APA: Advanced Programming, Algorithms and Data Structures

Emre Güney, PhD

Master in Bioinformatics for Health Sciences
Universitat Pompeu Fabra

Lecture 12-13

November 12-14, 2019

Turnip vs. cabbage: different look, different taste

Although cabbages and turnips share a recent common ancestor, they look and taste different:

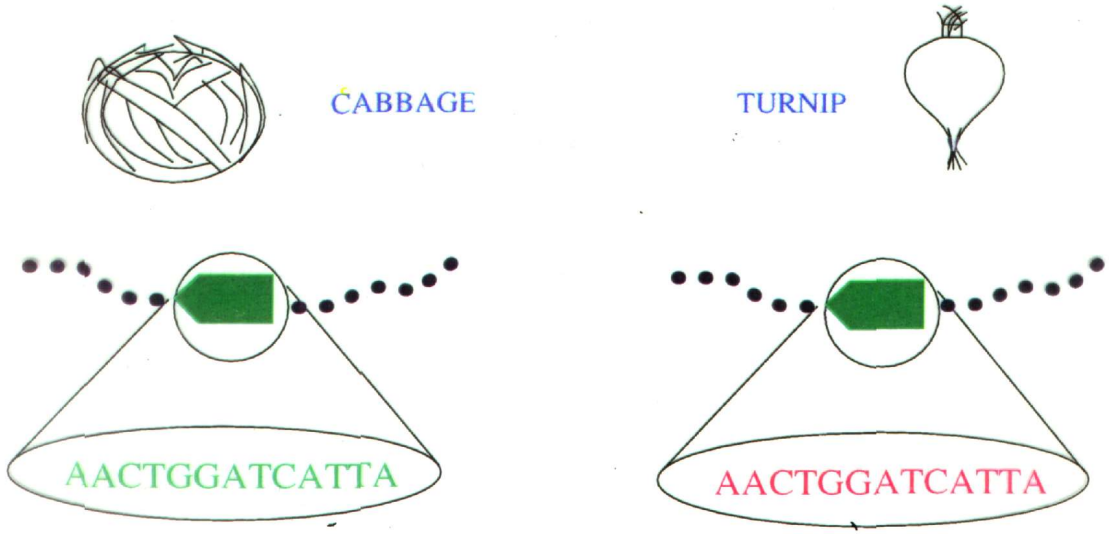
http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Turnip vs. cabbage

Comparing gene sequences yields no evolutionary information:



AACTGGATCATTA
AACTGGATCATTA

http://www.bioalgorithms.info

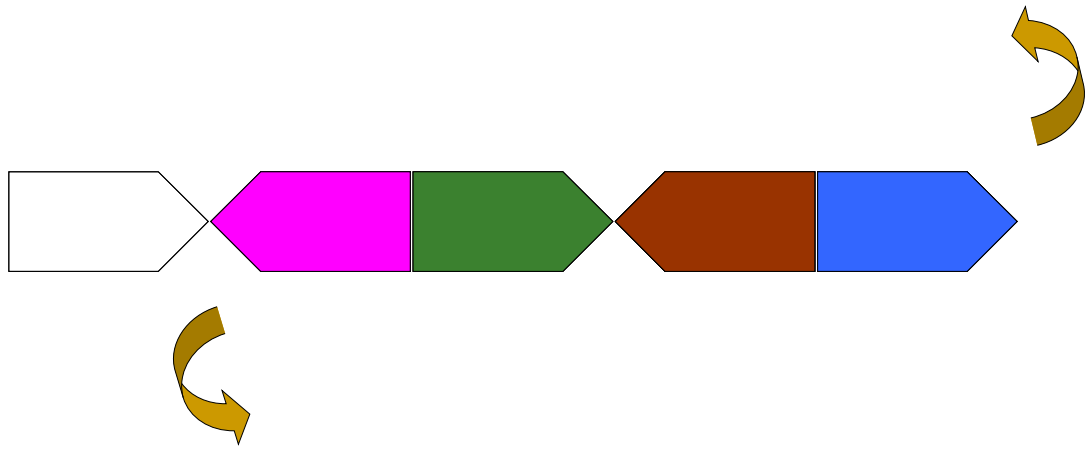
Source: Lehigh Uni. - Daniel Lopresti

Turnip vs. cabbage

Almost identical mtDNA gene sequences:

- In 1980's, Jeffrey Palmer studied evolution of plant organelles by comparing mitochondrial genomes of cabbage and turnip.
- Found 99% similarity between genes.
- Surprisingly identical gene sequences differed in gene order.
- This study helped pave way for analyzing *genome rearrangements* in molecular evolution.

