CSE 397-497: *Computational Issues in Molecular Biology*

Lecture 3

Spring 2004

Remember

For your class lecture, you must give me a ranked list of your top 3 topics in order of preference by 5:00 pm today:

order we will cover topics in course

- sequence comparison & alignment (pairwise & multiple),
- sequencing and sequence assembly,
- physical mapping of DNA,
- phylogenetic trees,
- genome rearrangements,
- RNA and protein structure prediction,
- DNA microarrays,
- DNA computing.

Sequence comparison and alignment

Consider:

$$\begin{tikzcd} \mathtt{AGTAGCATC} \arrow[d, leftrightarrow] & \textit{versus} & \mathtt{AGTAGCATC} \arrow[d, leftrightarrow] \\ \mathtt{AGTGCACC} & & \mathtt{GACACGATT} \end{tikzcd}$$

That the two DNA fragments on the left somehow seem more similar than the two on the right could be significant.

Question: how can we measure sequence similarity?

Motivation

Why is this important?

- Given a new DNA sequence, one of the first things a biologist will want to do is search databases of known sequences to see if anyone has recorded something similar. (As we've seen, genetic sequences are long and the databases are enormous, so efficiency will be an issue.)
- Sequence similarity can provide clues about function.
- Similarity can provide clues about evolutionary relationships.
- Many other problems from computational biology incorporate some notion of sequence similarity as a basic premise.

Sequence concepts

A *sequence* is a linear string of symbols over a finite alphabet.

(Note: *string* and *sequence* are often used synonymously.)

Some basic sequence concepts:

length	number of symbols in *s* (written |*s*|).
empty string	sequence of length 0 (written ε).
subsequence	sequence that can be obtained from *s* by removing some symbols (**`ACG`** is a subsequence of **`TATCTG`**).
supersequence	if *t* is a subsequence of *s*, then *s* is a supersequence of *t*.

Sequence concepts

More basic sequence concepts:

substring	sequence of consecutive symbols appearing in *s* (**`ACG`** is <u>not</u> a substring of **`TATCTG`**, but **`TCT`** is).

(Observation: every substring is a subsequence of the string in question, but not vice versa.)

superstring	if *t* is a substring of *s*, then *s* is a superstring of *t*.
interval	set of consecutive indices of a string, e.g., [2..4] refers to 2*nd* through 4*th* symbols of string *s*, or *s*[2]*s*[3]*s*[4].

Sequence concepts

And lastly:

prefix	substring of s of the form $s[1..j]$ where $0 \leq j \leq \|s\|$ (when $j = 0$, prefix is the empty string).
suffix	substring of s of the form $s[i..\|s\|]$ where $1 \leq i \leq \|s\| + 1$ (when $i = \|s\| + 1$, suffix is the empty string).

AT is a prefix (and substring and subsequence) of **ATCCAG**.

AG is a suffix (and substring and subsequence) of **ATCCAG**.

Sequences

The concept of a sequence is extremely broad. In this course, we are concerned with genetic sequences. However, there are other important kinds of sequence data:

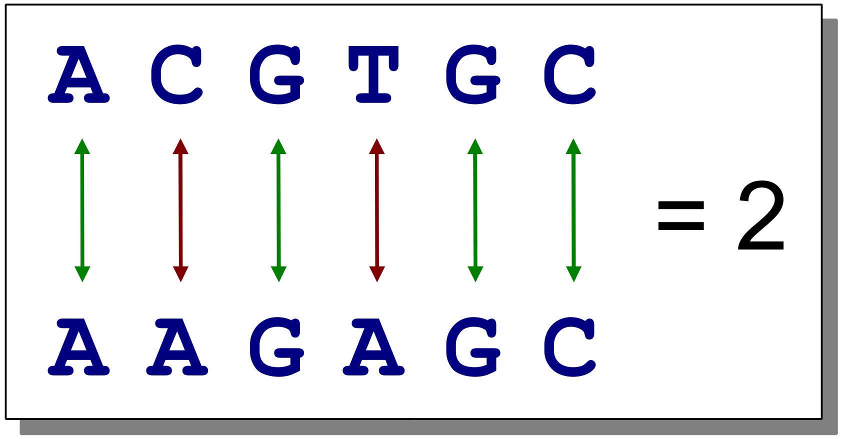
- ASCII text,
- speech,
- handwriting (pen-strokes).

Likewise, the same algorithmic techniques turn up again and again, often under different names:

- approximate string matching,
- edit (or evolutionary) distance,
- dynamic time warping.

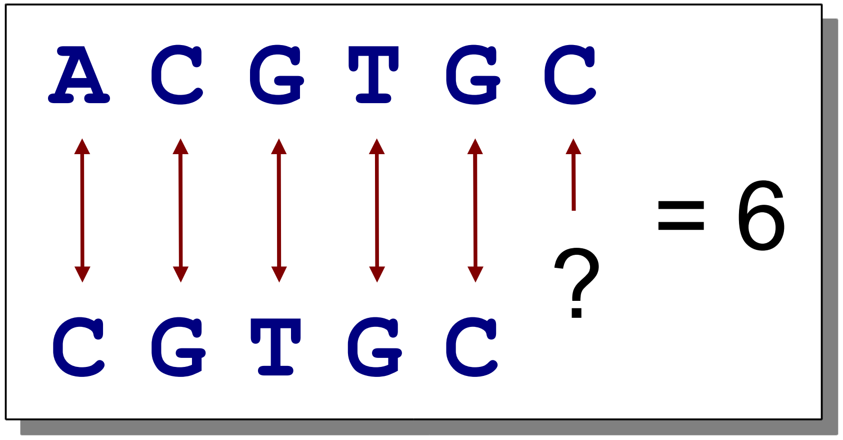
Sequence comparison: a false start

An obvious idea that comes to mind is to line up each symbol and count the number that don't match:



This is known as Hamming distance and forms the basis for most error correcting codes.

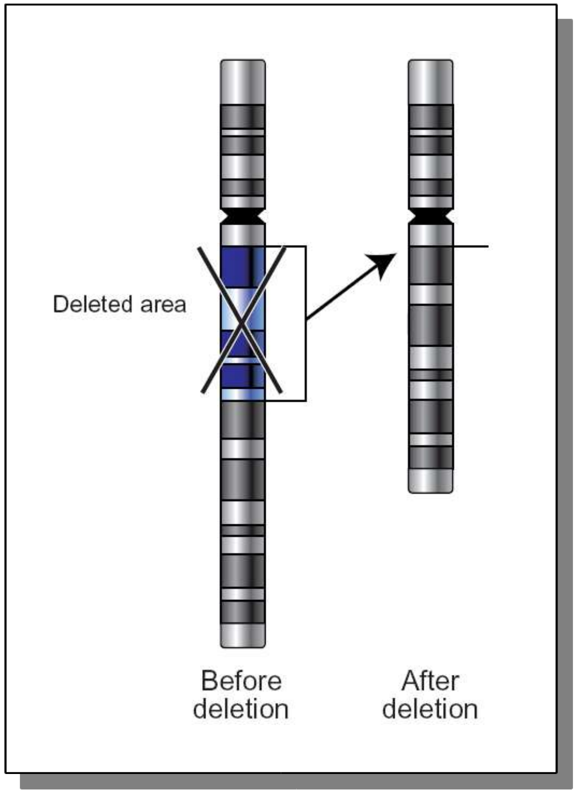
But it doesn't work for the kinds of sequences we care about:

