Sequence Alignment

Outline

1. Global Alignment

2. Scoring Matrices

3. Local Alignment

4. Alignment with Affine Gap Penalties

Section 1: Global Alignment

From LCS to Alignment: Change the Scoring

- **Recall**: The Longest Common Subsequence (LCS) problem allows only insertions and deletions (no mismatches).
- In the LCS Problem, we scored 1 for matches and 0 for indels, so our alignment score was simply equal to the total number of matches.
- Let's consider penalizing mismatches and indels instead.

From LCS to Alignment: Change the Scoring

- Simplest *scoring schema*: For some positive numbers μ and σ:
  - **Match Premium**: +1
  - **Mismatch Penalty**: $-\mu$
  - **Indel Penalty**: $-\sigma$

- Under these assumptions, the alignment score becomes as follows:

$$\text{Score} = \#matches - \mu(\#mismatches) - \sigma(\#indels)$$

- Our specific choice of μ and σ depends on how we wish to penalize mismatches and indels.

The Global Alignment Problem

- Input : Strings **v** and **w** and a scoring schema

- Output : An alignment with maximum score

- We can use dynamic programming to solve the Global Alignment Problem:

$$s_{i,j} = \max \begin{cases} s_{i-1,\,j-1} + 1 & \text{if } v_i = w_j \\ s_{i-1,\,j-1} - \mu & \text{if } v_i \neq w_j \\ s_{i-1,\,j} - \sigma & \\ s_{i,\,j-1} - \sigma & \end{cases}$$

μ : mismatch penalty
σ : indel penalty

Section 2: Scoring Matrices

Scoring Matrices

- To further generalize the scoring of alignments, consider a (4+1) x (4+1) **scoring matrix** δ.
  - The purpose of the scoring matrix is to score one nucleotide against another, e.g. A matched to G may be "worse" than C matched to T.
  - The addition of 1 is to include the score for comparison of a gap character "-".
- This will simplify the algorithm to the dynamic formula at right:

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{cases}$$

Note: For amino acid sequence comparison, we need a (20 + 1) x (20 + 1) matrix.

Scoring Matrices: Example

- Say we want to align AGTCA and CGTTGG with the following scoring matrix:

	A	**G**	**T**	**C**	**—**
A	1	-0.8	-0.2	-2.3	-0.6
G	-0.8	1	-1.1	-0.7	-1.5
T	-0.2	-1.1	1	-0.5	-0.9
C	-2.3	-0.7	-0.5	1	-1
—	-0.6	-1.5	-0.9	-1	n/a

Sample Alignment:

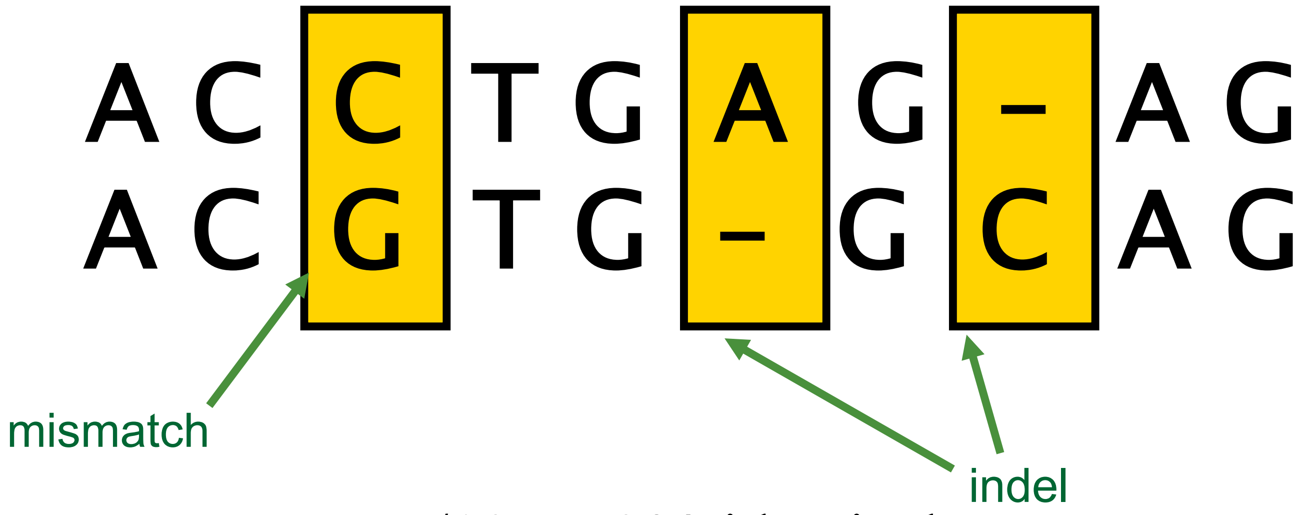
```
A GTC A
 CGTTGG
```

Score: –0.6 – 1 + 1 + 1 – 0.5 – 1.5 – 0.8 = –2.4

Percent Sequence Identity

- **Percent Sequence Identity:** The extent to which two nucleotide or amino acid sequences are invariant.

- **Example**:



7/10 = **70%** identical

How Do We Make a Scoring Matrix?

- Scoring matrices are created based on biological evidence.
- Alignments can be thought of as two sequences that differ due to mutations.
- Some of these mutations have little effect on the protein’s function, therefore some penalties, $\delta(v_i , w_j)$, will be less harsh than others.
- This explains why we would want to have a scoring matrix to begin with.

Scoring Matrix: Positive Mismatches

- Notice that although R and K are different amino acids, they have a positive mismatch score.

- Why? They are both positively charged amino acids → this mismatch will not greatly change the function of the protein.

	A	R	N	K
A	5	-2	-1	-1
R	-2	7	-1	3
N	-1	-1	7	0
K	-1	3	0	6

AKRANR

KAAANK

-1 + (-1) + (-2) + 5 + 7 + 3 = 11

Scoring Matrix: Positive Mismatches

- Notice that although R and K are different amino acids, they have a positive mismatch score.
- Why? They are both positively charged amino acids → this mismatch will not greatly change the function of the protein.

	A	R	N	K
A	5	-2	-1	-1
R	-2	7	-1	3
N	-1	-1	7	0
K	-1	3	0	6

AKRANR

KAAANK

-1 + (-1) + (-2) + 5 + 7 + 3 = 11

Scoring Matrix: Positive Mismatches

- Notice that although R and K are different amino acids, they have a positive mismatch score.

- Why? They are both positively charged amino acids → this mismatch will not greatly change the function of the protein.