Gene order comparison:

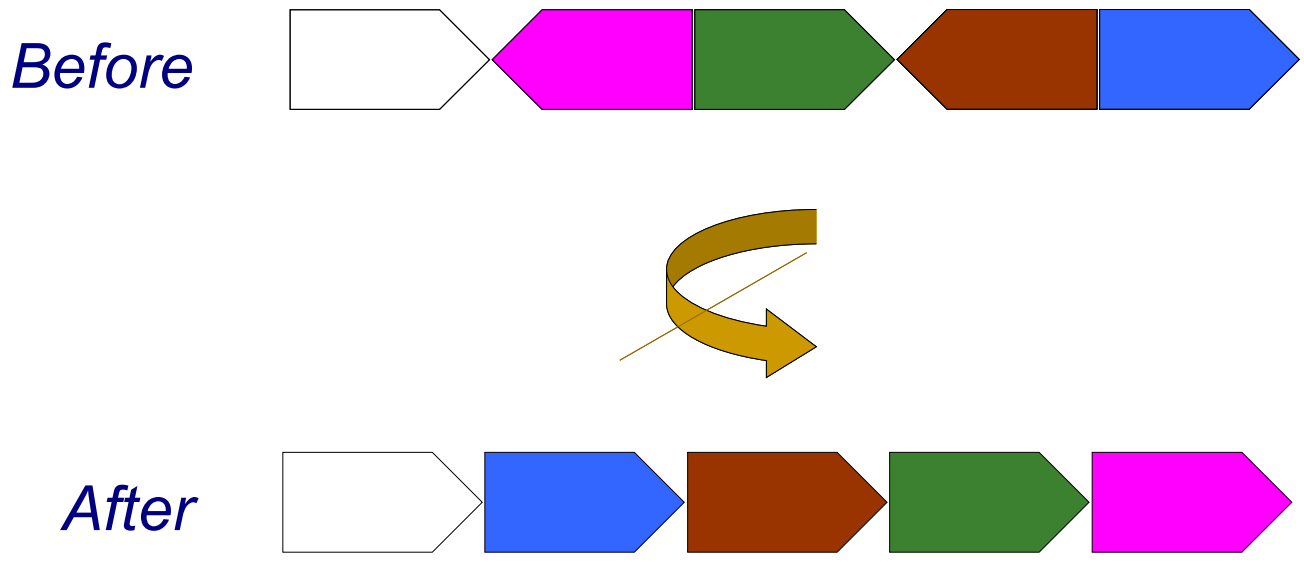


http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Turnip vs. cabbage: different mtDNA gene order

Gene order comparison:

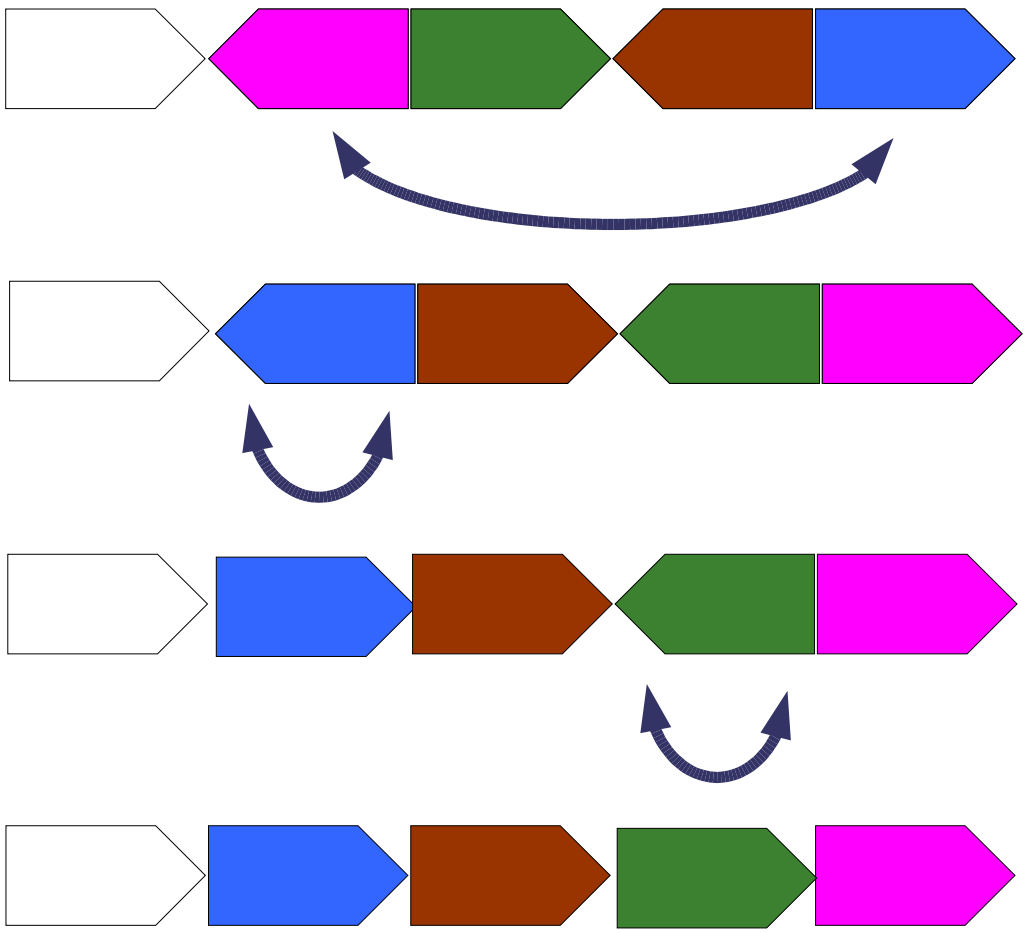


Evolution is manifested as divergence in gene order.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Turnip vs. cabbage