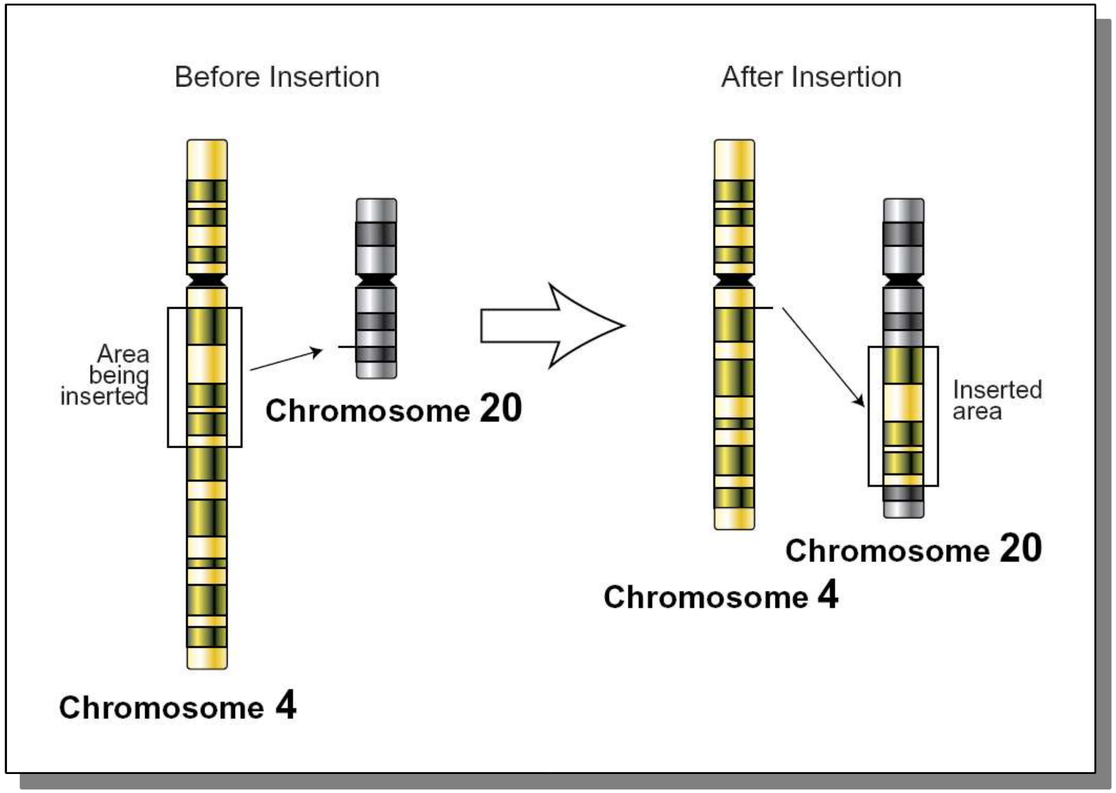


Just one missing symbol at the start of the second sequence leads to a large distance.

Sequence manipulation at the genetic level

Genomes aren't static ...



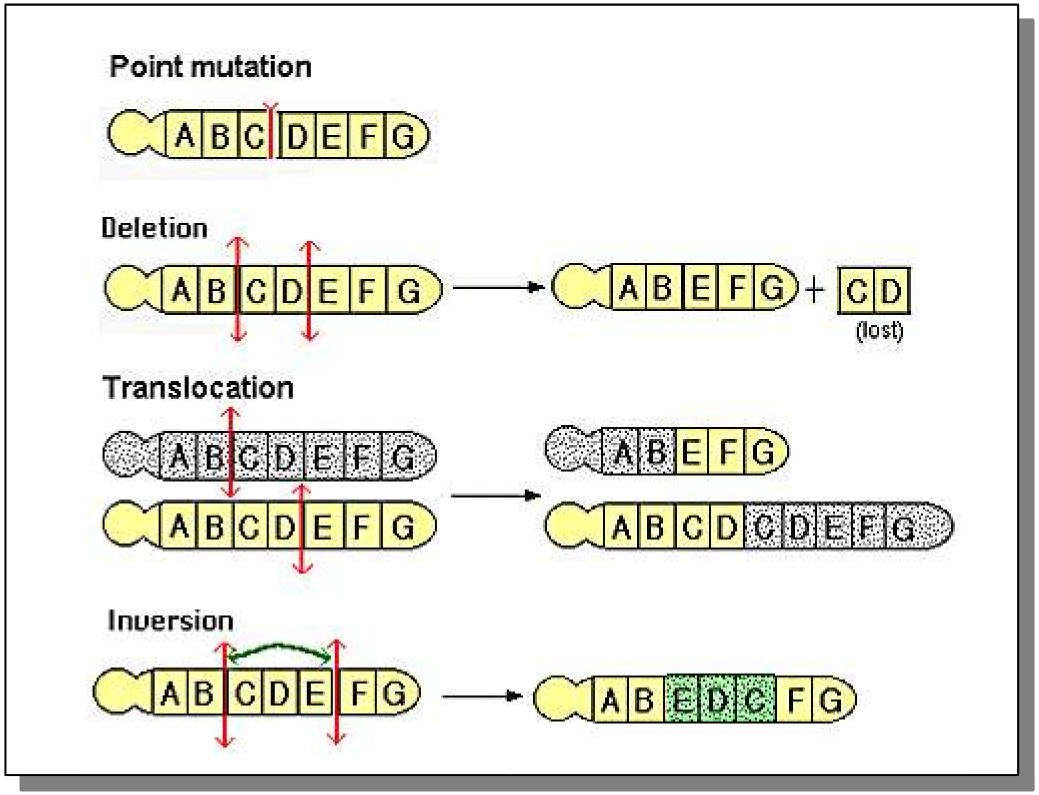


... sequence comparison must account for this.

http://www.accessexcellence.org/AB/GG/nhgri_PDFs/deletion.pdf
http://www.accessexcellence.org/AB/GG/nhgri_PDFs/insertion.pdf

Sequence manipulation at the genetic level

Different kinds of mutations can arise during DNA replication.



http://www.accessexcellence.org/AB/GG/mutation.html

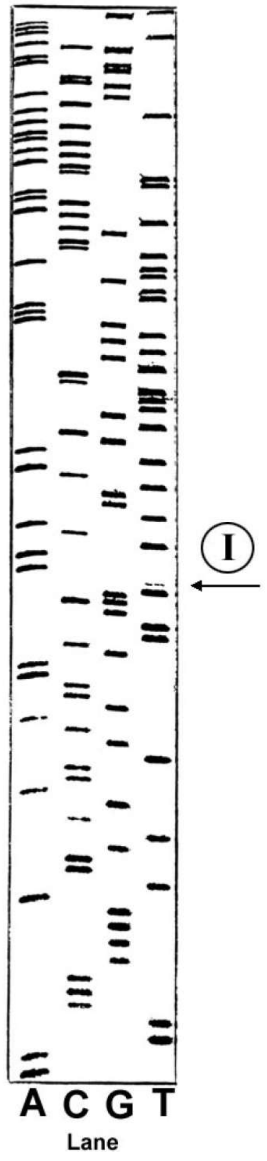
The human factor

In addition, errors can arise during the sequencing process:

> "...the error rate is generally less than 1% over the first 650 bases and then rises significantly over the remaining sequence."
>
> http://genome.med.harvard.edu/dnaseq.html

A hard-to-read gel (arrow marks location where bands of similar intensity appear in two different lanes):

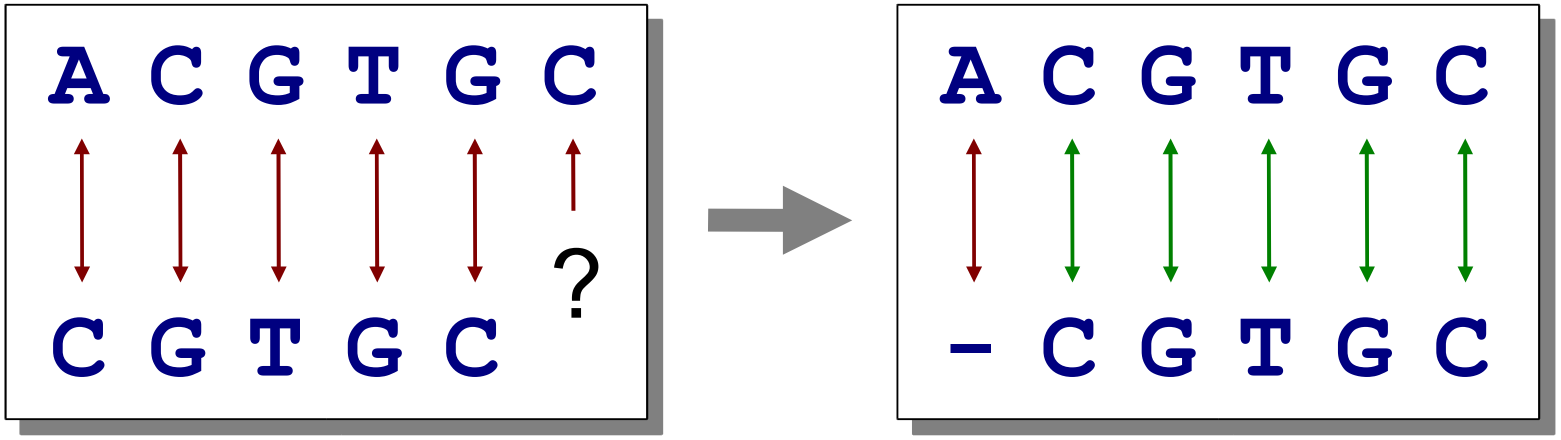
http://hshgp.genome.washington.edu/teacher_resources/99-studentDNASequencingModule.pdf



