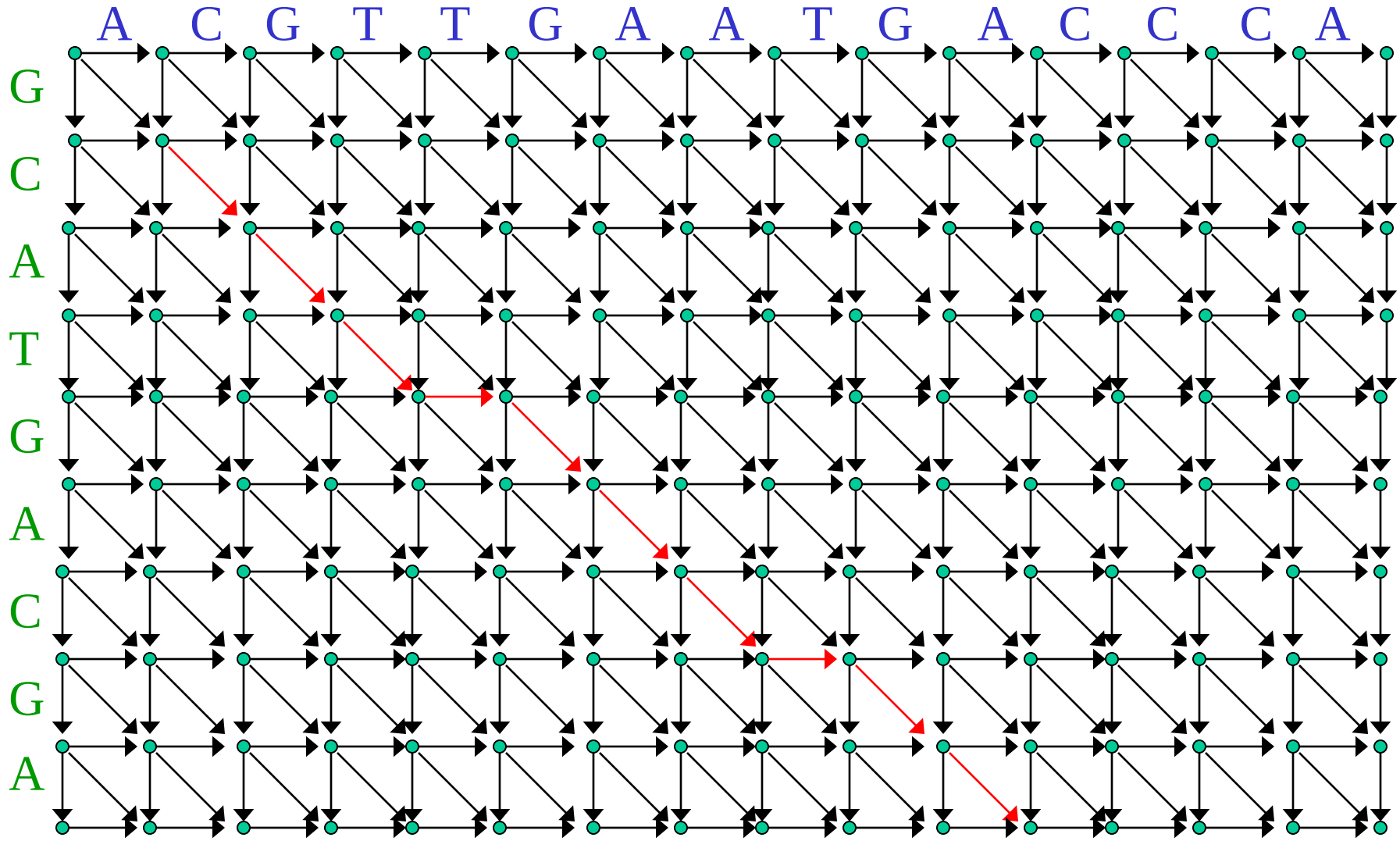
Lecture 11

- Indel penalties
- Word nucleation algorithms
  - BLAST
- Genome alignment



Above path corresponds to following alignment (w/ lower case letters considered unaligned):

```
aCGTTGAATGAccca
gCAT-GAC-GA
```