	A	R	N	K
A	5	-2	-1	-1
R	-2	7	-1	3
N	-1	-1	7	0
K	-1	3	0	6

```
AKRANR
KAAANK
```

-1 + (-1) + (-2) + 5 + 7 + 3 = 11

Mismatches with Low Penalties

- Amino acid changes that tend to preserve the physicochemical properties of the original residue:
  - Polar to Polar
  - Aspartate to Glutamate
  - Nonpolar to Nonpolar
  - Alanine to Valine
  - Similarly-behaving residues
  - Leucine to Isoleucine

Scoring Matrices: Amino Acid vs. DNA

- Two commonly used amino acid substitution matrices:
  1. PAM
  2. BLOSUM

- DNA substitution matrices:
  - DNA is less conserved than protein sequences
  - It is therefore less effective to compare coding regions at the nucleotide level
  - Furthermore, the particular scoring matrix is less important.

PAM

- **PAM**: Stands for **P**oint **A**ccepted **M**utation
- 1 PAM = PAM_1 = 1% average change of all amino acid positions.
- **Note**: This *doesn't* mean that after 100 PAMs of evolution, every residue will have changed:
  - Some residues may have mutated several times.
  - Some residues may have returned to their original state.
  - Some residues may not changed at all.

PAM_X

- $PAM_x = PAM_1^x$ (x iterations of PAM_1)
  - **Example**: $PAM_{250} = PAM_1^{250}$

- PAM_{250} is a widely used scoring matrix:

	Ala A	Arg R	Asn N	Asp D	Cys C	Gln Q	Glu E	Gly G	His H	Ile I	Leu L	Lys K	...
Ala A	13	6	9	9	5	8	9	12	6	8	6	7	...
Arg R	3	17	4	3	2	5	3	2	6	3	2	9	
Asn N	4	4	6	7	2	5	6	4	6	3	2	5	
Asp D	5	4	8	11	1	7	10	5	6	3	2	5	
Cys C	2	1	1	1	52	1	1	2	2	2	1	1	
Gln Q	3	5	5	6	1	10	7	3	7	2	3	5	
...													
Trp W	0	2	0	0	0	0	0	0	1	0	1	0	
Tyr Y	1	1	2	1	3	1	1	1	3	2	2	1	
Val V	7	4	4	4	4	4	4	4	5	4	15	10	

BLOSUM

- **BLOSUM**: Stands for **Bl**ocks **Su**bstitution **M**atrix

- Scores are derived from *observations* of the frequencies of substitutions in blocks of local alignments in related proteins.

- BLOSUM62 was created using sequences sharing no more than 62% identity.

- A sample of BLOSUM62 is shown at right.

	C	**S**	**T**	**P**	**…**	**F**	**Y**	**W**
C	9	-1	-1	3	…	-2	-2	-2
S	-1	4	1	-1	…	-2	-2	-3
T	-1	1	4	1	…	-2	-2	-3
P	3	-1	1	7	…	-4	-3	-4
…	…	…	…	…	…	…	…	…
F	-2	-2	-2	-4	…	6	3	1
Y	-2	-2	-2	-3	…	3	7	2
W	-2	-3	-3	-4	…	1	2	11

http://www.uky.edu/Classes/BIO/520/BIO520WWW/blosum62.htm

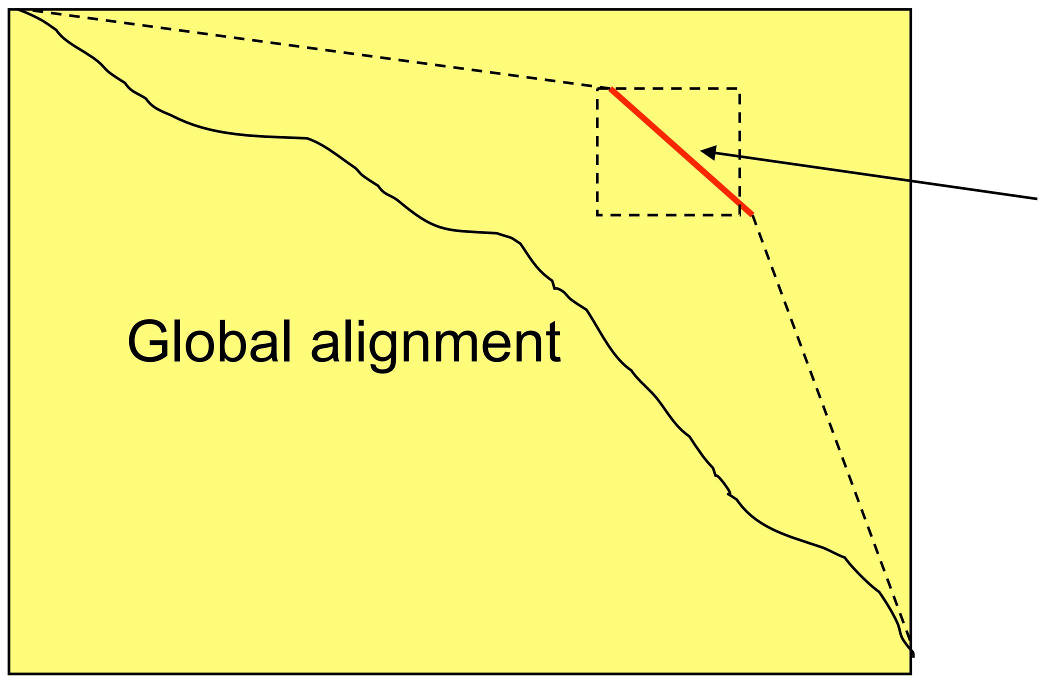
Section 3:
Local Alignment

Local Alignment: Why?

- Two genes in different species may be similar over short conserved regions and dissimilar over remaining regions.

- **Example**: Homeobox genes have a short region called the *homeodomain* that is highly conserved among species.
  - A global alignment would not find the homeodomain because it would try to align the *entire* sequence.
  - Therefore, we search for an alignment which has a positive score *locally*, meaning that an alignment on substrings of the given sequences has a positive score.

Local Alignment: Illustration



Compute a "mini" Global Alignment to get Local Alignment

Local vs. Global Alignment: Example

- Global Alignment:

```
--T—-CC-C-AGT—-TATGT-CAGGGGACACG—A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || | | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG—T-CAGAT--C
```