Sequence alignment

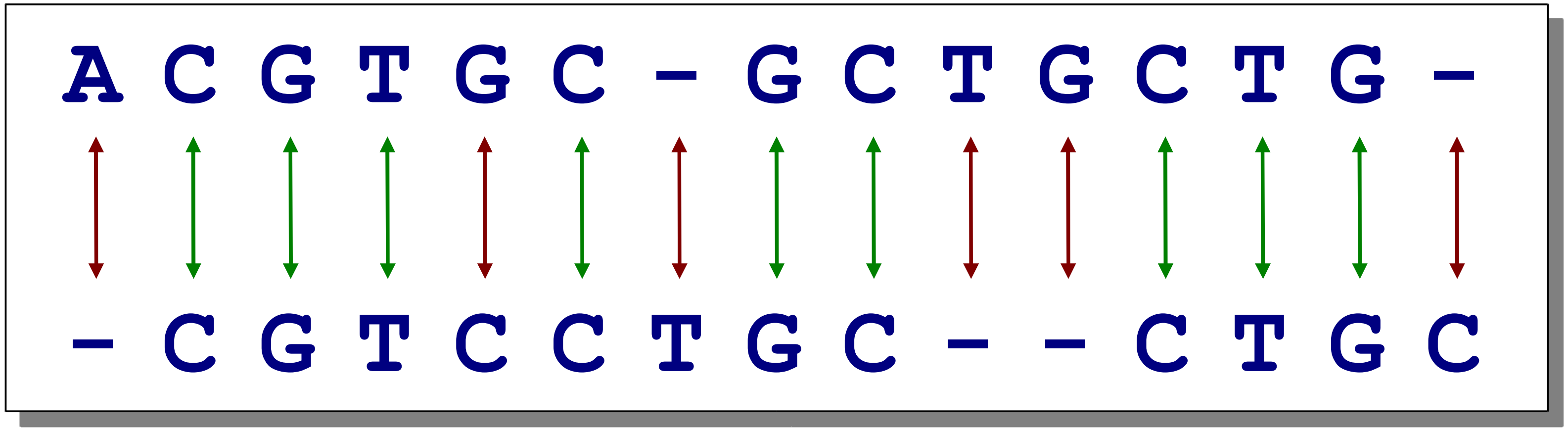
As we have seen, the two sequences we wish to compare may have different lengths. As a result, we need to allow for deletions and insertions.

The notion of an *alignment* helps us visualize this:

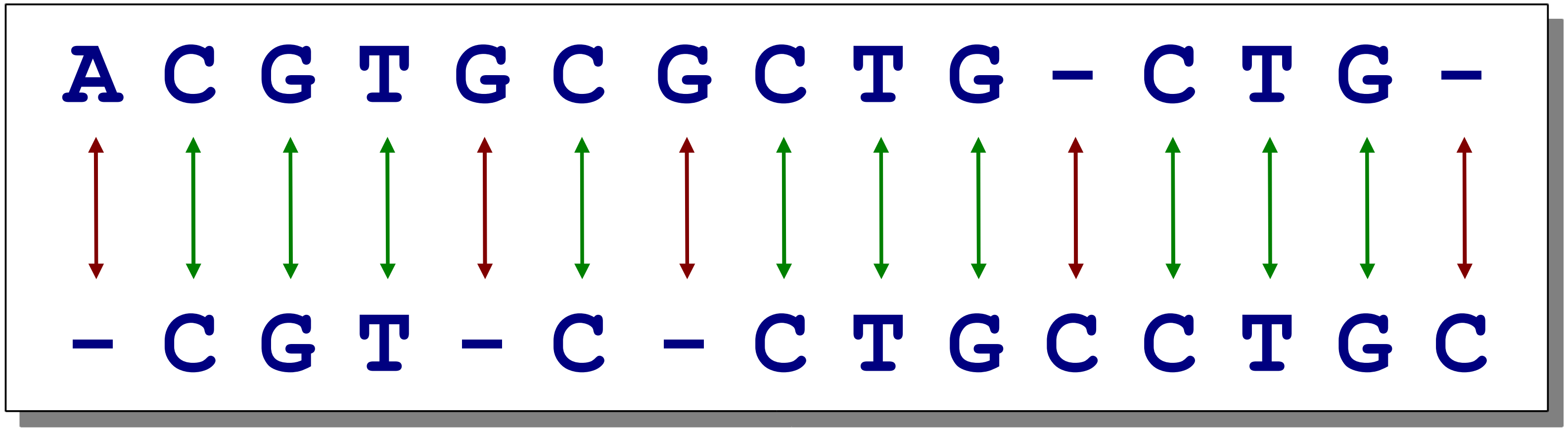


Alignment permits us to incorporate “spaces” (represented by a dash) in one or both sequences to make them the same length.

Sequence alignment



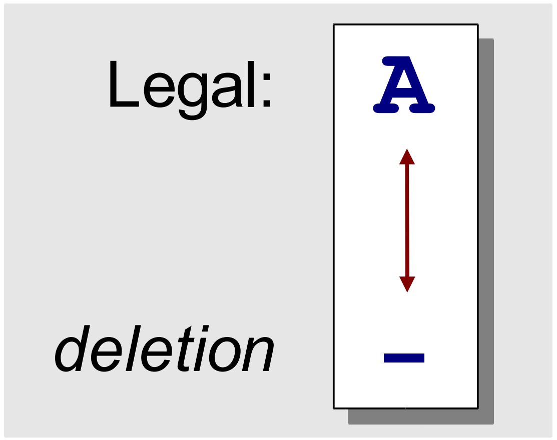
This alignment has 6 mismatches. Is it the best possible?

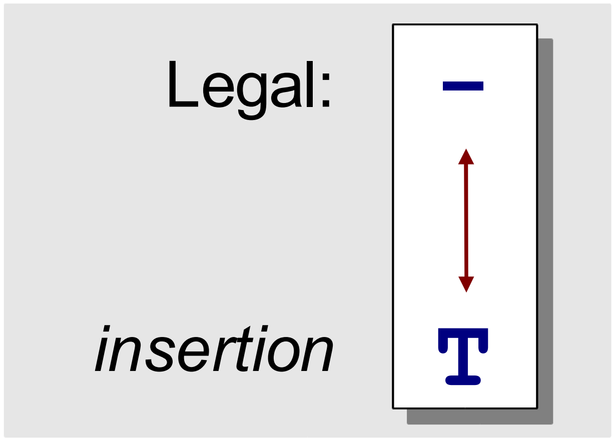


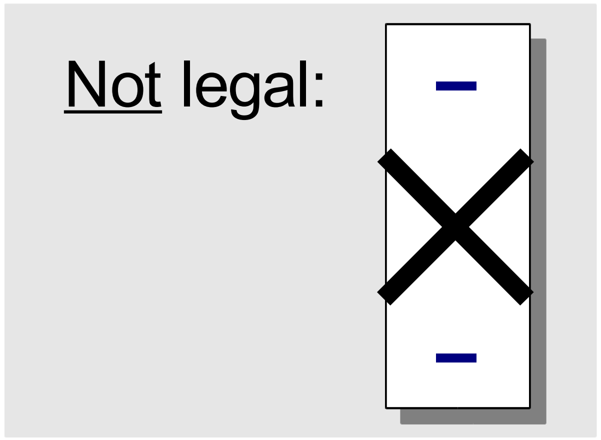
How can we find the best alignment?