Gap Penalties

```
TNAVAHVD-----DMPNAL
YEAAIQLQVTGVVVTDATL
```

- Usual scoring scheme assigns same penalty g to each gap edge, so
  - weights on extended gaps of size s are *linear* in s, i.e.
  - total gap penalty $gap(s) = s \times g$.
  - e.g. in above example, if each g = -6, total penalty on gap would be

$$gap(5) = 5 \times -6 = -30$$

- Would like more flexible gap penalties:
- In proteins, insertions & deletions are rare;
  - but when occur, often consist of several residues, because
    - they are in regions (loops) tolerant of length changes
  - at DNA level, indels in protein coding sequence usually a multiple of 3 nucleotides
    - otherwise, would change reading frame
- In noncoding DNA sequence,
  - the most common indel size is 1
  - *but* larger indels occur much more frequently than multiple independent single-base indels

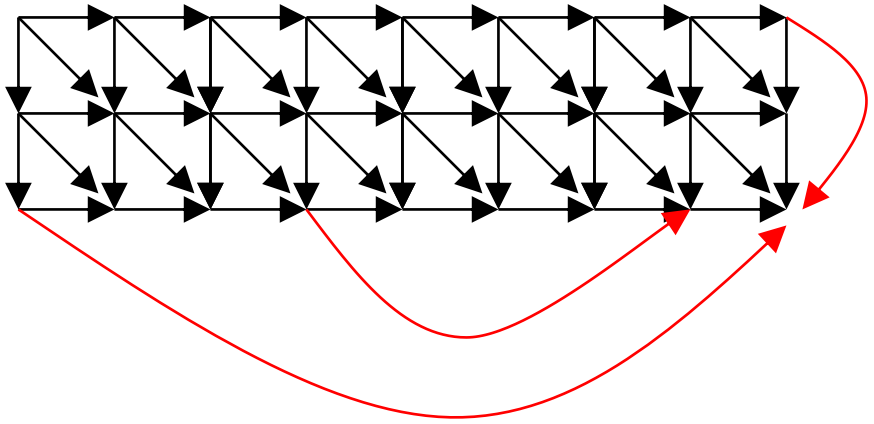
- Can allow arbitrary *convex* gap penalties
  - $gap(s+t) \geq gap(s) + gap(t)$, where s and t are (integer) gap sizes

by extending edit graph:

- add edges corresponding to *arbitrary length* gaps from each vertex to each horizontally or vertically downstream vertex
- (convexity condition prevents favoring two adjacent short gaps over a single long gap).

Time complexity now $O(MN(M+N))$

- often unacceptable for moderate M, N.
- Also: how to choose appropriate weights? (need data to estimate!)



Affine Gap Penalties

- *Affine* gap penalties:
  - less general than arbitrary convex penalties, but
  - more general than linear penalties.
- Two parameters:
  - *gap opening* penalty g_o
  - *gap extension* penalty g_e
- *gap*(*n*) (penalty for size *n* gap) is then

$$g_o + n\, g_e \;\; = g_i + (n - 1)\, g_e$$

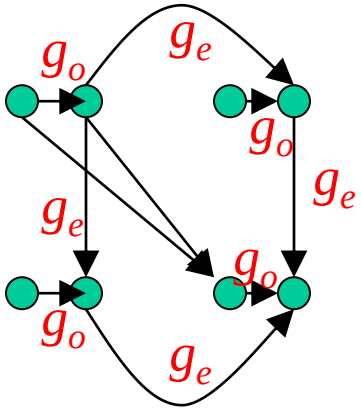
where the gap *initiating* penalty $g_i = g_o + g_e$

- Example: for BLOSUM62, good penalties are
  - g_i = -12,
  - g_e = -2

  These perform *much* better than linear penalty
  - (e.g. g = -6)
- N.B. Durbin *et al.* reverse g_i and g_o
  - g_i is called the 'gap opening' penalty
- Can obtain affine penalties using extension of edit graph, retaining complexity $O(MN)$:

Edit Graph for Affine Gap Penalties

Double # vertices, creating left-right pair in place of each original vertex. Each cell looks like this:



each left vertex has out-degree and in-degree = 2

each right vertex has out-degree and in-degree = 3

• gap-opening edges from left vertex to right vertex of each pair : weight g_o

• gap extension edges going horizontally or vertically between right vertices : weight g_e

• diagonal edges originate from either left or right vertex, but always go to a left vertex.