- Local Alignment—better alignment to find conserved segment:

```
                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

The Local Alignment Problem

- Goal: Find the best local alignment between two strings.
- Input : Strings ***v*** and ***w*** as well as a scoring matrix *δ*
- Output : Alignment of substrings of ***v*** and ***w*** whose alignment score is maximum among all possible alignments of all possible substrings of ***v*** and ***w***.

Local Alignment: How to Solve?

- We have seen that the Global Alignment Problem tries to find the longest path between vertices (0,0) and (*n,m*) in the edit graph.

- The **Local Alignment Problem** tries to find the longest path among paths between *arbitrary vertices* (*i,j*) and (*i'*, *j'*) in the edit graph.

Local Alignment: How to Solve?

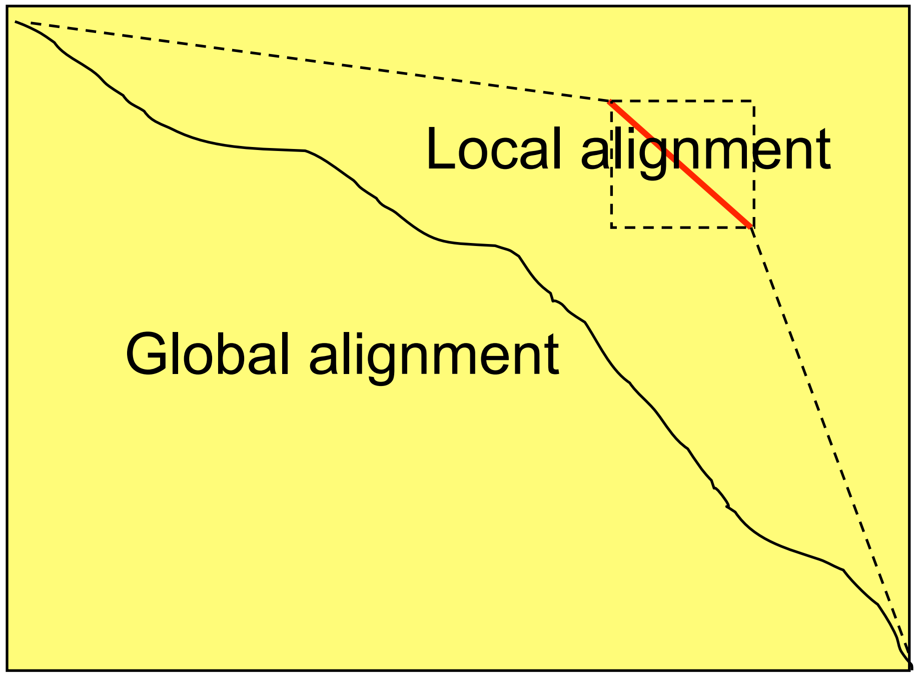
- We have seen that the Global Alignment Problem tries to find the longest path between vertices (0,0) and (*n,m*) in the edit graph.

- The **Local Alignment Problem** tries to find the longest path among paths between *arbitrary vertices* (*i,j*) and (*i'*, *j'*) in the edit graph.

- **Key Point**: In the edit graph with negatively-scored edges, Local Alignment may score higher than Global Alignment.

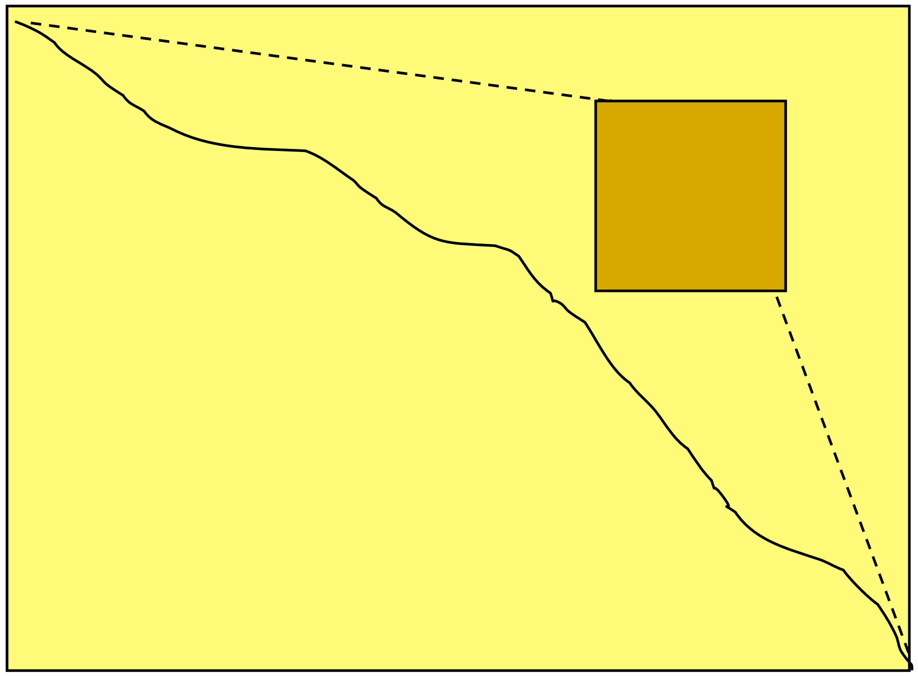
The Problem with This Setup

- In the grid of size *n x n* there are ~n^2 vertices (i,j) that may serve as a source.



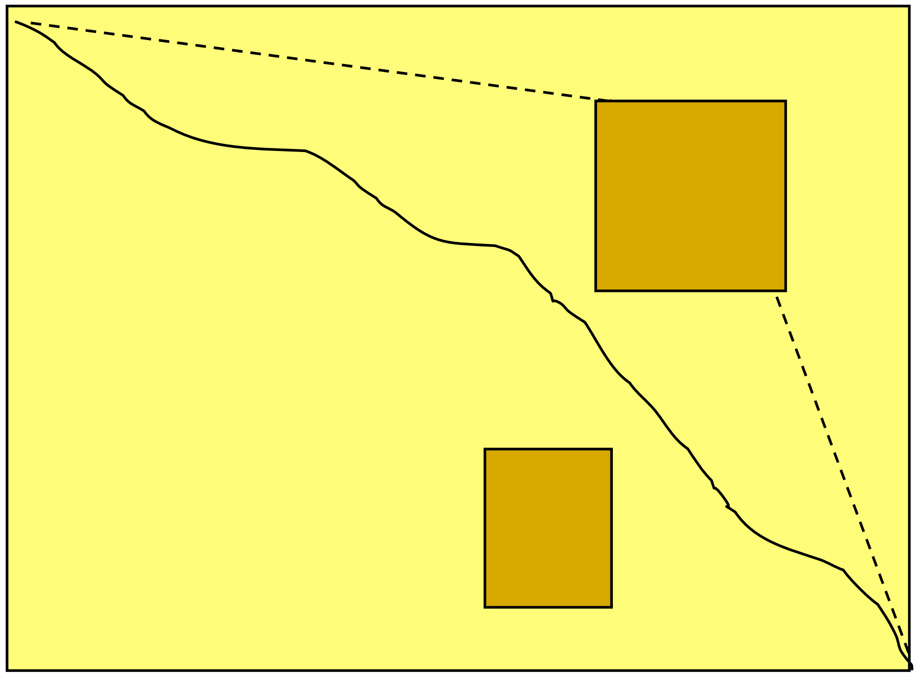
