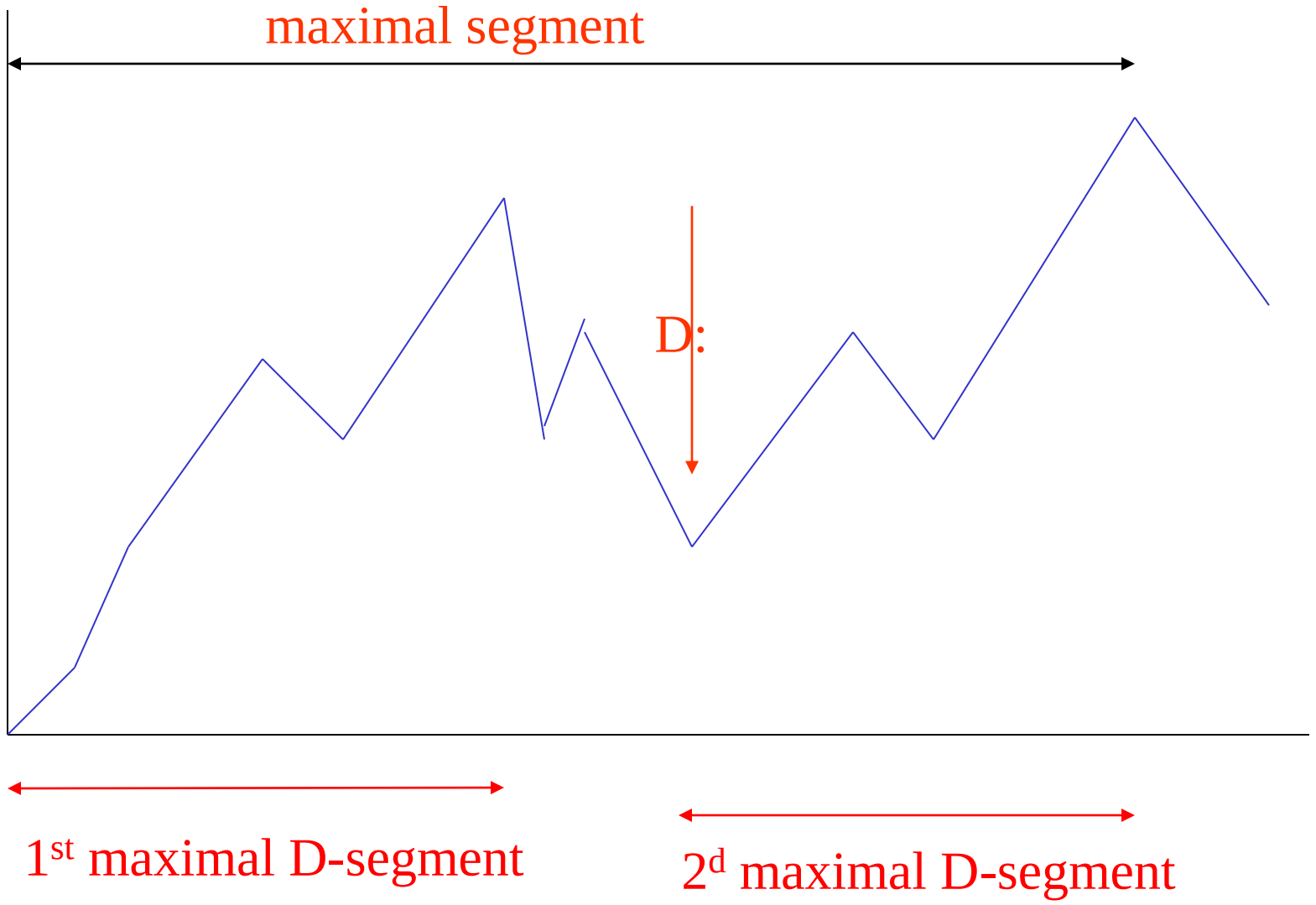
- Example:


50
-45
100
now entire segment has score = 105, & satisfies *P1* ***and*** *P2*

A better problem!

- to avoid this, have max allowed ‘dropoff” D < 0
- *D-segment* is segment without any subsegments of score < D
- *maximal D-segment* is D-segment I such that
  - *P1:* no subsegment of I has higher score than I
  - *P2:* no D-segment properly containing I satisfies *P1*
- Problem: given S ($\geq -D$), find all maximal D-segs of score $\geq S$
  - (algorithm fails if $S < -D$)

Maximal D-segments



- *O*(*N*) algorithm to find all maximal D-segs:

```
cumul = max = 0; start = 1;
for (i = 1; i ≤ N; i++) {
    cumul += s[i];
    if (cumul ≥ max)
        {max = cumul; end = i;}
    if (cumul ≤ 0 or cumul ≤ max + D or i == N) {
        if (max ≥ S)
            {print start, end, max; }
        max = cumul = 0; start = end = i + 1; /* NO BACKTRACKING
            NEEDED! */
    }
}
```

- So *more biologically relevant* problem is also *computationally simpler*!
- what are appropriate S and D?
  - mainly an empirical question (based on known examples); altho
    - interpretation via 2-state HMM can be useful
    - Karlin-Altschul theory tells when they are ‘statistically significant’

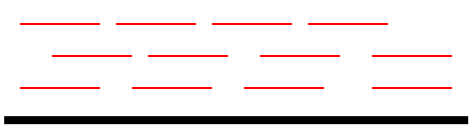
D-Segments

- Powerful tool for analyzing ‘linear’ data
  - Single sequences (incl. motifs, numerical data)
  - Fixed alignment
- Strengths:
  - Very simple to program
  - Very fast, even for mammalian genomes
- Main limitation:
  - Only allows two types of segments (‘target’ and ‘background’)
    - Essentially a generalization of 2-state HMMs
    - multi-state HMMs are more flexible

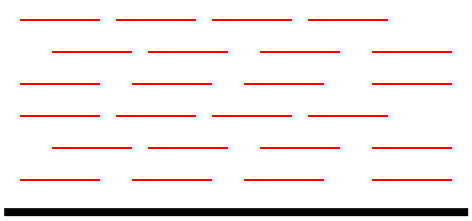
CNVs & Read Depth

- CNV = ‘copy number variant’– e.g. region that is single copy in reference sequence but duplicated in sample
- One way to detect: map reads from sample onto reference, look for regions of atypical coverage depth

‘*Single-copy*’ in sample and reference



multi-copy in sample



HW 6: finding CNVs using D-segments

- ***data***: next-gen read alignments to genome
- observed symbols: *counts* of **# *read starts*** at each position (0, 1, 2, ≥ 3)
  - *frequencies* from ***Poisson dist'n*** with appropriate mean
- target regions: ***heterozygous duplications***
  - One chrom = reference allele, other is dup
  - Poisson mean = **1.5** X background mean