- Paths in the augmented graph still correspond to alignments
  - can ∃ more than one path for same alignment
  - but highest scoring paths still give best alignments
- Score assigned to size n gap is $g_o + n\, g_e$
  - *i.e.* affine penalty
- ‘Smith-Waterman-Gotoh algorithm’

Finding values for gap penalties

- Direct definition as LLR seems problematic
  - what are ‘random alignments’?
- *Empirical approach*: Given a score matrix (e.g. BLOSUM62), for various (g_o , g_e) choices
  - Align real sequences to known homologues & simulated sequences
  - Measure score discrimination (E-values of homologue alignments)
  - Find (g_o , g_e) giving best discrimination

Gap attraction

- When there are multiple close indels, finding the correct alignment can be problematic:

- If *true* alignment is

```
...acagaatcaggGtcc-gtta...
...acagaatcagg-tccCgtta...
```

reported (maximum-scoring) alignment will be

```
...acagaatcaggGtccgtta...
...acagaatcaggtccgtta...
```

(2 mismatches cost less than 2 indels)

- Similarly, if *true* alignment is

```
...acagaatcaggGtccCgtta...
...acagaatcagg-tcc-gtta...
```

reported alignment will be

```
...acagaatcaggGtcCcgtta...
...acagaatcagg--tccgtta...
```

(size-2 indel + mismatch cost less than 2 size-1 indels)

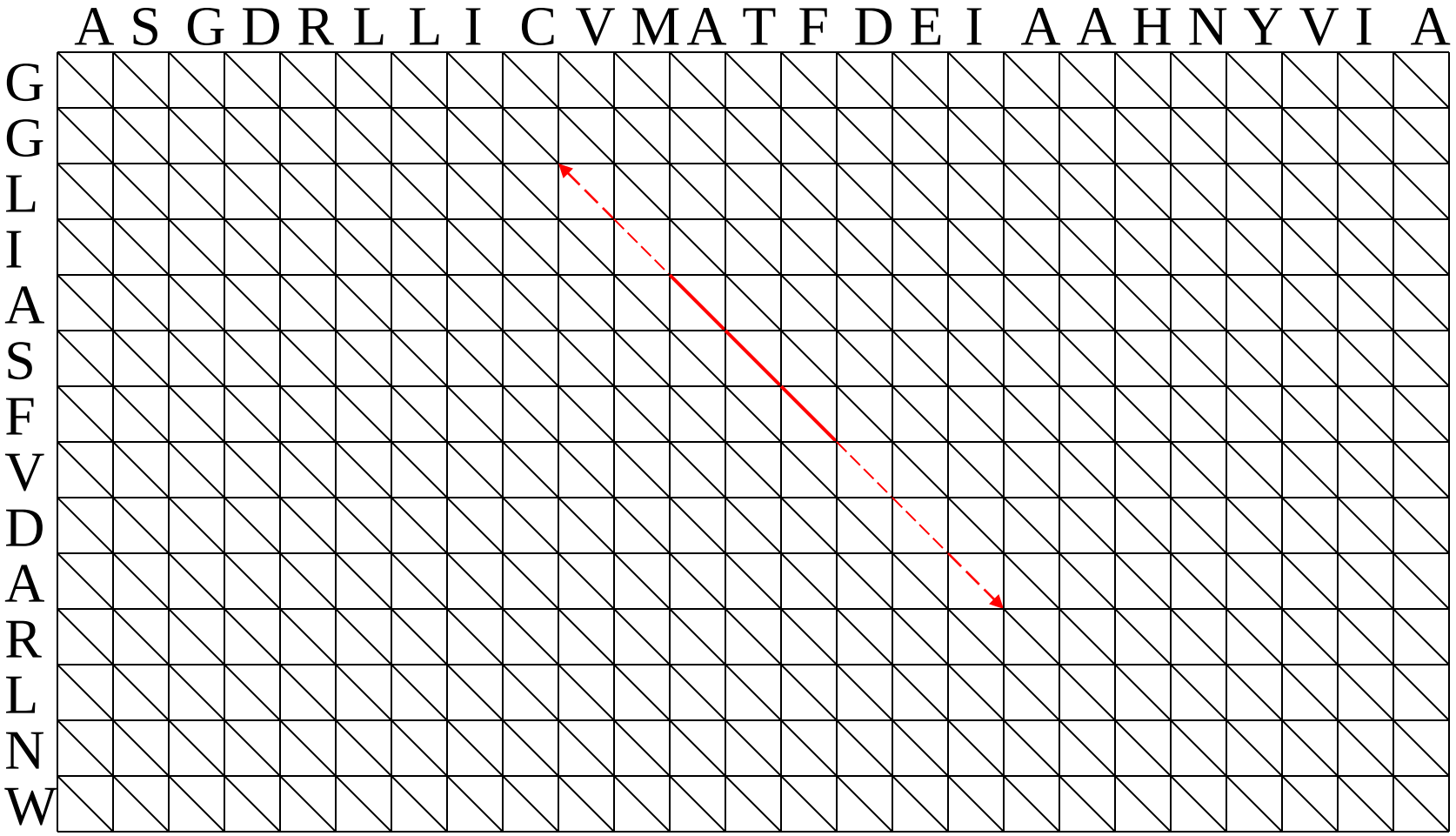
- This is an issue even for highly similar genomes!
  - But worse with increasing divergence
- Ideally, report alignments with local indications of uncertainty
  - or at least, several alignments with varying alignment penalties