The Problem with This Setup

- In the grid of size *n x n* there are ~n^2 vertices (*i,j*) that may serve as a source.
- For each such vertex computing alignments from (*i,j*) to (*i',j'*) takes $O(n^2)$ time.



The Problem with This Setup

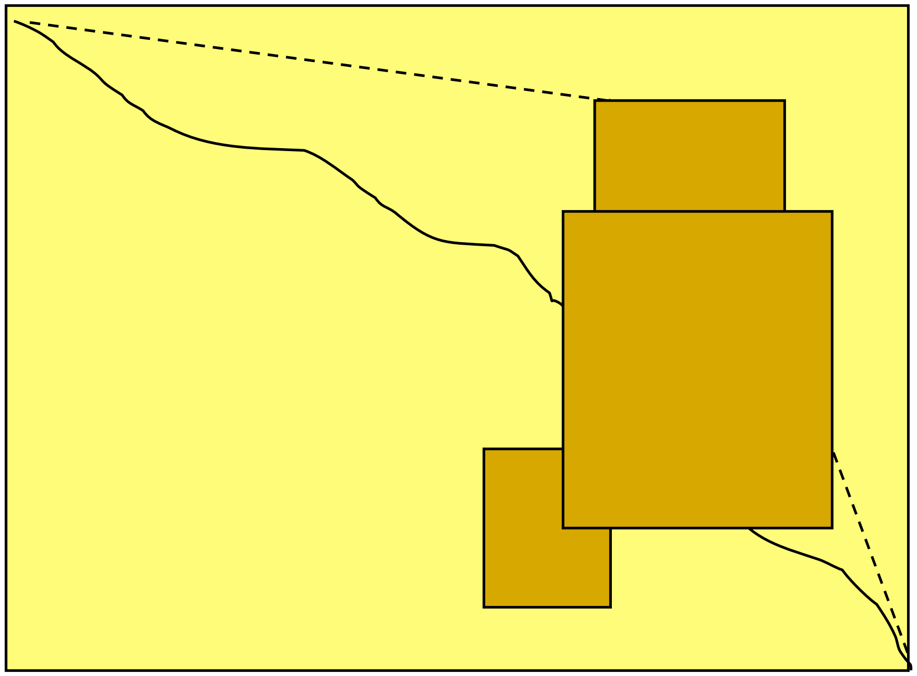
- In the grid of size *n x n* there are ~n^2 vertices (*i,j*) that may serve as a source.
- For each such vertex computing alignments from (*i,j*) to (*i',j'*) takes $O(n^2)$ time.



The Problem with This Setup

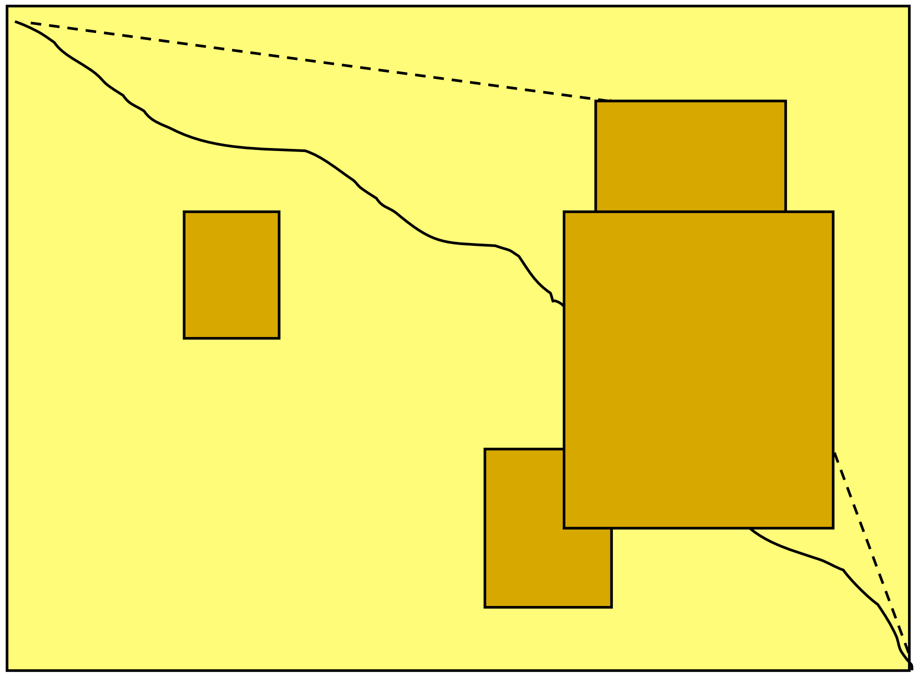
- In the grid of size $n \times n$ there are $\sim n^2$ vertices (i,j) that may serve as a source.

- For each such vertex computing alignments from (i,j) to (i',j') takes $O(n^2)$ time.



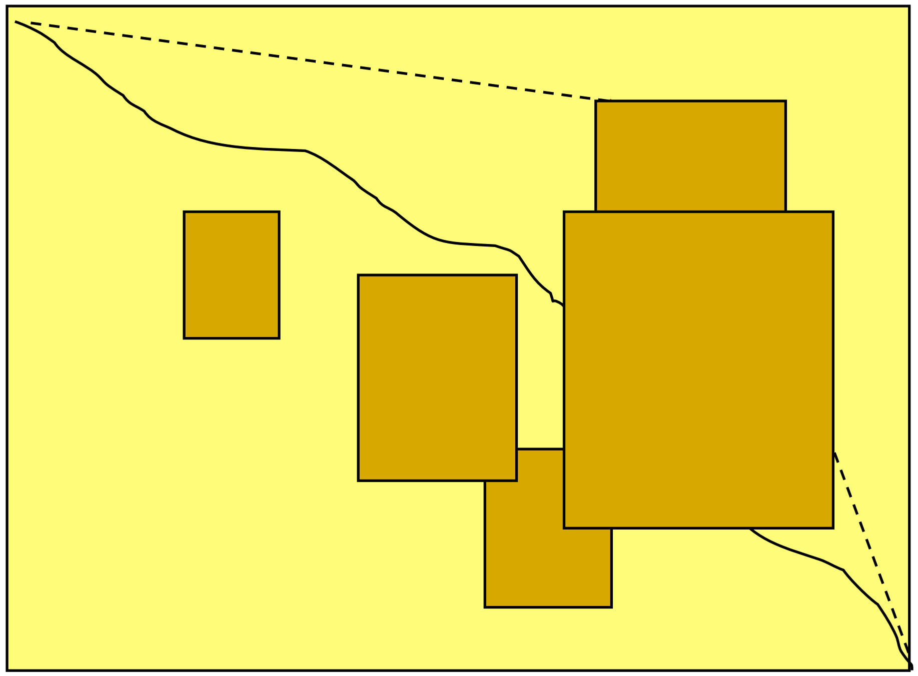
The Problem with This Setup

- In the grid of size *n x n* there are ~n^2 vertices (*i,j*) that may serve as a source.
- For each such vertex computing alignments from (*i,j*) to (*i',j'*) takes $O(n^2)$ time.



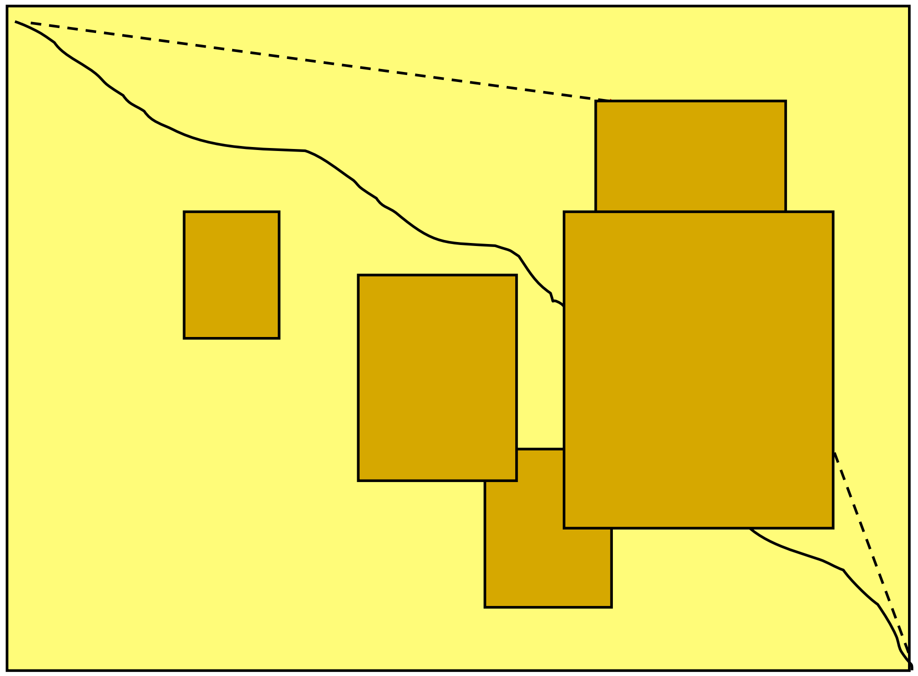
The Problem with This Setup

- In the grid of size *n x n* there are ~n^2 vertices (i,j) that may serve as a source.
- For each such vertex computing alignments from (i,j) to (i',j') takes $O(n^2)$ time.



The Problem with This Setup

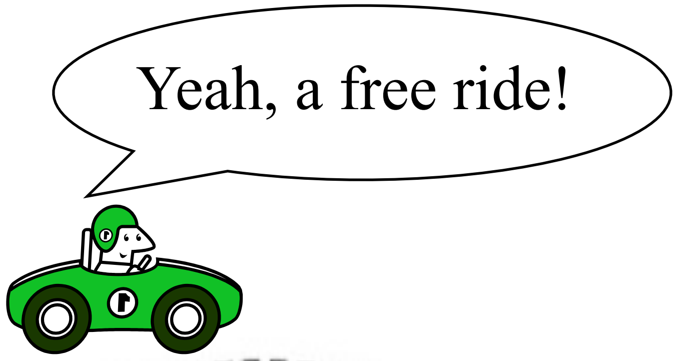
- In the grid of size *n x n* there are ~n^2 vertices (*i,j*) that may serve as a source.
- For each such vertex computing alignments from (*i,j*) to (*i'*,*j'*) takes $O(n^2)$ time.
