Today's Lecture

- Likelihood ratios & Neyman-Pearson lemma
- Sequence alignment and evolution
- Edit graph & alignment algorithms
  - Smith-Waterman algorithm

Likelihood Ratios

- The *likelihood* of a model M given an observation s is

$$L(M \mid s) = P(s \mid M)$$

  This is *not* the *probability* of the model! – (the sum over all models is not 1).

- The *likelihood ratio* (LR) of two models M_a and M_0 is given by

$$LR(M_a, M_0 \mid s) = \frac{L(M_a \mid s)}{L(M_0 \mid s)}$$

  The numerator and denominator may both be very small!

- The *log likelihood ratio* (LLR) is the logarithm of the likelihood ratio.

Simple Hypothesis Testing

- Suppose we wish to decide between two models:
  - M_a (the *alternative hypothesis*), and
  - M_0 (the *null hypothesis*)

  using an observation s from a sample space S. (e.g.
  - s a sequence,
  - M_a a site model
  - M_0 a "background" (non-site) model.
- Strategy:
  - choose a subset $C \subset S$, called the *critical region* for the comparison.
  - If s falls within C, reject M_0 (accept M_a),
  - otherwise accept M_0 (reject M_a).

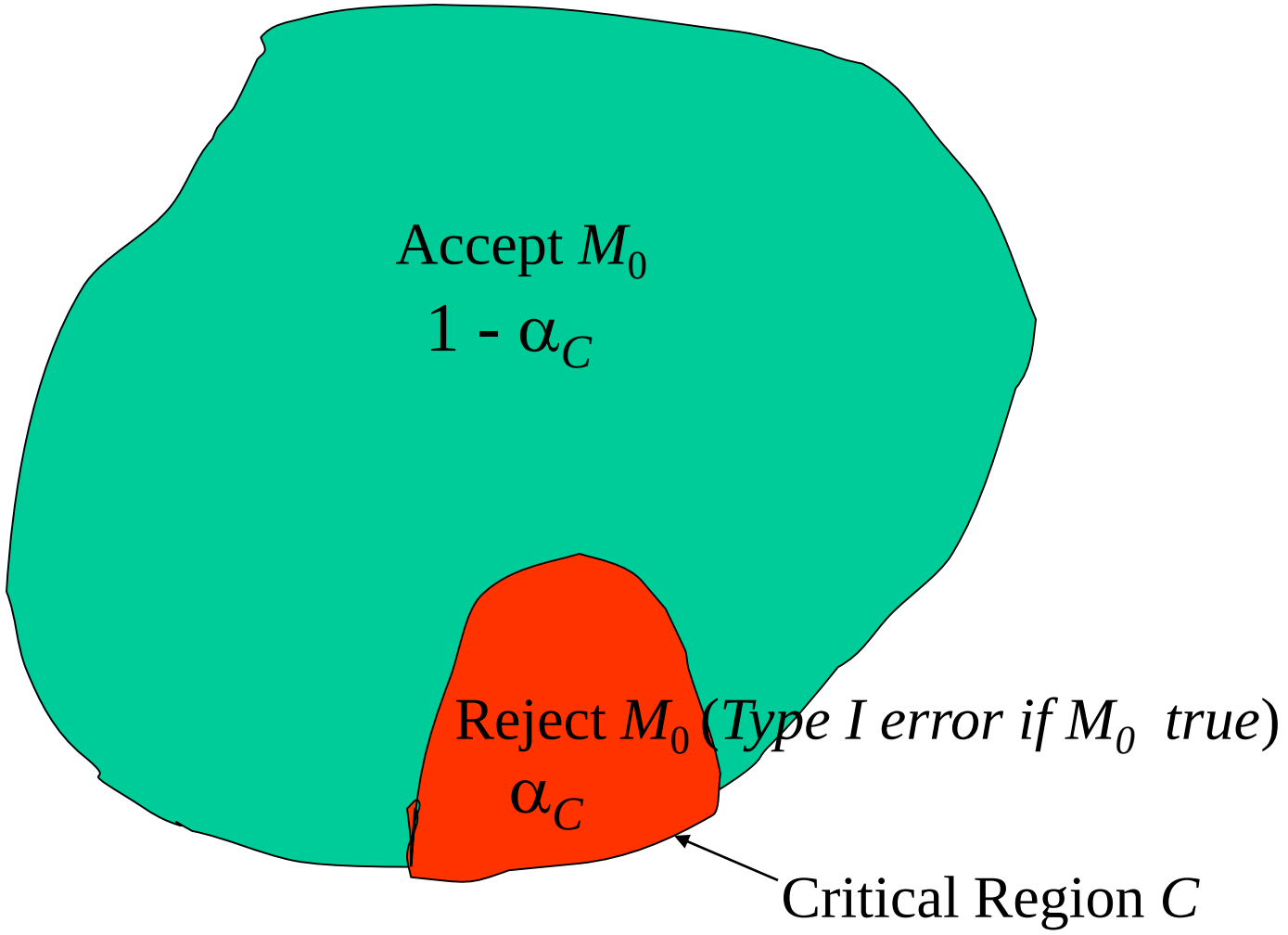
Types of Errors with Hypothesis Test

- a *Type I error* occurs if we reject M_0 when it is true.
  - For a given critical region C, the prob of committing a Type I error is denoted α_C