  but this is almost never done
- Problem is ameliorated with multiple sequence alignments

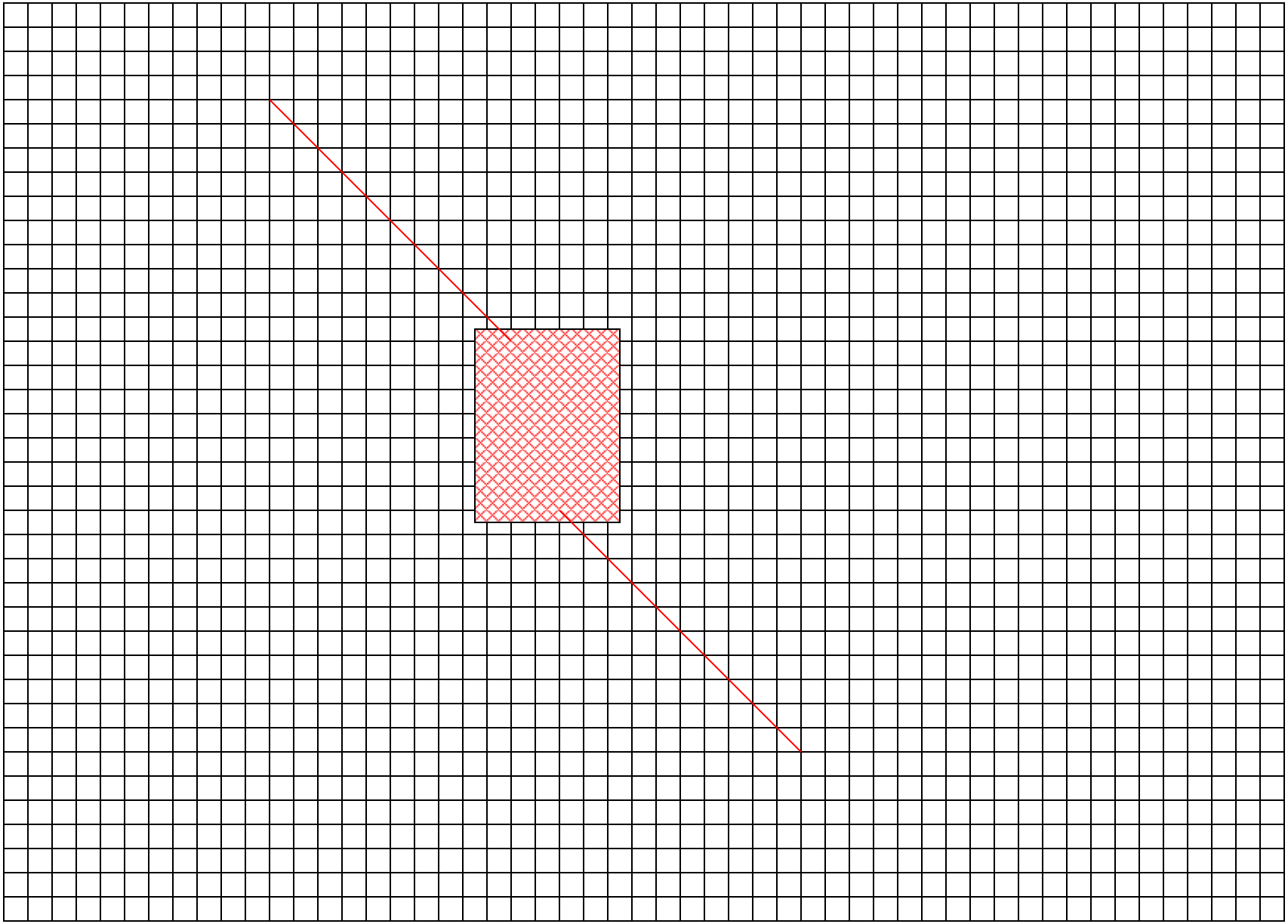
Word Nucleation Algorithms

- Idea: find short (perfect or imperfect) word matches to 'nucleate' graph search
  - Each such match defines short *diagonal* path
  - Only search part of graph 'surrounding' this path
- BLAST: allow *imperfect* short (e.g. length 3) matches.
  - "*Neighbors*": set of 3-residue sequences having ≥ min score T against some 3-residue sequence of query
  - Scan database seqs until hit word in neighbor list
  - then do ungapped extension (along diagonal defined by word match)
    - 'significant' matches are those with scores ≥ a threshold S
    - Ungapped matches are effective for detecting related proteins:
      - **true protein alignments usually include substantial gap-free regions.**

BLAST: Word Nucleating Alignment



– If find ≥ 2 significant ungapped matches in same seq, expand search to connecting region of matrix, allowing gaps:



Other Word Nucleation Programs

- FASTA:
  - look for clusters of short exact matches, on nearby diagonals;
  - when found, extend to gapped alignment
- *cross_match*:
  - do full search of *bands* around exact matches
- These all still time complexity $O(MN)$
  - because # word matches proportional to MN

but with much smaller constant.

- In database searches, most seqs unrelated to query.

  suggests following strategy:
- use fast word-nucleation algorithm
  - e.g. just looking for gap-free matches