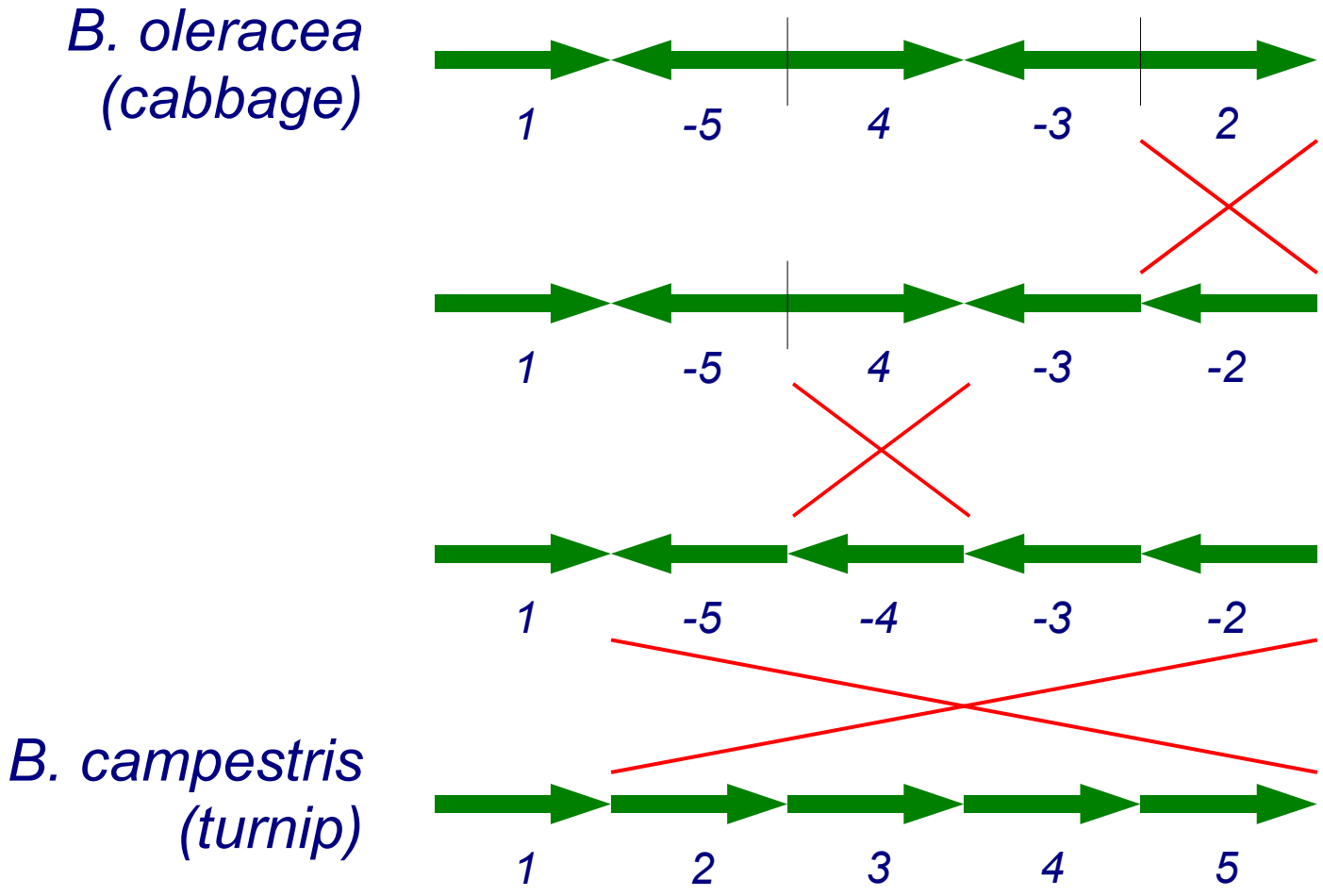


Three reversals does the trick!

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

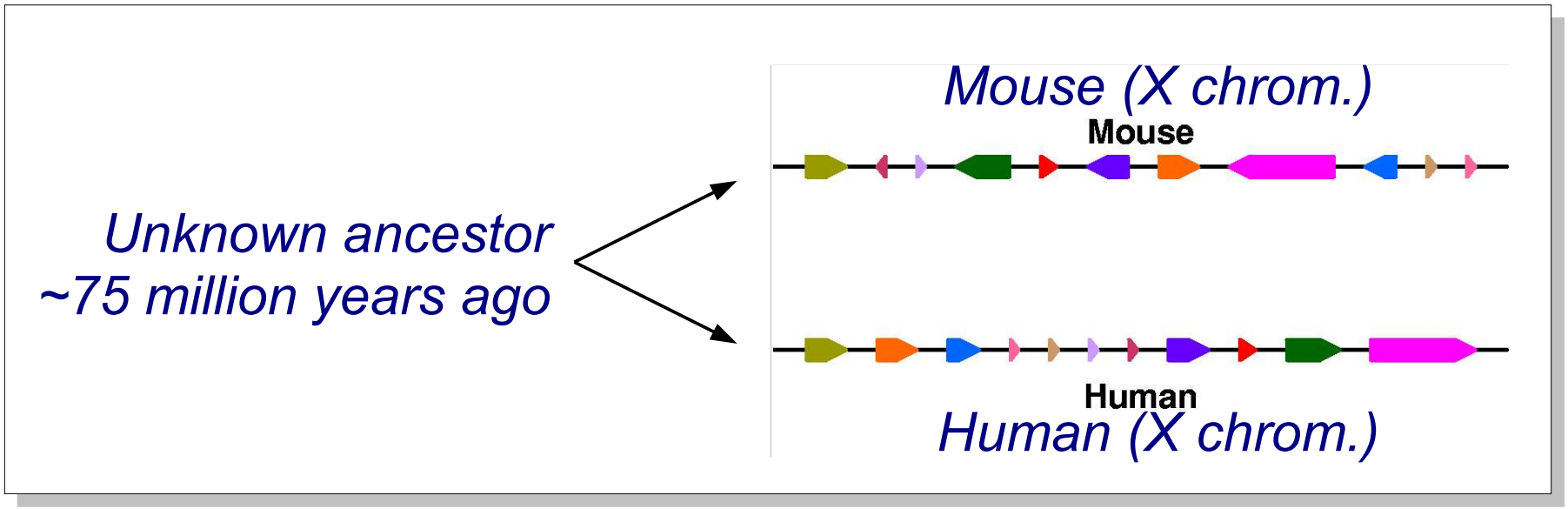
Transforming cabbage into turnip



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Genome rearrangements



- What are the similarity blocks and how to find them?
- What is architecture of ancestral genome?
- What is evolutionary scenario for transforming one genome into another?