$$\alpha_C = P(C \mid M_0) = \Sigma_{s \in C} P(s \mid M_0)$$

- α_C is called the *significance level* of the test

Sample Space S – probabilities under M_0

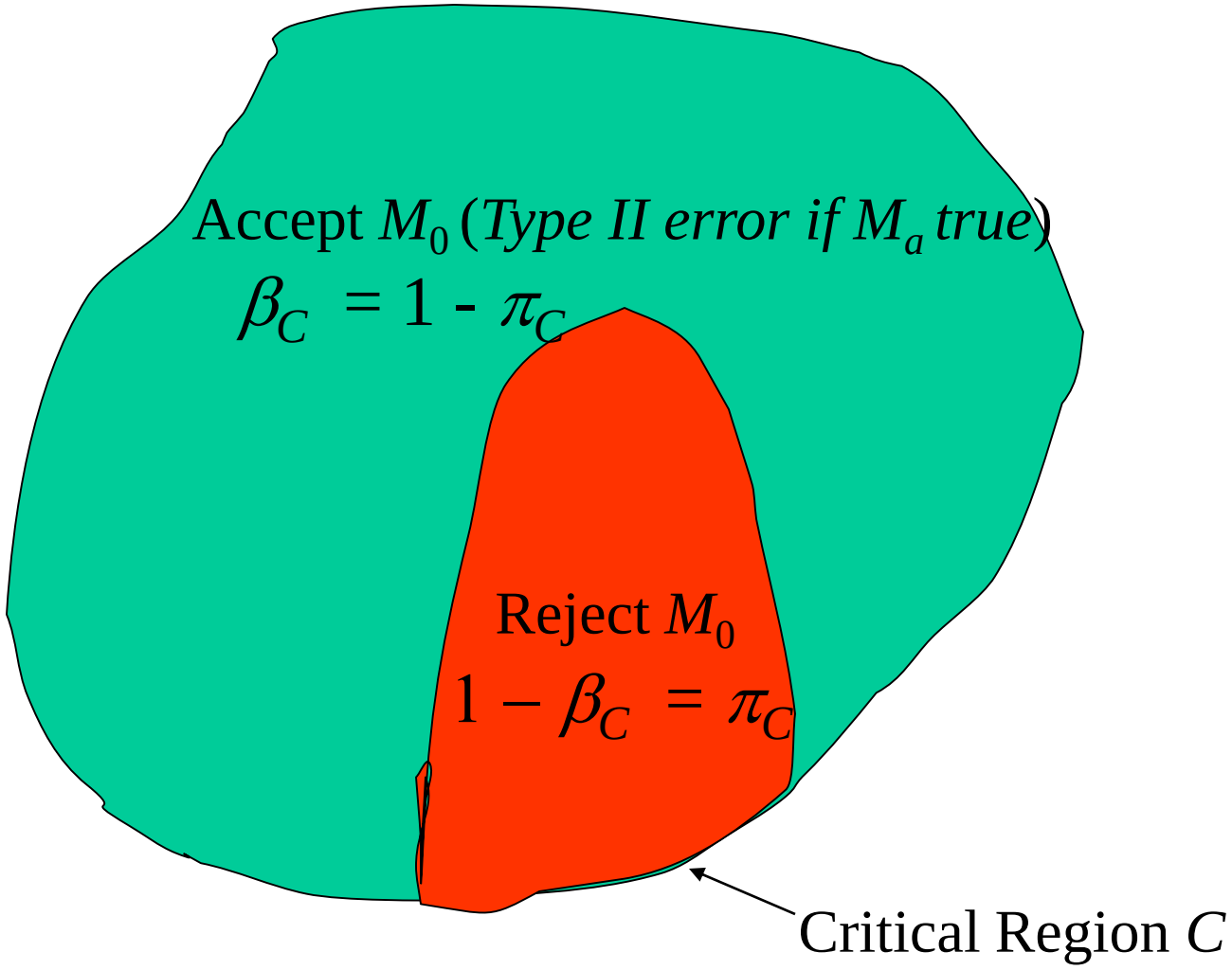


- a *Type II error* occurs if we accept M_0 when it is false.
  - For a given C, prob of committing a Type II error is denoted β_C

$$\beta_C = \Sigma_{s \notin C} P(s \mid M_a) = 1 - P(C \mid M_a)$$

- $\pi_C = 1 - \beta_C$ is called the *power* of the test.

Sample Space S – probabilities under M_a



- Designing a test involves a tradeoff between significance and power
  - smaller C gives smaller Type I error but larger Type II error (lower power).

Likelihood Ratio Tests

- A *likelihood ratio test* of models M_a and M_0 is a hypothesis test of the two models, with critical region C defined by

$$C = C_\Lambda = \{s \mid LR(M_a, M_0 \mid s) \geq \Lambda\}$$

for some non-negative constant Λ, the *cutoff value*.