- This gives an overall runtime of O(n^4), which is a bit too slow…can we do better?

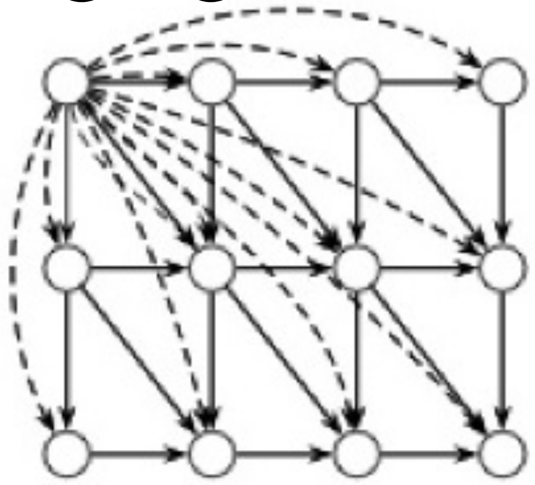


Local Alignment Solution: Free Rides

- The solution actually comes from *adding* vertices to the edit graph.

- The dashed edges represent the "free rides" from (0, 0) to every other node.
    - Each "free ride" is assigned an edge weight of 0.
    - If we start at (0, 0) instead of (i, j) and maximize the longest path to (i', j'), we will obtain the local alignment.





Smith-Waterman Local Alignment Algorithm

- The largest value of $s_{i,j}$ over the whole edit graph is the score of the best local alignment.

- The recurrence:

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, w_j) \\ s_{i,j-1} + \delta(-, w_j) \end{cases}$$

- Notice that the 0 is the only difference between the global alignment recurrence…hence our new algorithm is O(n^2)!

Section 4: Alignment with Affine Gap Penalties

Scoring Indels: Naïve Approach

- In our original scoring schema, we assigned a fixed penalty σ to every indel:
  - $-\sigma$ for 1 indel
  - -2σ for 2 consecutive indels
  - -3σ for 3 consecutive indels
  - Etc.

- **However**…this schema may be too severe a penalty for a series of 100 consecutive indels.

Affine Gap Penalties

- In nature, a series of *k* indels often come as a single event rather than a series of *k* single nucleotide events:

- **Example**:

More Likely