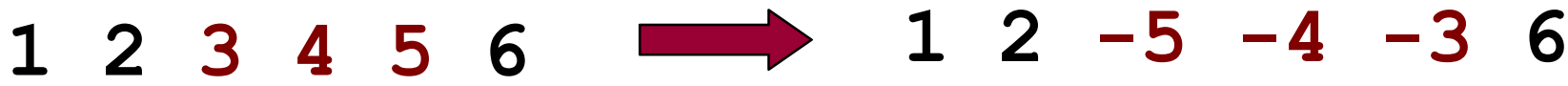
http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

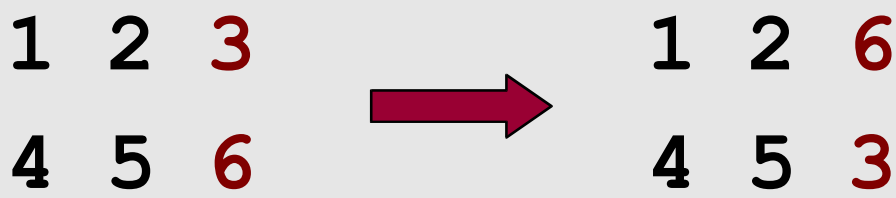
LEHIGH UNIVERSITY

Types of rearrangements

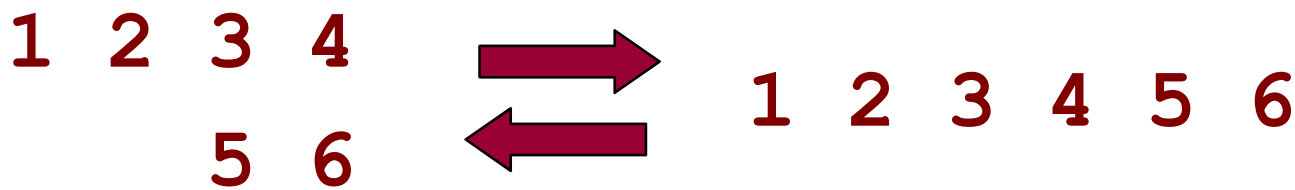
Reversal



Translocation



Fusion



Fission

http://www.bioalgorithms.info

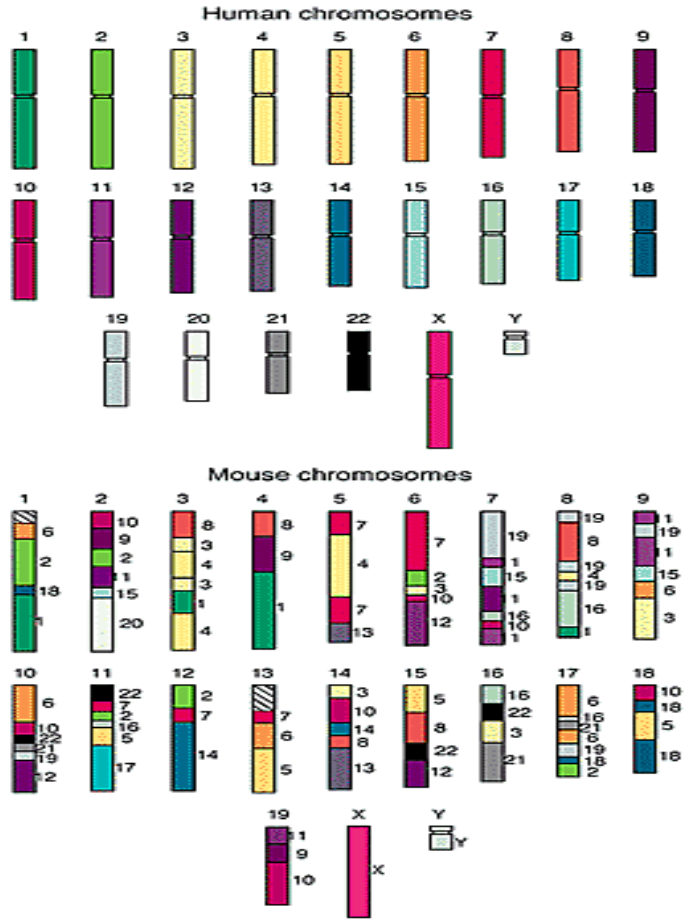
Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Comparative genomic architectures

Mouse Genome vs. Human Genome:

- Humans and mice have similar genomes, but their genes are ordered differently.
- ~245 rearrangements:
  - reversals,
  - fusions,
  - fissions,
  - translocations.