- Neyman-Pearson lemma motivates use of the *likelihood ratio* as an optimal *discriminator*, or "score"
  - even in contexts where we aren't explicitly testing hypotheses.
- any monotonic function $f(LR)$ of likelihood ratio has equivalent optimality properties
  - because defines the same set of critical regions:

$$LR(M_a, M_0 \mid s) \geq \Lambda \Leftrightarrow f(LR(M_a, M_0 \mid s)) \geq f(\Lambda)$$

- convenient to take f to be the log function, in which case we get the *log likelihood ratio*.

Neyman-Pearson lemma

Let M_a and M_0 be two models, and C_Λ the critical region defined by a likelihood ratio test of M_a vs. M_0 with

- cutoff value Λ,
- significance level α_Λ, and
- power $\pi_\Lambda = 1 - \beta_\Lambda$.

Then if C is any other critical region, we have

- If $\alpha_C < \alpha_\Lambda$, then $\pi_C < \pi_\Lambda$ (and $\beta_C > \beta_\Lambda$)
- If $\alpha_C = \alpha_\Lambda$, then $\pi_C \leq \pi_\Lambda$ (and $\beta_C \geq \beta_\Lambda$)

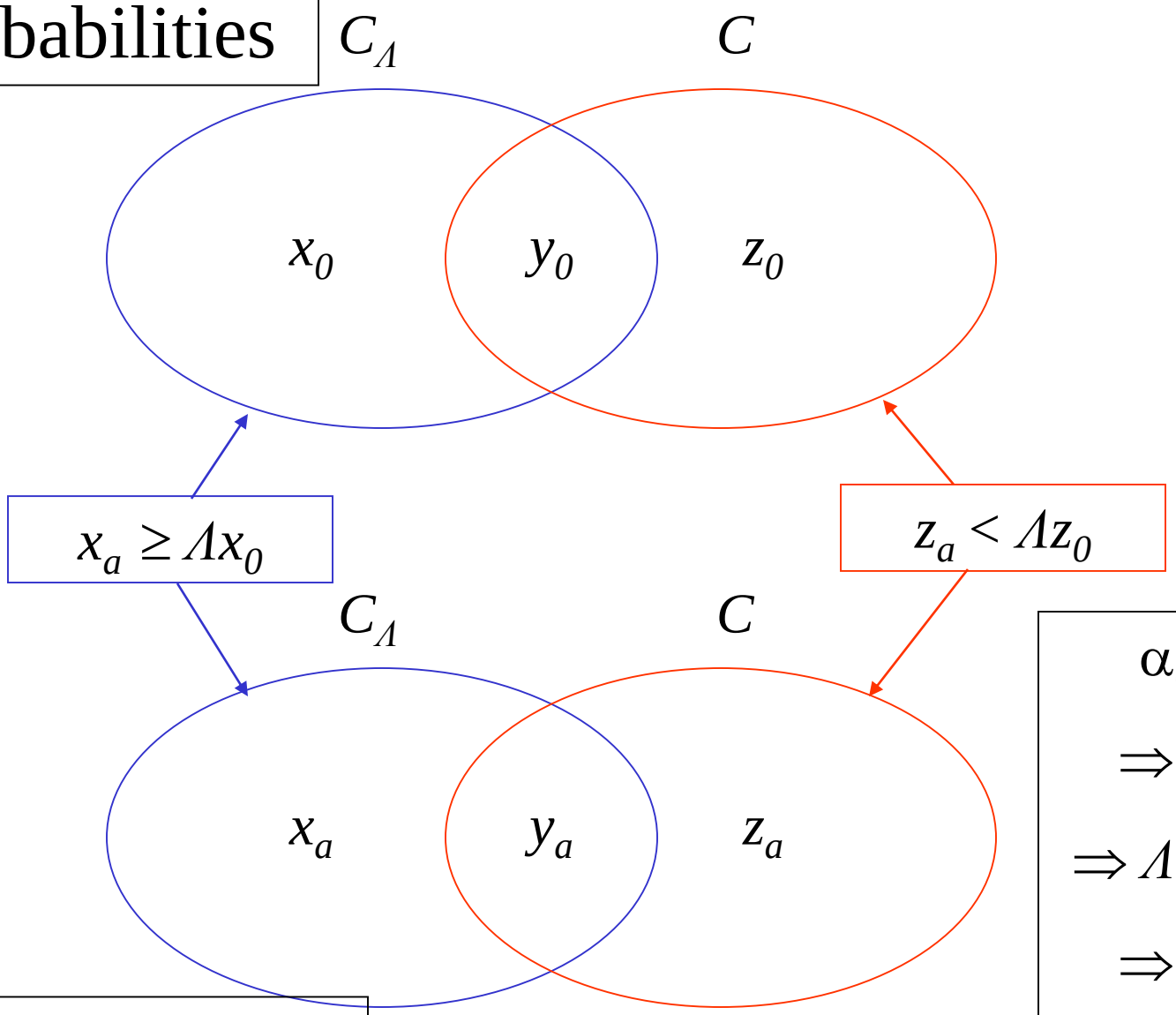
In other words, the likelihood ratio test with significance level α_Λ is the most powerful test

- (has the lowest type II error rate)

with that significance level.