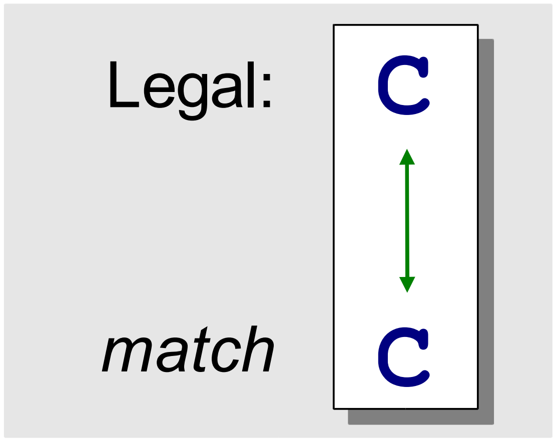
Sequence alignment

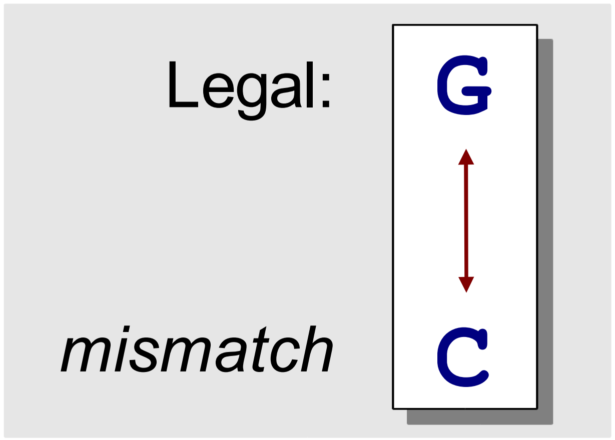
First some ground-rules.











Sequence comparison: the basic algorithm

We can't afford to enumerate all possible alignments looking for the best one – that would be an exponential search.

Fortunately we don't have to. The optimal alignment can be found using a technique known as *dynamic programming*.

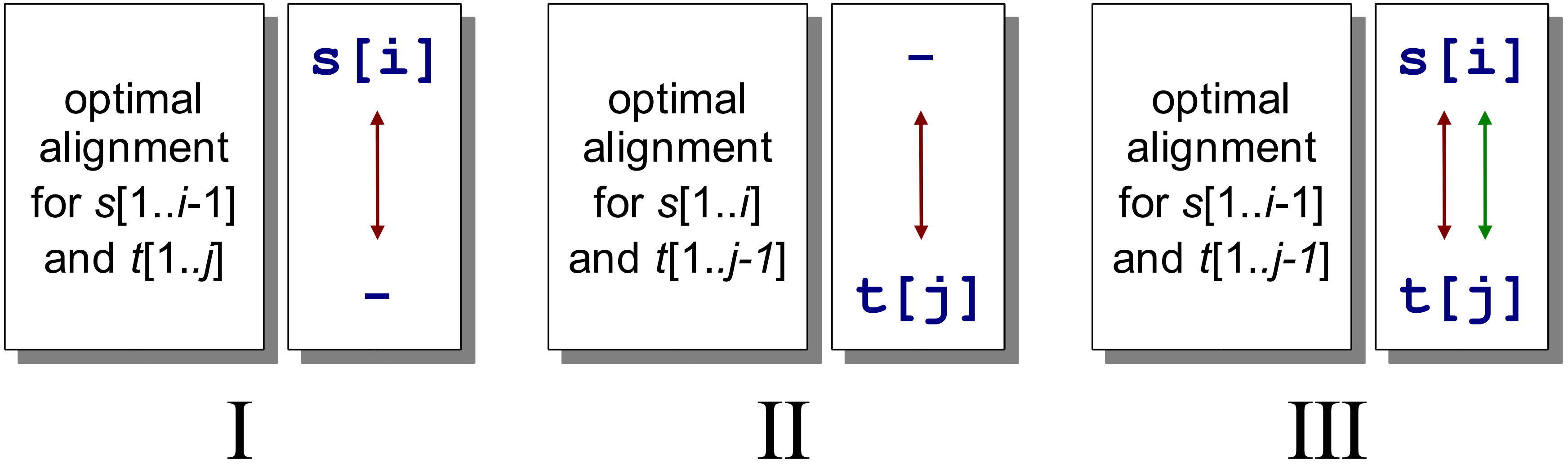
Dynamic programming is based on the premise of computing the solutions to smaller subproblems first and then using these to solve successively larger problems until we have our answer.

(Dynamic programming was invented by Richard Bellman in the 1950's. Its application to sequence comparison came later, in the 1970's.)

http://fens.sabanciuniv.edu/msie/operations_research_50_years/anniversary/or50/1526-5463-2002-50-01-0048.pdf

Sequence comparison: the basic algorithm

Given two sequences *s* and *t*, consider what's required to compute optimal alignment for prefixes *s*[1..*i*] and *t*[1..*j*]. Based on our rules for alignments, there are three possible cases:


optimal alignment for s[1..i-1] and t[1..j]
s[i]
-
I
optimal alignment for s[1..i] and t[1..j-1]
-
t[j]
II
optimal alignment for s[1..i-1] and t[1..j-1]
s[i]
t[j]
III

So, assuming we've already computed solutions for all shorter prefixes, we can compute the alignment for *s*[1..*i*] and *t*[1..*j*].

Sequence comparison: the basic algorithm

Conceptually, this might look something like this:

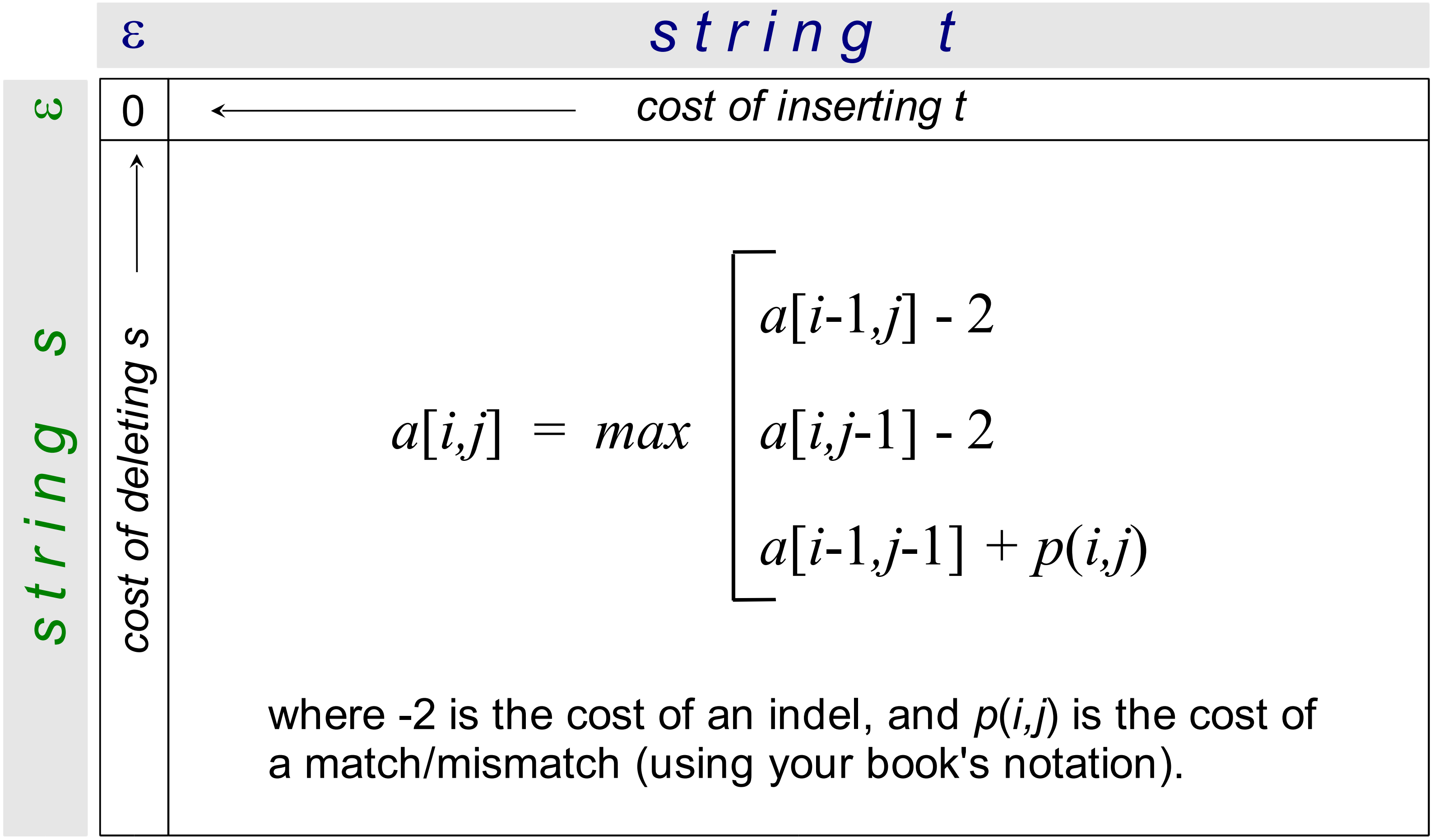
optimal alignment at *s*[1..*i*] and *t*[1..*j*] = max