```
ATA__GC
ATATTGC
```

Less Likely

```
ATAG_GC
AT_GTGC
```

Normal scoring would give the same score for both alignments

Accounting for Gaps

- **Gap**: Contiguous sequence of spaces in one of the rows of an alignment.

- **Affine Gap Penalty** for a gap of length *x* is:

$$-(\rho + \sigma x)$$

  - $\rho > 0$ is the **gap opening penalty**: penalty for introducing a gap.
  - $\sigma > 0$ is the **gap extension penalty**: penalty for each indel in a gap.
  - ρ should be large relative to σ, since starting a gap should be penalized more than extending it.

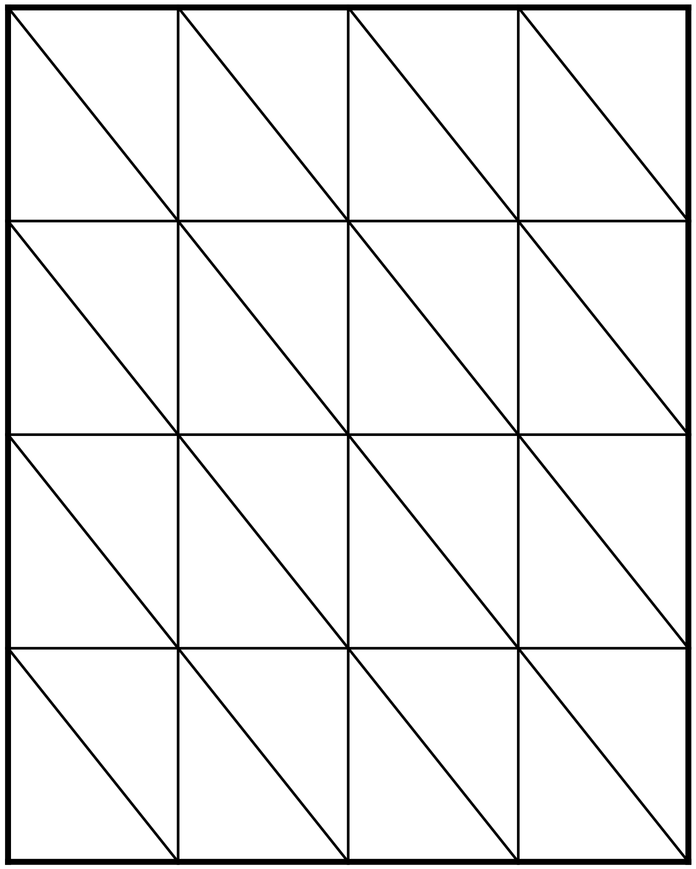
Affine Gap Penalties

- Gap penalties:
  - $-\rho - \sigma$ when there is 1 indel,
  - $-\rho - 2\sigma$ when there are 2 indels,
  - $-\rho - 3\sigma$ when there are 3 indels,
  - $-\rho - x \cdot \sigma$ when there are x indels.

Affine Gap Penalties and the Edit Graph

- To reflect affine gap penalties, we have to add "long" horizontal and vertical edges to the edit graph. Each such edge of length x should have weight

$$-\rho - x \cdot \sigma$$

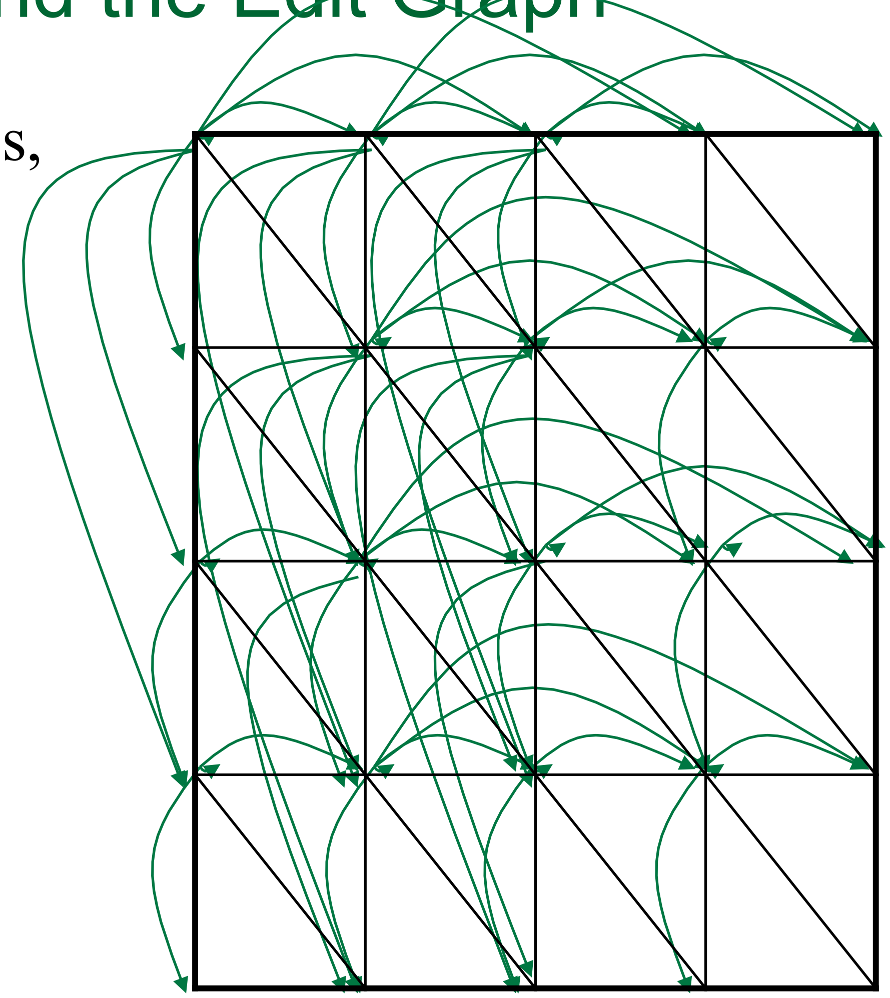


Affine Gap Penalties and the Edit Graph