Idea of Neyman-Pearson lemma *proof*:

M_0 probabilities



M_a probabilities

$$\alpha_C < \alpha_\Lambda$$

$$\Rightarrow z_0 < x_0$$

$$\Rightarrow \Lambda\, z_0 < \Lambda\, x_0$$

$$\Rightarrow z_a < x_a$$

$$\Rightarrow \pi_C < \pi_\Lambda$$

- ***Proof***: Suppose $\alpha_C < \alpha_\Lambda$ .Then

$$\sum_{s \in C} P(s \mid M_0) < \sum_{s \in C_\Lambda} P(s \mid M_0)$$

Subtract from both sides the terms involving $s \in C \cap C_\Lambda$ This leaves

$$(1) \quad \sum_{s \in C \setminus C_\Lambda} P(s \mid M_0) < \sum_{s \in C_\Lambda \setminus C} P(s \mid M_0)$$

- By definition of the likelihood ratio test, for any observation s,

$$s \in C_\Lambda \Leftrightarrow P(s \mid M_a) \geq \Lambda P(s \mid M_0)$$

- From this, it follows that

(2) $$\sum_{s \in C \setminus C_\Lambda} \frac{1}{\Lambda} P(s \mid M_a) < \sum_{s \in C \setminus C_\Lambda} P(s \mid M_0)$$

and

(3) $$\sum_{s \in C_\Lambda \setminus C} P(s \mid M_0) \leq \sum_{s \in C_\Lambda \setminus C} \frac{1}{\Lambda} P(s \mid M_a)$$

- Combining (2), (1), and (3)

$$\sum_{s\in C\setminus C_\Lambda}\frac{1}{\Lambda}P(s\,|\,M_a) < \sum_{s\in C\setminus C_\Lambda}P(s\,|\,M_0) < \sum_{s\in C_\Lambda\setminus C}P(s\,|\,M_0) \le \sum_{s\in C_\Lambda\setminus C}\frac{1}{\Lambda}P(s\,|\,M_a)$$

so (cancelling the common factor $1/\Lambda$)

$$\sum_{s\in C\setminus C_\Lambda}P(s\,|\,M_a) < \sum_{s\in C_\Lambda\setminus C}P(s\,|\,M_a)$$

so, adding in the terms corresponding to $s \in C \cap C_\Lambda$

$$\sum_{s\in C}P(s\,|\,M_a) < \sum_{s\in C_\Lambda}P(s\,|\,M_a)$$

i.e $\pi_C < \pi_\Lambda$ The other part of the lemma ($\pi_C \le \pi_\Lambda$ if $\alpha_C = \alpha_\Lambda$) is proved similarly.

Aligning sequences

- Major uses in genome analysis:
  - To find relationship between sequences from "same" genome
    - (still need to allow for discrepancies – due to errors/polymorphisms)

    E.g.
    - finding gene structure by aligning cDNA to genome
    - assembling sequence reads in genome sequencing project
    - NextGen applications: "Resequencing", ChIPSeq, etc
  - To detect evolutionary relationships:
    - illuminates function of distantly related sequences under selection
    - finds corresponding positions in neutrally evolving sequence
      - to illuminate mutation process
      - helps find non-neutrally evolving (functional) regions

- Often we're interested in details of alignment
  - (i.e. precisely which residues are aligned),

  but
- sometimes only interested in whether alignment score is large enough to imply that sequences are likely to be related