  to identify sequences ‘of interest’
  - having scores above a (low) threshold

  then use full Smith-Waterman on these
  - can get sensitivity nearly as good as full Smith-Waterman search.

Genome alignment

- Challenges:
  - Size
  - Repeated sequence
    - Duplications
    - Transposable elements
    - Processed pseudogenes
  - Other segmental changes
    - Deletions
    - Inversions, translocations
  - Mutation rate variation
- Segmental changes don’t conform to edit graph framework!

Strategy

- Find (many!) word-nucleated local alignments
- Word size *w*: sensitivity vs specificity
  - Example: human (~3 Gb) vs mouse (~2.5 Gb)
    - ~70% identity in homologous regions
    - For each human word, expect $5 \times 10^9 / 4^w$ chance occurrences in mouse (+ rev complement)
    - Total matches: $15 \times 10^{18} / 4^w$
      - Want *w large enough* for this to be manageable
    - Prob that the *homologous* word matches: $.7^w$
      - once every $(1 / .7)^w = 1.43^w$ bp
      - Want *w small enough* to ensure ≥ 1 match within homologous regions
    - $w = 15$: ~15×10^9 matches; 1 per 214 homologous bp

- Avoid high-frequency words
- Avoid nucleating in known repeats & duplications
  - But extend into them!
- Use appropriate score matrix & gap penalties!
  - Otherwise, get junk alignments or portions thereof

- Finally, identify ***chains*** of ***compatible*** local alignments
  - Ideally, catalogue the segmental changes that have occurred (duplications, transposable element insertions etc)