- To reflect affine gap penalties, we have to add "long" horizontal and vertical edges to the edit graph. Each such edge of length *x* should have weight

$$-\rho - x \cdot \sigma$$

- There are many such edges!



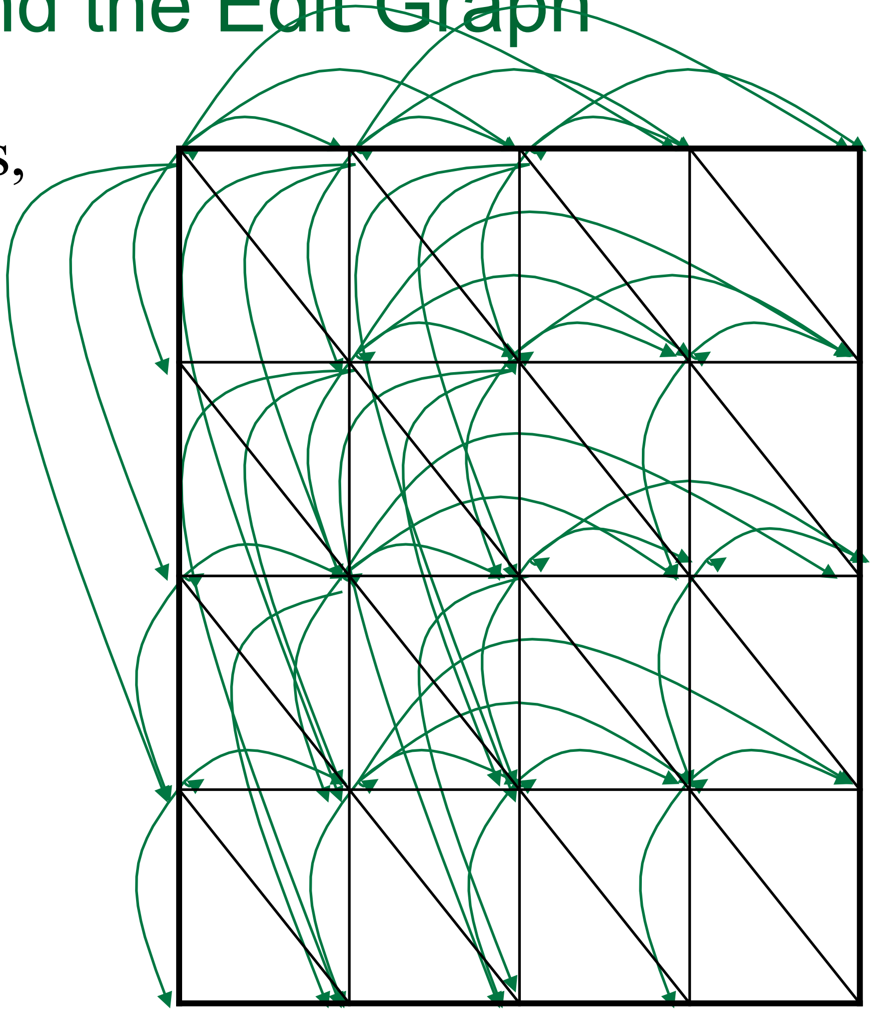
Affine Gap Penalties and the Edit Graph

- To reflect affine gap penalties, we have to add "long" horizontal and vertical edges to the edit graph. Each such edge of length x should have weight

$$-\rho - x \cdot \sigma$$

- There are many such edges!

- Adding them to the graph increases the running time of alignment by a factor of n to $O(n^3)$.



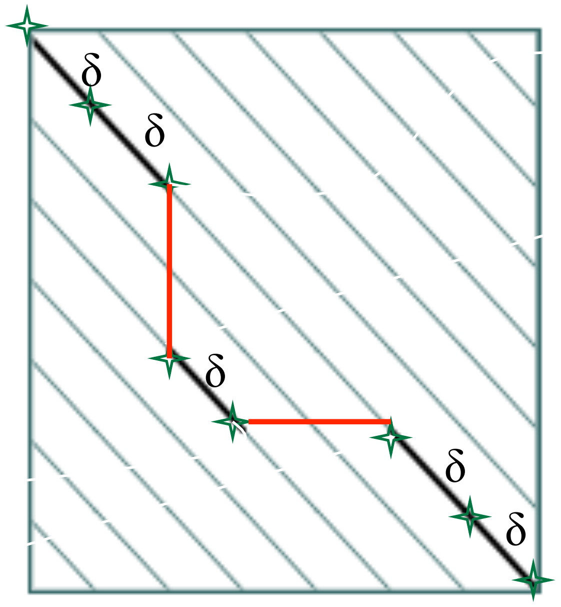
Affine Gap Penalties and 3 Layer Manhattan Grid

- The three recurrences for the scoring algorithm creates a 3-layered graph.
  - The **main level** extends matches and mismatches.
  - The **lower level** creates/extends gaps in sequence *v*.
  - The **upper level** creates/extends gaps in sequence *w*.

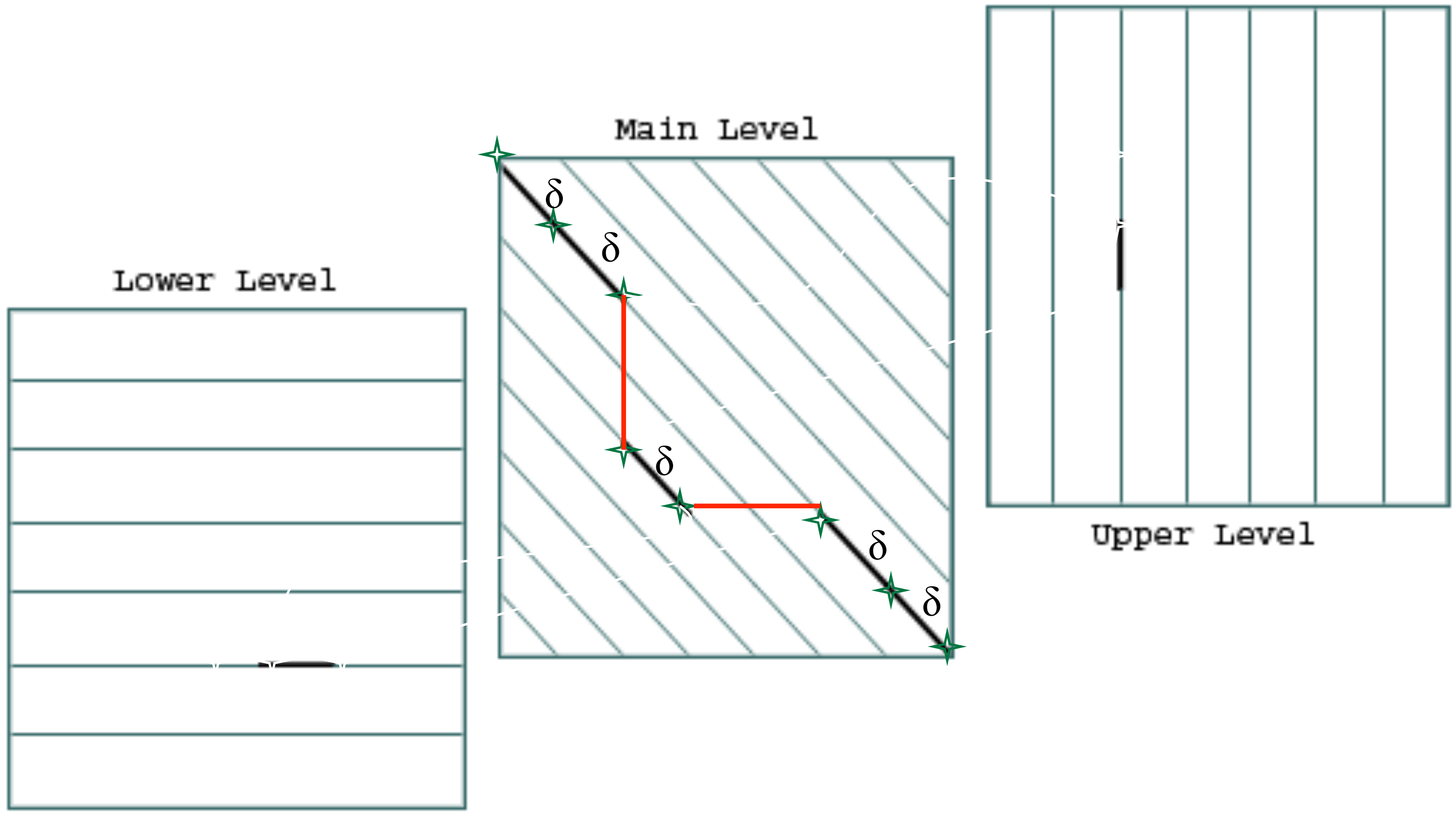
- A jumping penalty is assigned to moving from the main level to either the upper level or the lower level ($-\rho - \sigma$).