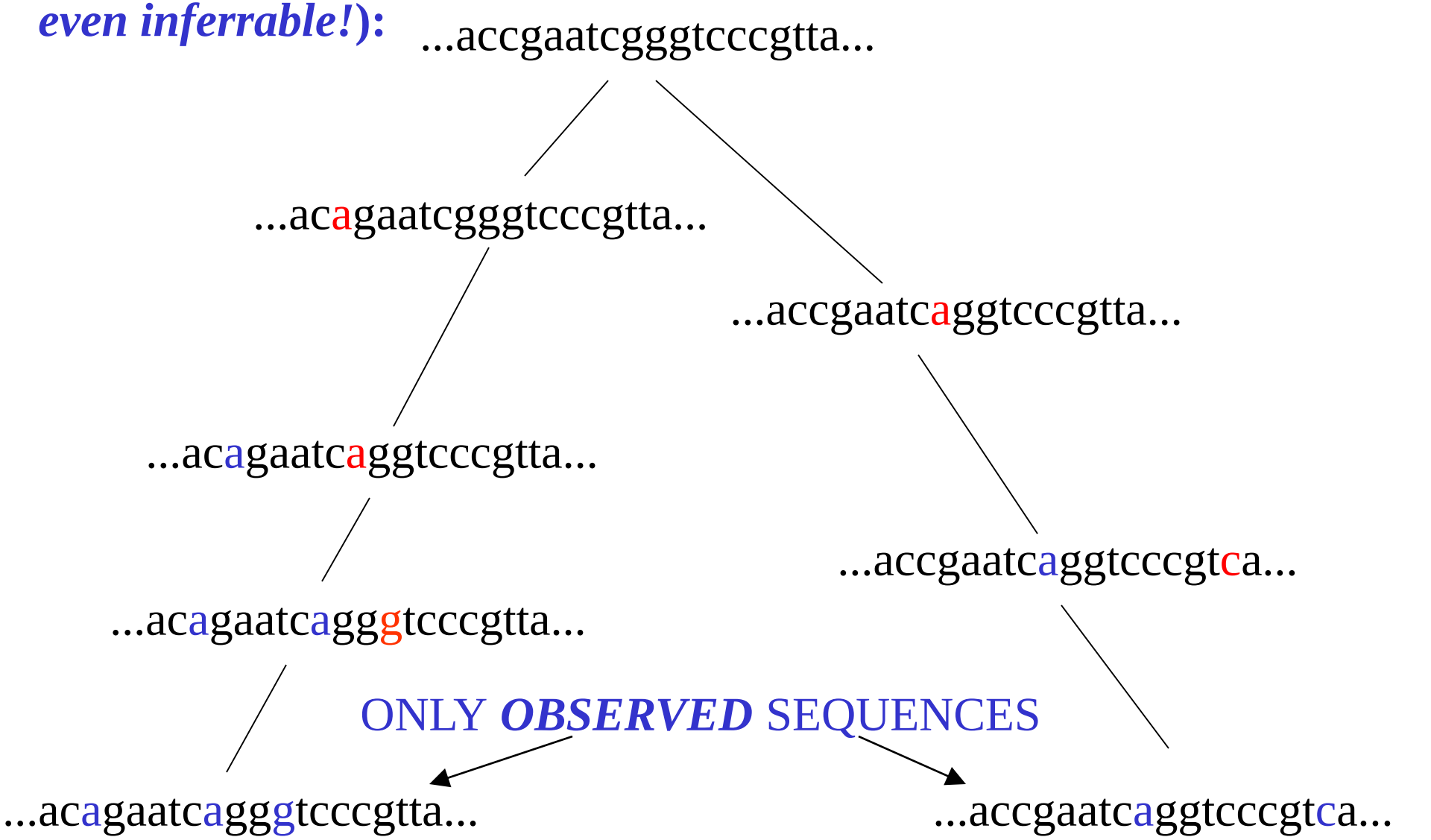
Sequences & evolution

- Similar sequences of sufficient length usually have a common evolutionary origin
  - i.e. are homologous
- For a pair of sequences
  - "% similarity" makes sense
  - "% homology" doesn't
- In alignment of two homologous sequences
  - differences mostly represent *mutations* that occurred in one or both lineages, but
  - Not all mutations are inferrable from the alignment

(Observed) ALIGNMENT:
(*may not be unique*!)

```
...acagaatcagggtcccgtta...
...accgaatcagg-tcccgtca...
```

(Unobserved) MUTATION HISTORY (*in general, this is not even inferrable!*):



Complications

- Parallel & back mutations

  $\Rightarrow$ estimating total # of mutations requires statistical modelling

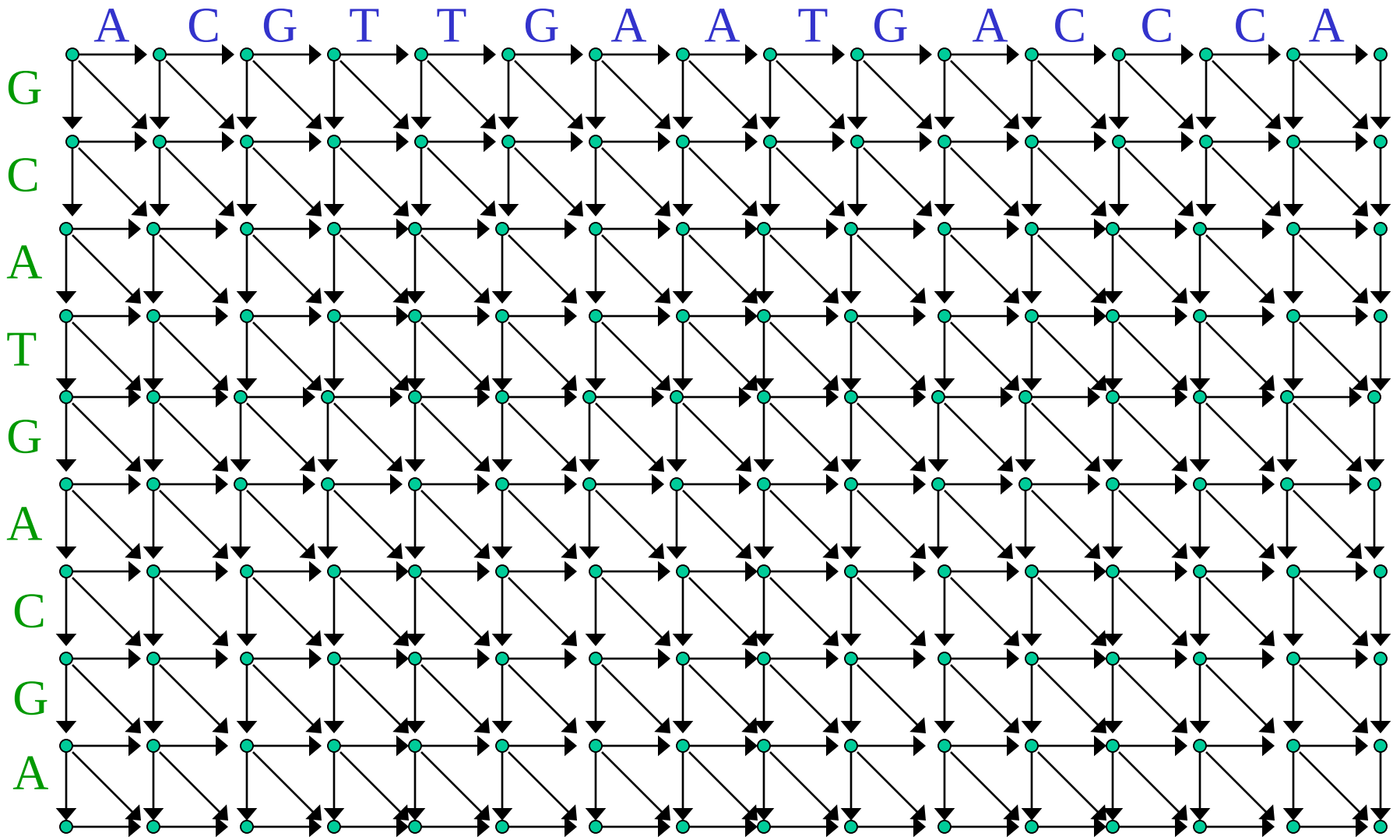
- Insertion/deletion, & segmental mutations