- optimal alignment at *s*[1..*i-1*] and *t*[1..*j*] + cost of deleting *s*[*i*]
- optimal alignment at *s*[1..*i*] and *t*[1..*j-1*] + cost of inserting *t*[*j*]
- optimal alignment at *s*[1..*i-1*] and *t*[1..*j-1*] + cost of matching *s*[*i*] and *t*[*j*]

Here we assume that deletions, insertions, and mismatches have negative costs, while matches have positive cost.

Sequence comparison: the basic algorithm

This computation can be viewed as building a 2-D matrix:



$$a[i,j] = \max \begin{cases} a[i-1,j] - 2 \\ a[i,j-1] - 2 \\ a[i-1,j-1] + p(i,j) \end{cases}$$

where -2 is the cost of an indel, and *p*(*i*,*j*) is the cost of a match/mismatch (using your book's notation).

Sequence comparison: the basic algorithm

Stated more generally, say that our two sequences are:

$$s[1]s[2]s[3]...s[m] \qquad\qquad t[1]t[2]t[3]...t[n]$$

Then:

$$a[0,0] = 0$$

$$a[i,0] = a[i\text{-}1,j] + c_{del}(s[i]) \qquad 1 \le i \le m$$

$$a[0,j] = a[0,j\text{-}1] + c_{ins}(t[j]) \qquad 1 \le j \le n$$

And:

$$a[i,j] = max \left[\begin{array}{l} a[i\text{-}1,j] + c_{del}(s[i]) \\ a[i,j\text{-}1] + c_{ins}(t[j]) \\ a[i\text{-}1,j\text{-}1] + c_{sub}(s[i],t[j]) \end{array} \right. \qquad 1 \le i \le m,\ 1 \le j \le n$$

Where c_{del}, c_{ins}, and c_{sub} are the costs of a deletion, an insertion, and a substitution, respectively.

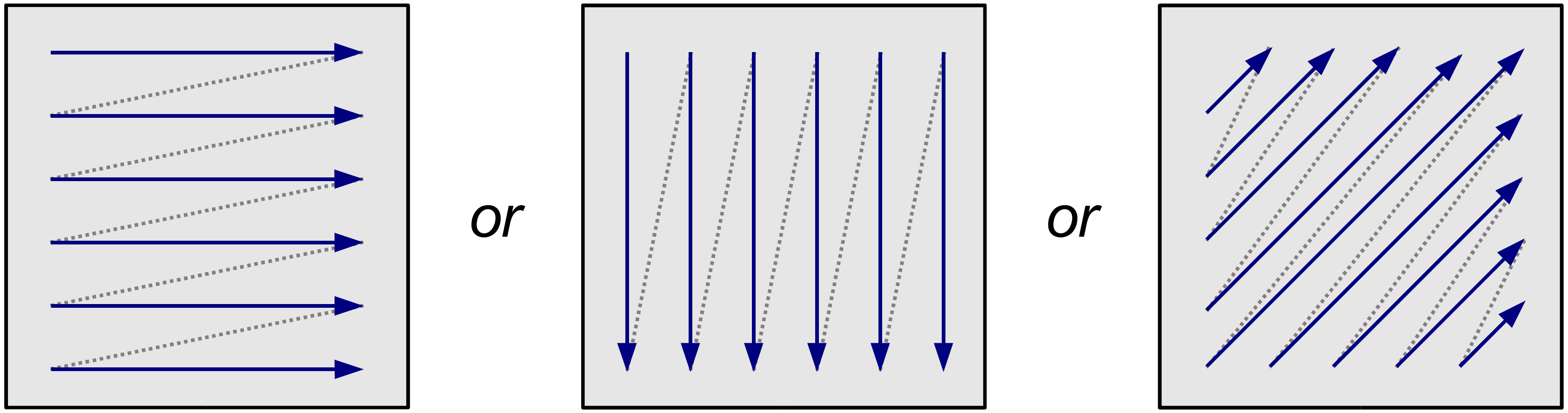
Sequence comparison: the basic algorithm

Example: say that c_{del} = -1, c_{ins} = -1,
c_{sub} = -1 if mismatch and +1 if match

		C	A	T
	0	-1	-2	-3
A	-1	-1	0	-1
C	-2	0	-1	-1
G	-3	-1	-1	-2
T	-4	-2	-2	0

Sequence comparison: some observations

Computation can progress in a number of ways:



For sequences of length *m* and *n*,

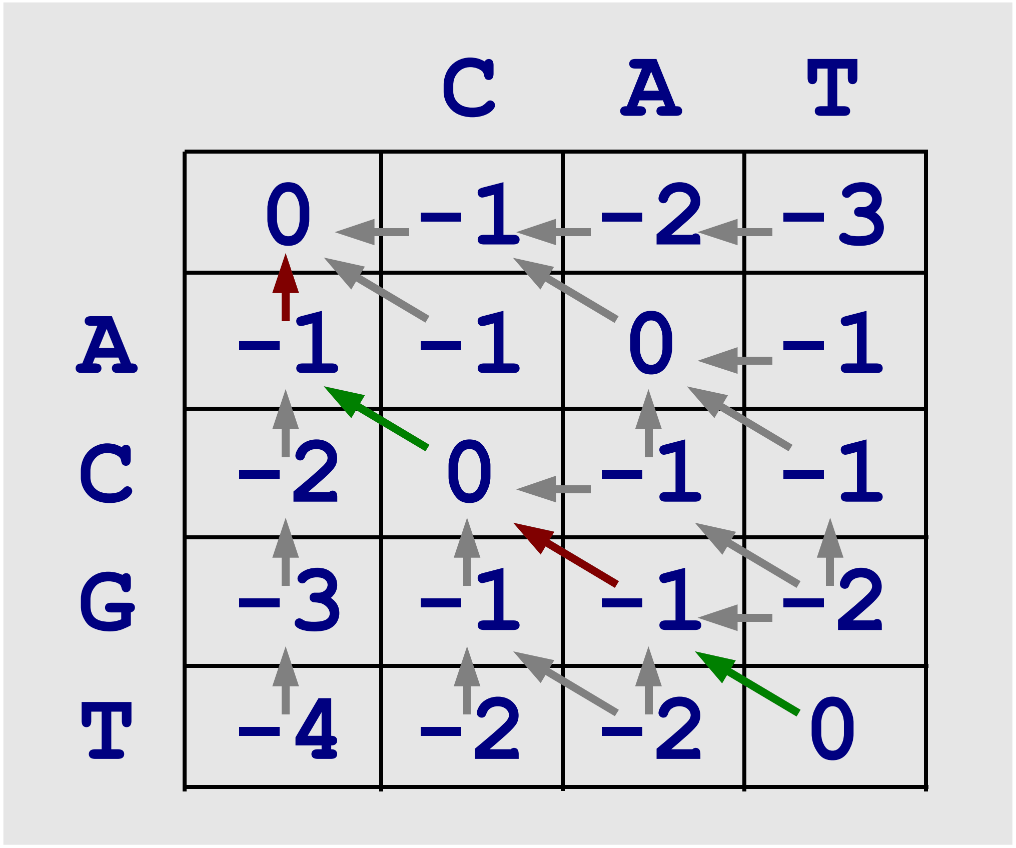
$$s[1]s[2]s[3]...s[m] \qquad t[1]t[2]t[3]...t[n]$$

Computation time = $O(mn)$, space = $O(mn)$.