- There is a gap extension penalty for each continuation on a level other than the main level ($- \sigma$).

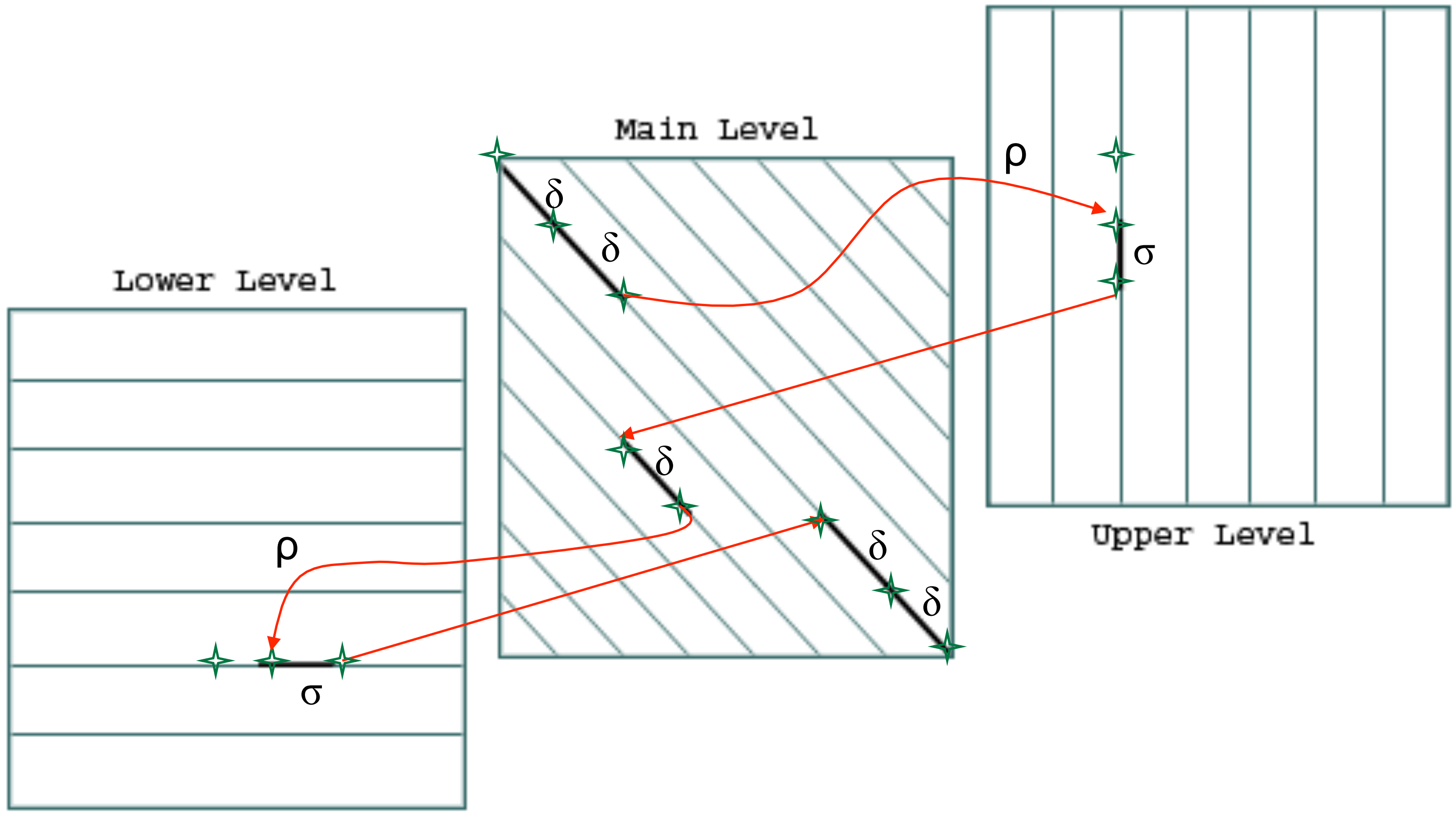
Visualizing Edit Graph: Manhattan in 3 Layers



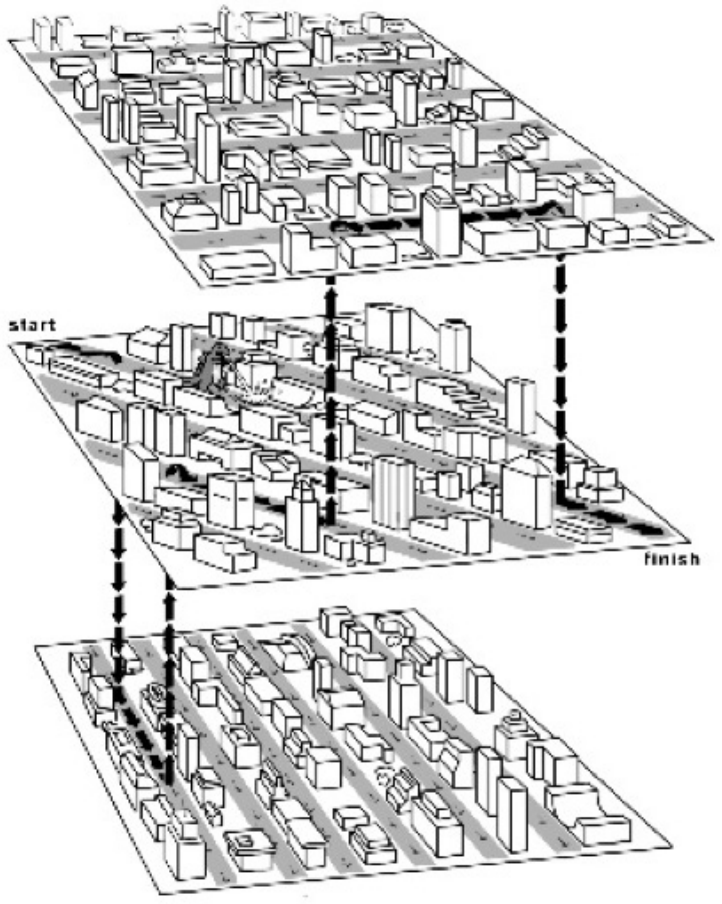
Visualizing Edit Graph: Manhattan in 3 Layers



Visualizing Edit Graph: Manhattan in 3 Layers



The 3-leveled Manhattan Grid



Gaps in w

Matches/Mismatches

Gaps in v

Affine Gap Penalty Recurrences

$$\overset{\downarrow}{s}_{i,j} = \max \begin{cases} \overset{\downarrow}{s}_{i-1,j} - \sigma & \text{Continue gap in } w \text{ (deletion)} \\ s_{i-1,j} - (\rho + \sigma) & \text{Start gap in } w \text{ (deletion): from middle} \end{cases}$$

$$\vec{s}_{i,j} = \max \begin{cases} \vec{s}_{i,j-1} - \sigma & \text{Continue gap in } v \text{ (insertion)} \\ s_{i,j-1} - (\rho + \sigma) & \text{Start gap in } v \text{ (insertion): from middle} \end{cases}$$

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) & \text{Match or mismatch} \\ \overset{\downarrow}{s}_{i,j} & \text{End deletion: from top} \\ \overset{\rightarrow}{\mathrm{s}}_{\mathrm{i,j}} & \text{End insertion: from bottom} \end{cases}$$