  $\Rightarrow$ finding the correct alignment can be problematic (‘gap attraction’)

  -- even in closely related sequences!

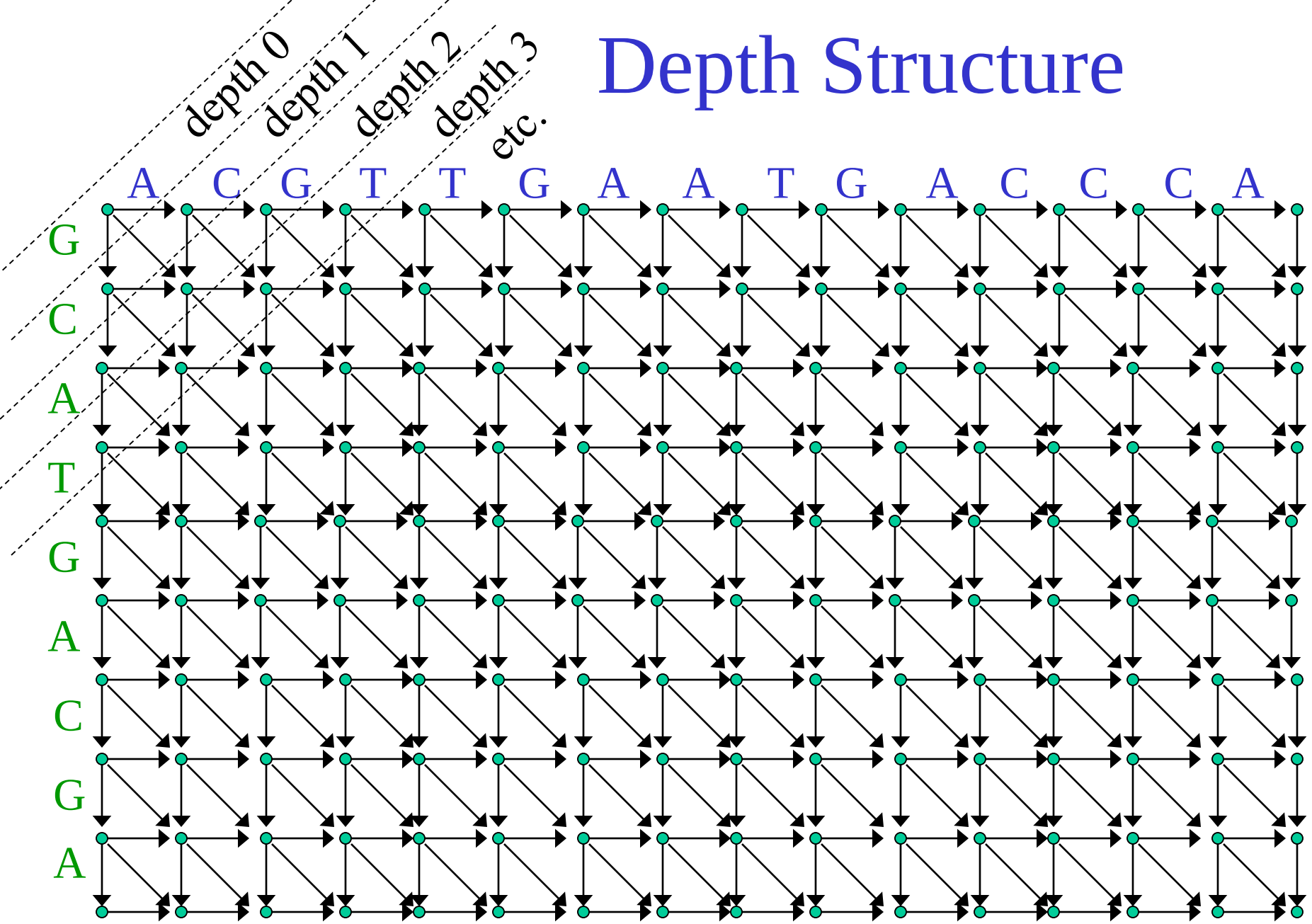
Sequence alignments correspond to *paths* in a *DAG*!