Sequence comparison: getting an alignment

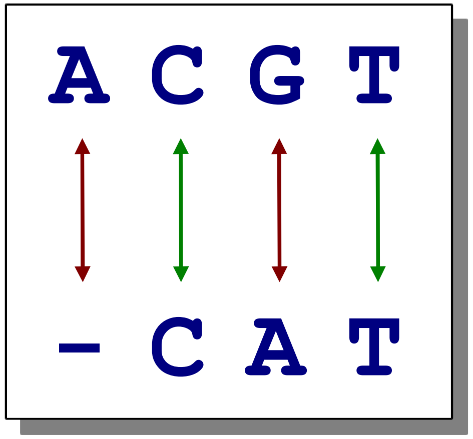
We started with the notion of alignment. How do we get this?

By keeping track of optimal decisions made during algorithm,



		C	A	T
	0	-1	-2	-3
A	-1	-1	0	-1
C	-2	0	-1	-1
G	-3	-1	-1	-2
T	-4	-2	-2	0

and then tracing back optimal path.

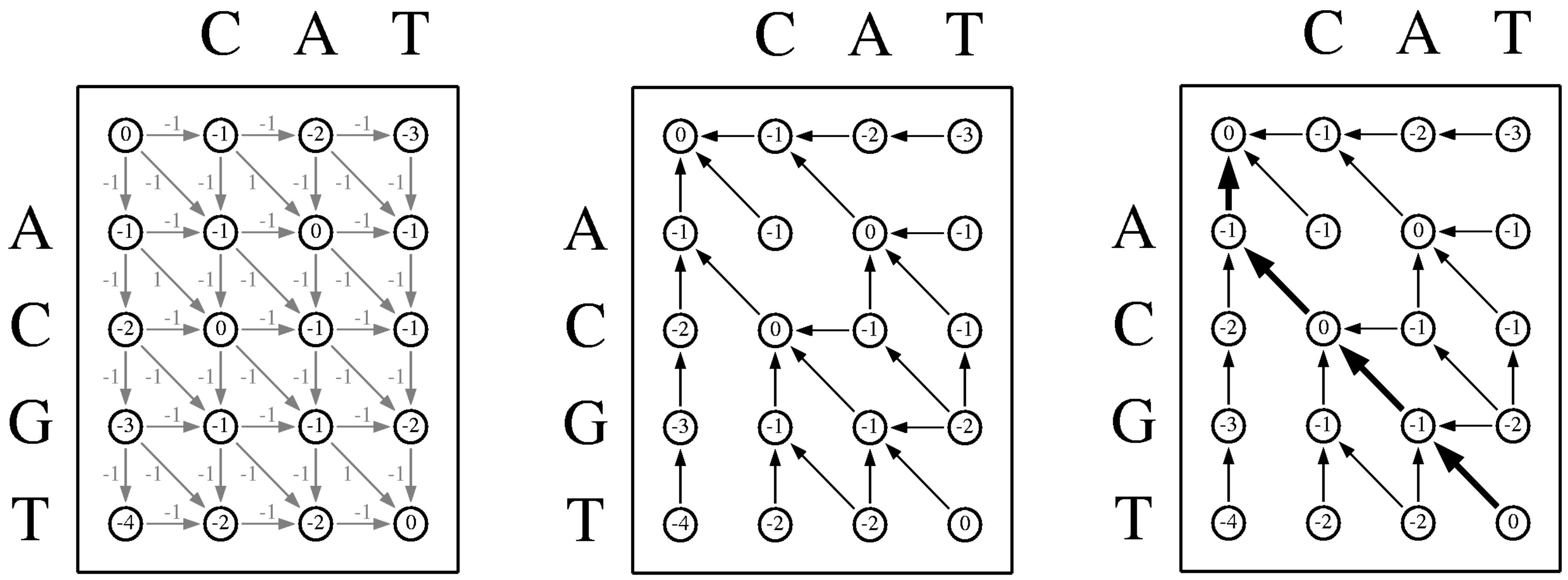


A	C	G	T
-	C	A	T

(May not be unique).

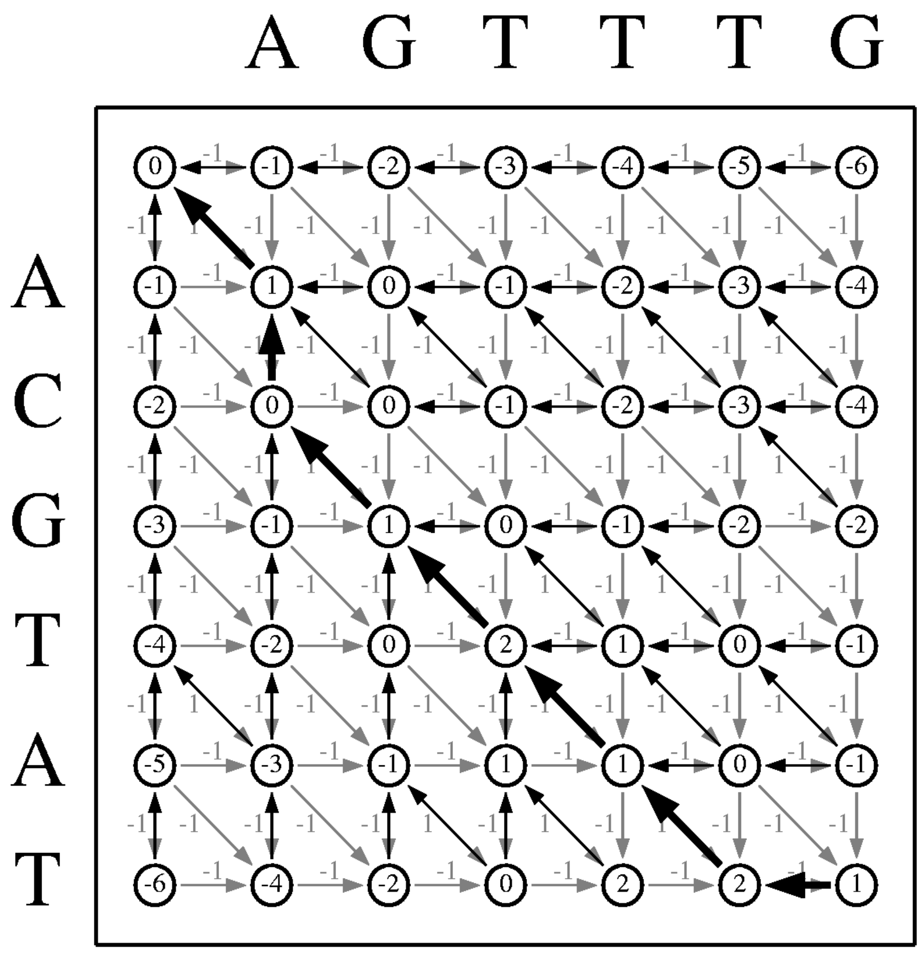
More examples

Comparing **ACGT** vs. **CAT**:



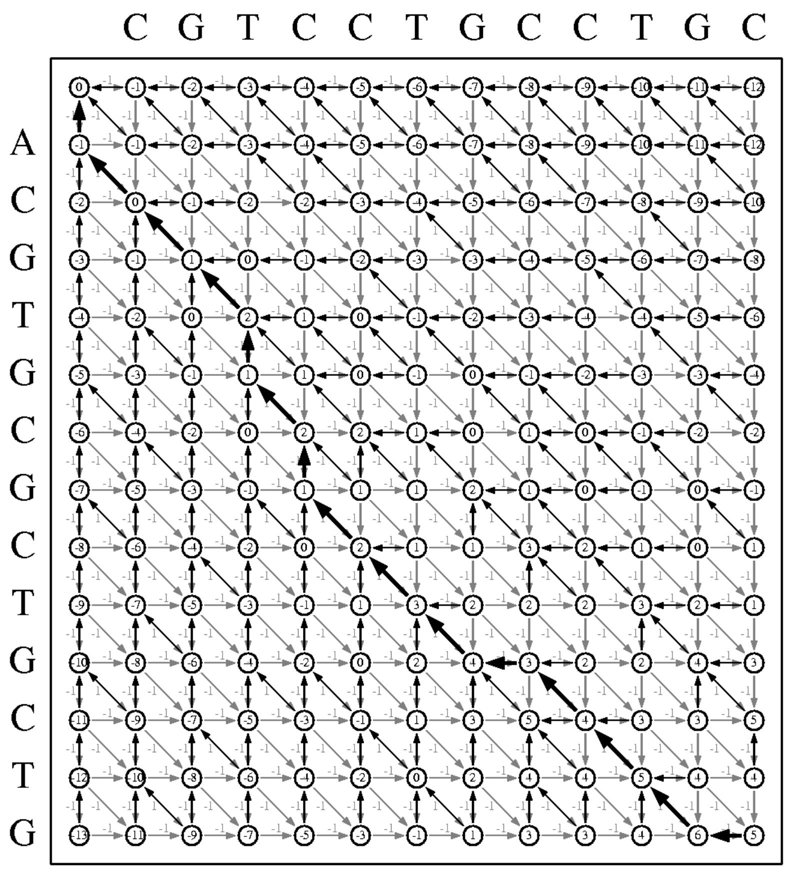
More examples

Comparing **ACGTAT** vs. **AGTTTG**:



More examples

Comparing **ACGTGCGCTGCTG** vs. **CGTCCTGCCTGC**:



Recall the trace we saw earlier:

```
A C G T G C G C T G - C T G -
↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕
- C G T - C - C T G C C T G C
```

Optimality of the algorithm

How do we know this agorithm finds an optimal alignment?

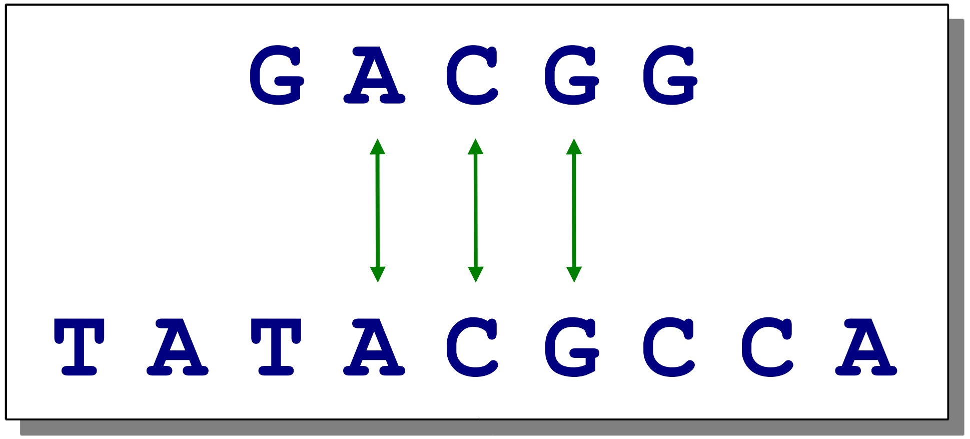
Your textbook finesses this point, but basically it hinges on:

- Bellman's *principle of optimality* for dynamic programming.
- The cost functions behaving properly (i.e., they must satisfy the *triangle inequality*).

This could be an interesting topic for a lecture or final paper ...

Local comparison

So far, we have assumed the sequences must be matched in their entirety. But this ignores interesting similarities that might be present at the subsequence level:



Fortunately, a slight modification of the original algorithm can handle this.

Local comparison

As before, say that our two sequences are:

$$s[1]s[2]s[3]...s[m] \qquad t[1]t[2]t[3]...t[n]$$

Then:

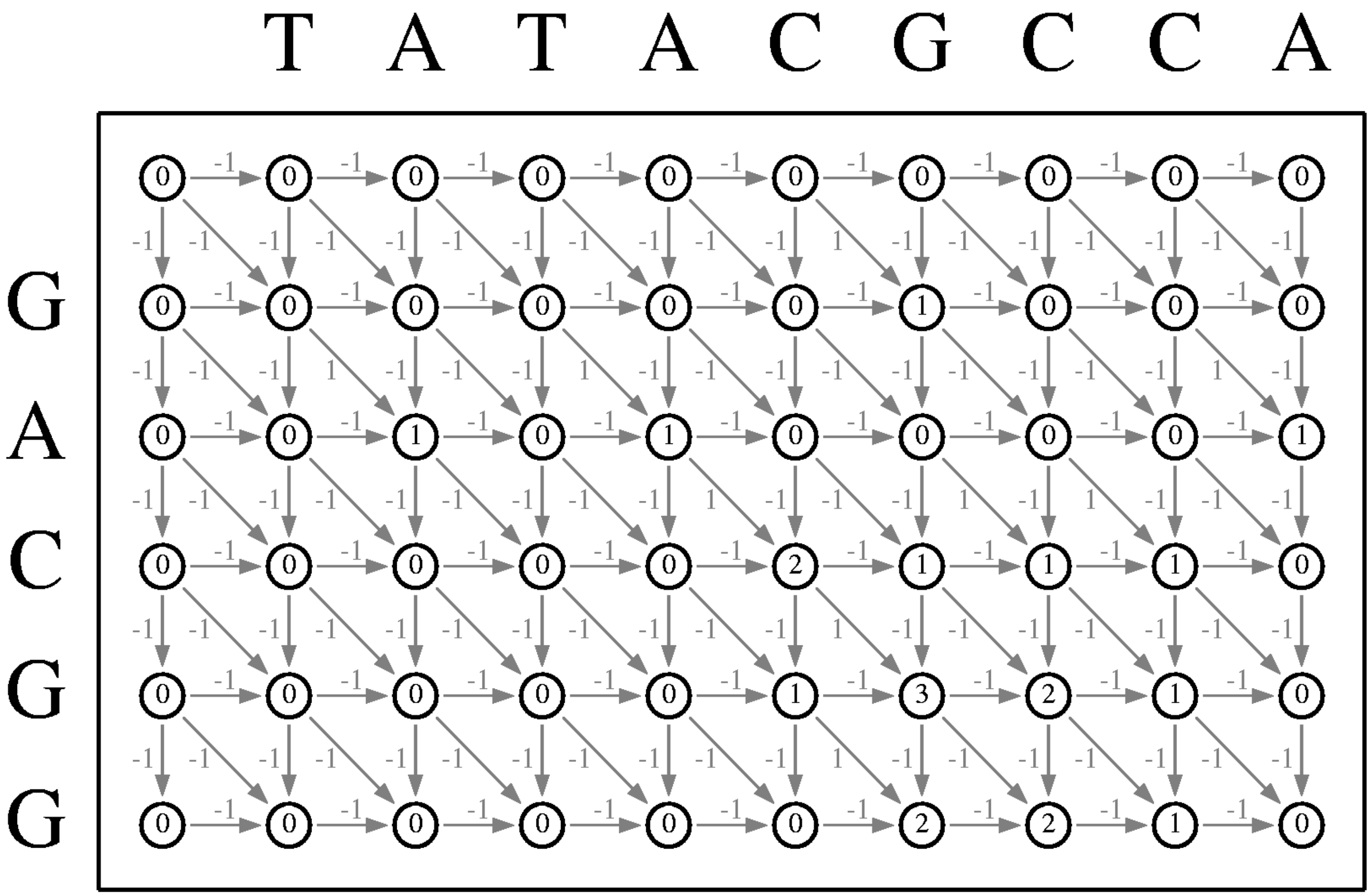
$$\begin{aligned} a[0,0] &= 0 & & \\ a[i,0] &= 0 & & 1 \le i \le m \\ a[0,j] &= 0 & & 1 \le j \le n \end{aligned}$$

And:

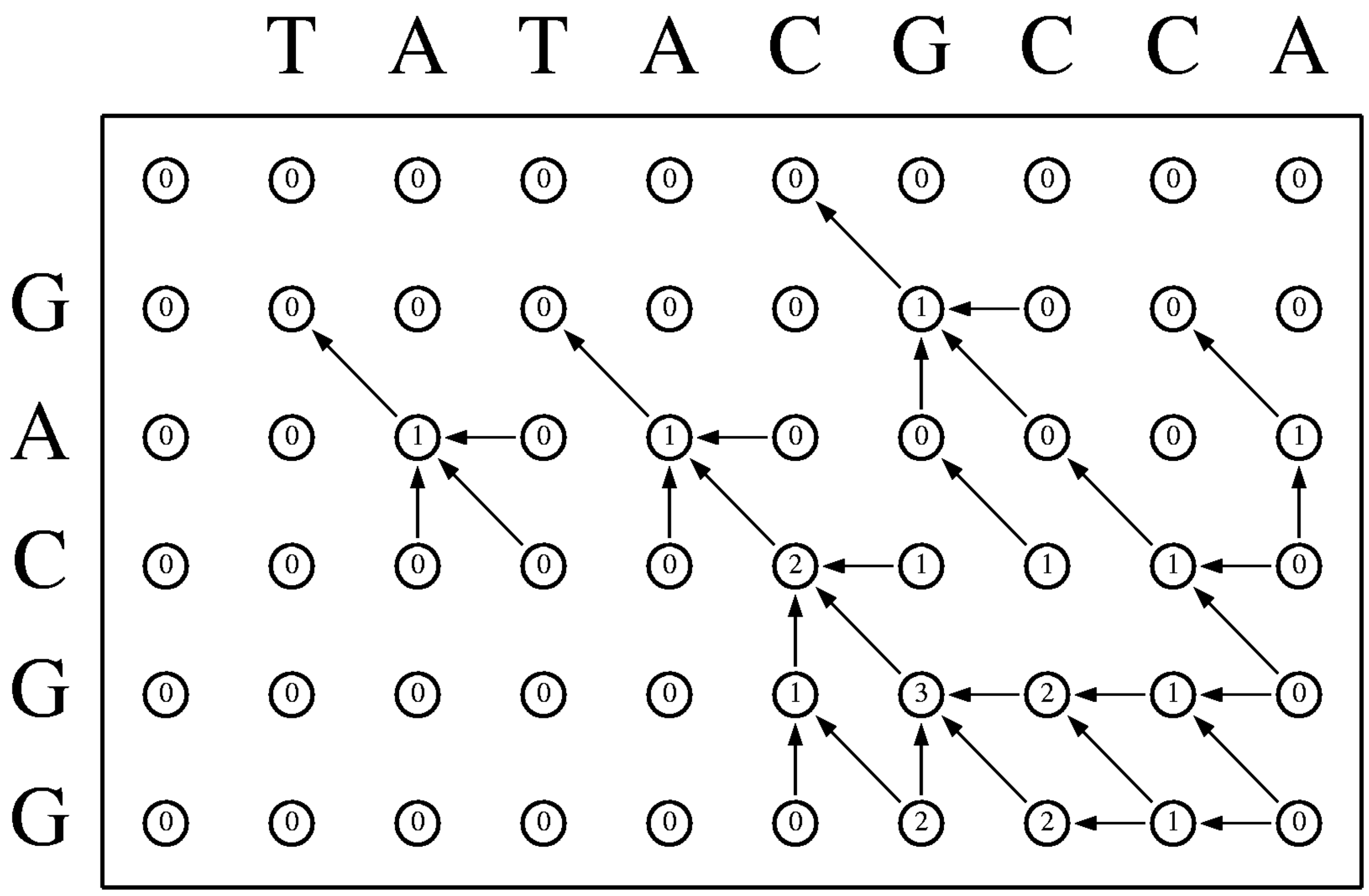
$$a[i,j] = max \begin{bmatrix} a[i\text{-}1,j] + c_{del}(s[i]) \\ a[i,j\text{-}1] + c_{ins}(t[j]) \\ a[i\text{-}1,j\text{-}1] + c_{sub}(s[i],t[j]) \\ 0 \end{bmatrix} \qquad 1 \le i \le m,\ 1 \le j \le n$$

When done, we search the matrix for the largest value.

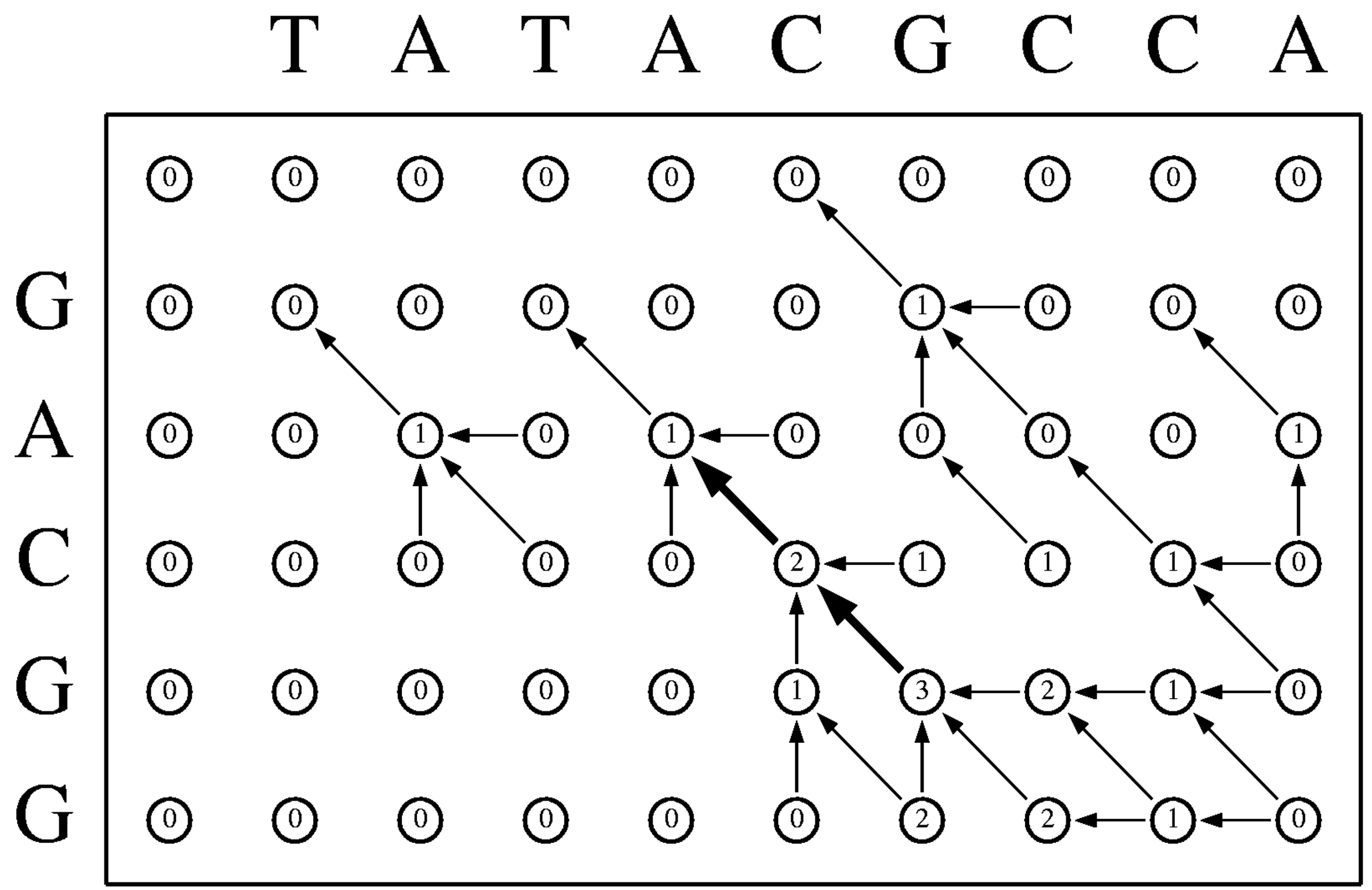
Local comparison



Local comparison



Local comparison



Wrap-up

Readings for next time:

- Section 3.3 in your textbook.
- "The emerging landscape of bioinformatics software systems" by L.S. Heath and N. Ramakrishnan, *IEEE Computer*, July 2002, pp. 41-45. (Available online or in Blackboard.)

Remember:

- Come to class prepared to discuss what you have read.
- Check Blackboard regularly for updates.