The *Edit Graph* for a Pair of Sequences

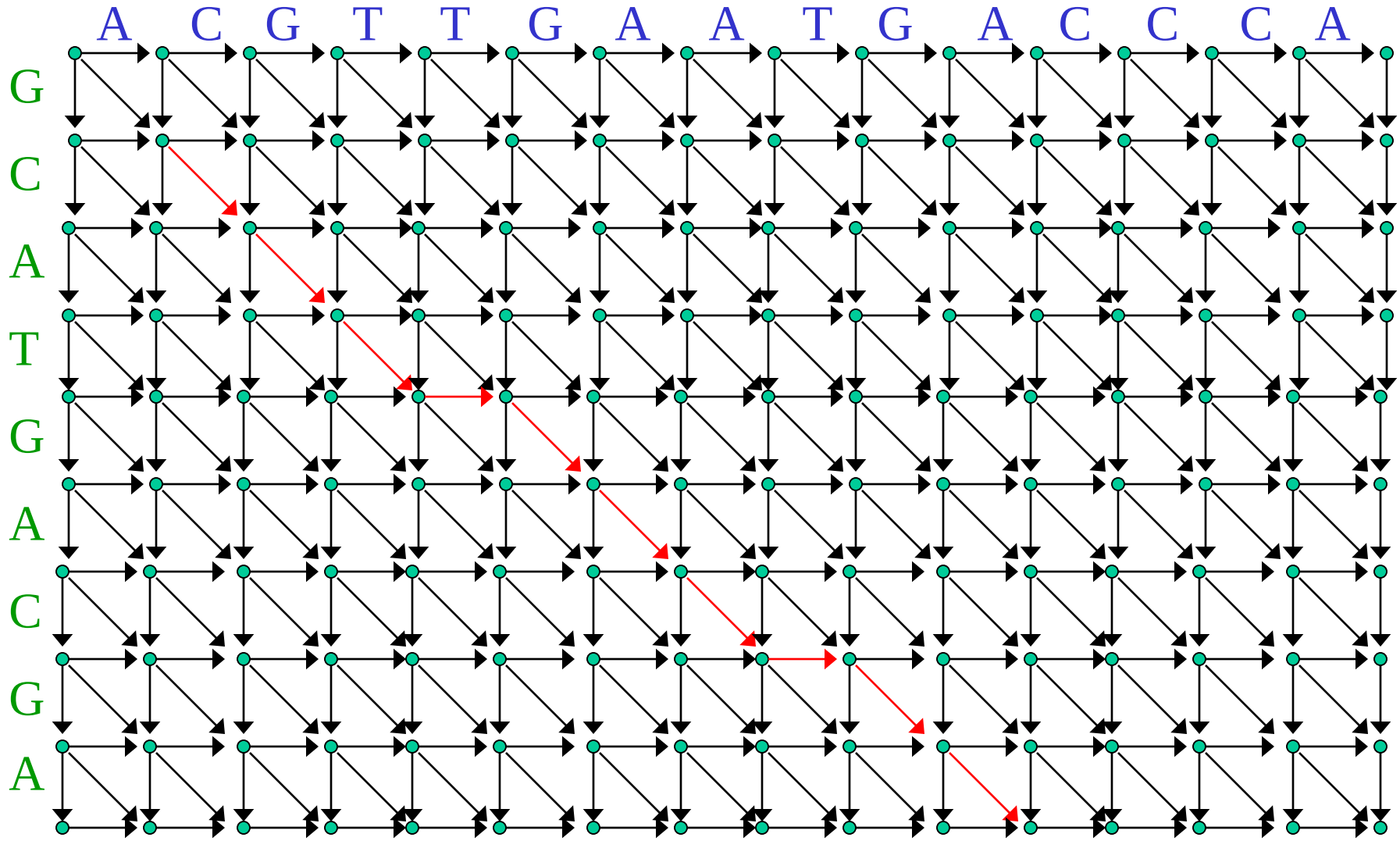


- The edit graph is a DAG.
  - Except on the boundaries, the nodes have in-degree and out-degree both 3.
- The depth structure is as shown on the next slide. Child of node of depth *n* always has
  - depth *n* + 1 (for a horizontal or vertical edge), or
  - depth *n* + 2 (for a diagonal edge).

Depth Structure



- *Paths* in edit graph correspond to *alignments* of subsequences
  - each edge on path corresponds to alignment column.
  - diagonal edges correspond to column of two aligned residues;
  - horizontal edges correspond to column with
    - residue in 1st (top, horizontal) sequence
    - gap in the 2d (vertical) sequence
  - vertical edges correspond to column with
    - residue in 2d sequence
    - gap in 1st sequence



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

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

Weights on Edit Graphs

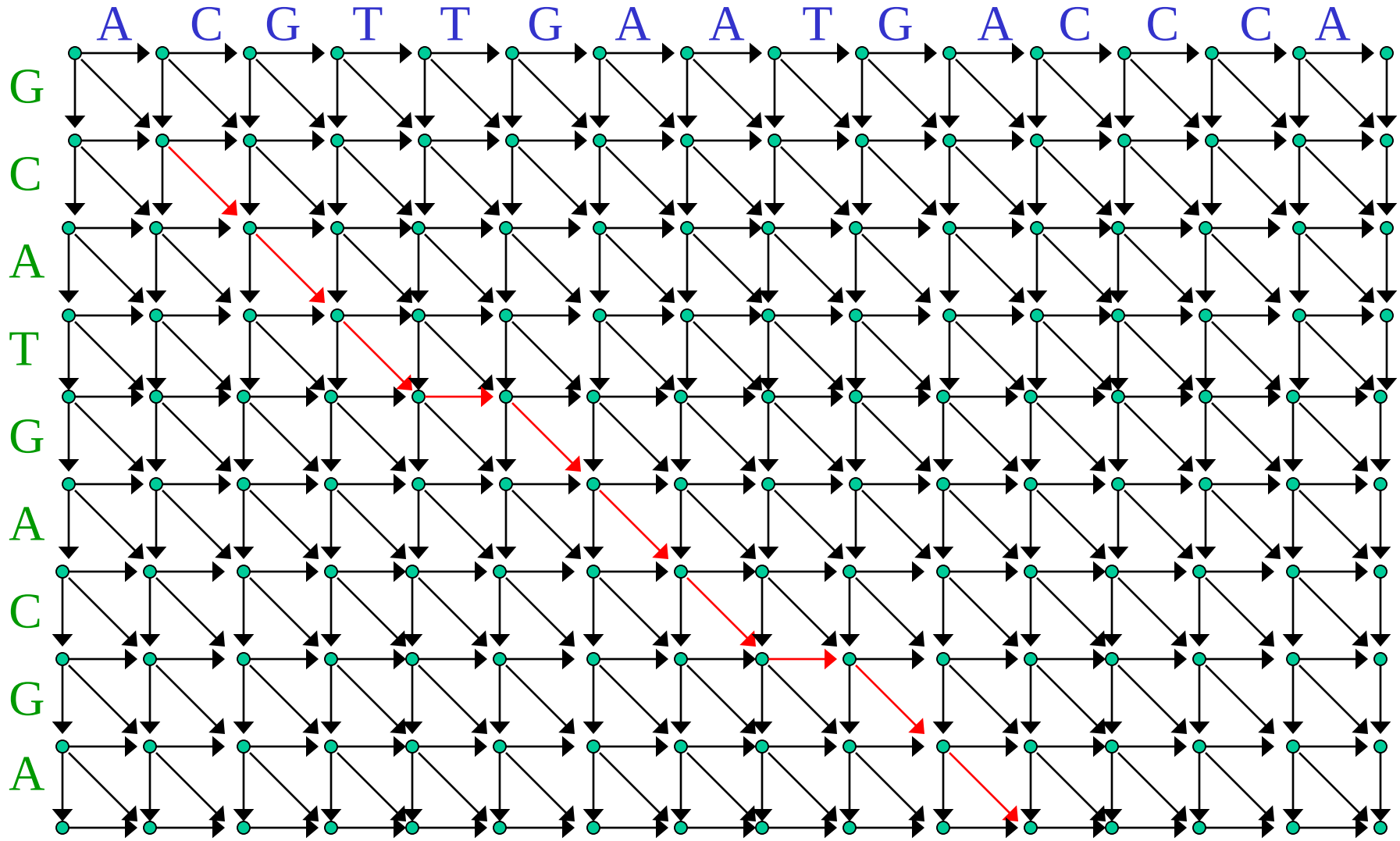
- Edge weights correspond to scores on alignment columns.
- Highest weight path corresponds to highest-scoring alignment for that scoring system.
- Weights may be assigned using
  - a *substitution score matrix*,
    - assigns a score to each possible pair of residues occurring as alignment column

    and
  - a *gap penalty*
    - assigns a score to column consisting of residue opposite a gap.
  - Example for protein sequences: BLOSUM62

BLOSUM62 Score Matrix

GAP -12 -2

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0	-4
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1	-4
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1	-4
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1	-4
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2	-4
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1	-4
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1	-4
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1	-4
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1	-4
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1	-4
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1	-4
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1	-4
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2	-4
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1	-4
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1	-4
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1	-4
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1	-4
*	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	1



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

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

Alignment algorithms

- *Smith-Waterman* algorithm to find highest scoring alignment

  = dynamic programming algorithm to find highest-weight path

  – Is a *local* alignment algorithm:
    - finds alignment of subsequences rather than the full sequences.
- Can process nodes in any order in which parents precede children. Commonly used alternatives are
  – depth order
  – row order
  – column order