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Waardenburg's Syndrome

Mouse provides insight into human genetic disorder:

- Waardenburg's syndrome is characterized by pigmentary dysphasia.
- Gene implicated in disease linked to human Chromosome 2, but not clear where exactly it was located on chromosome.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Waardenburg's Syndrome

"Splotch" mice:

- A breed of mice (with splotch gene) had similar symptoms caused by same type of gene as in humans.
- Scientists succeeded in identifying location of gene responsible for disorder in mice.
- Finding gene in mice gives clues as to where same gene is located in humans.

http://www.bioalgorithms.info

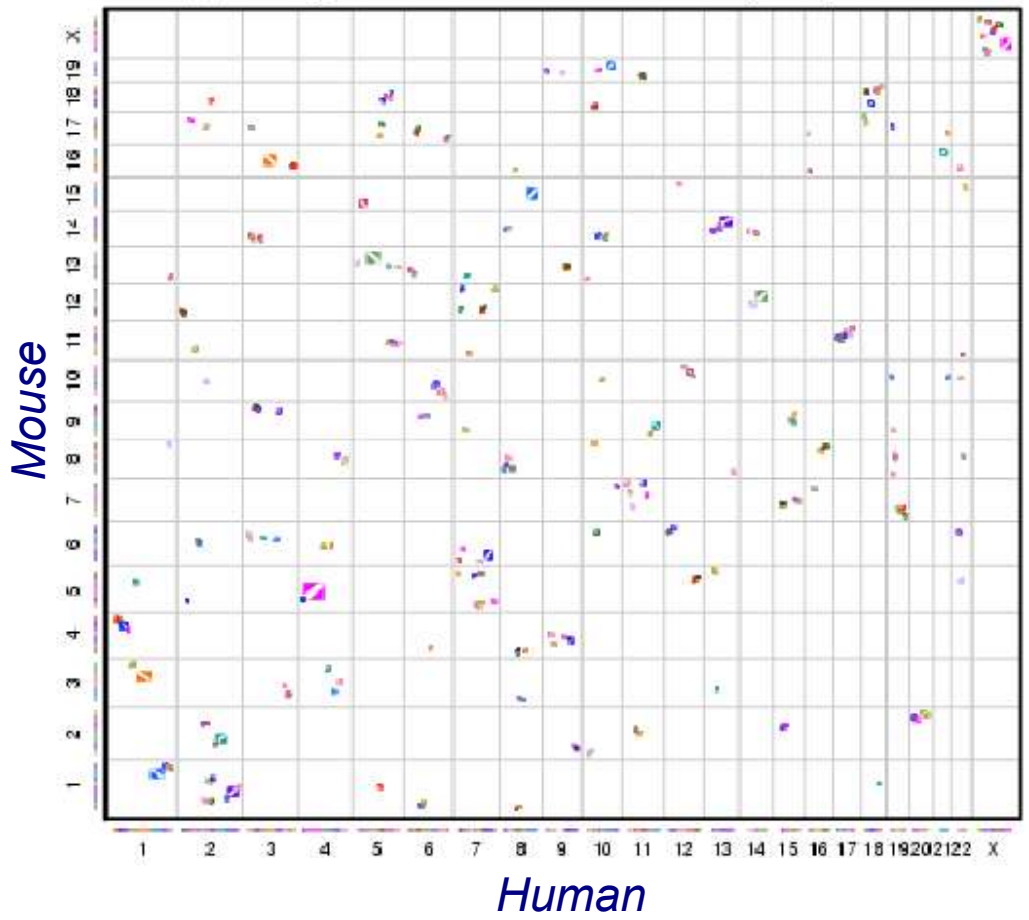
Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Comparative genomic architectures: human vs. mouse

To locate where corresponding gene is in humans, we must analyze relative architectures of human and mouse genomes.

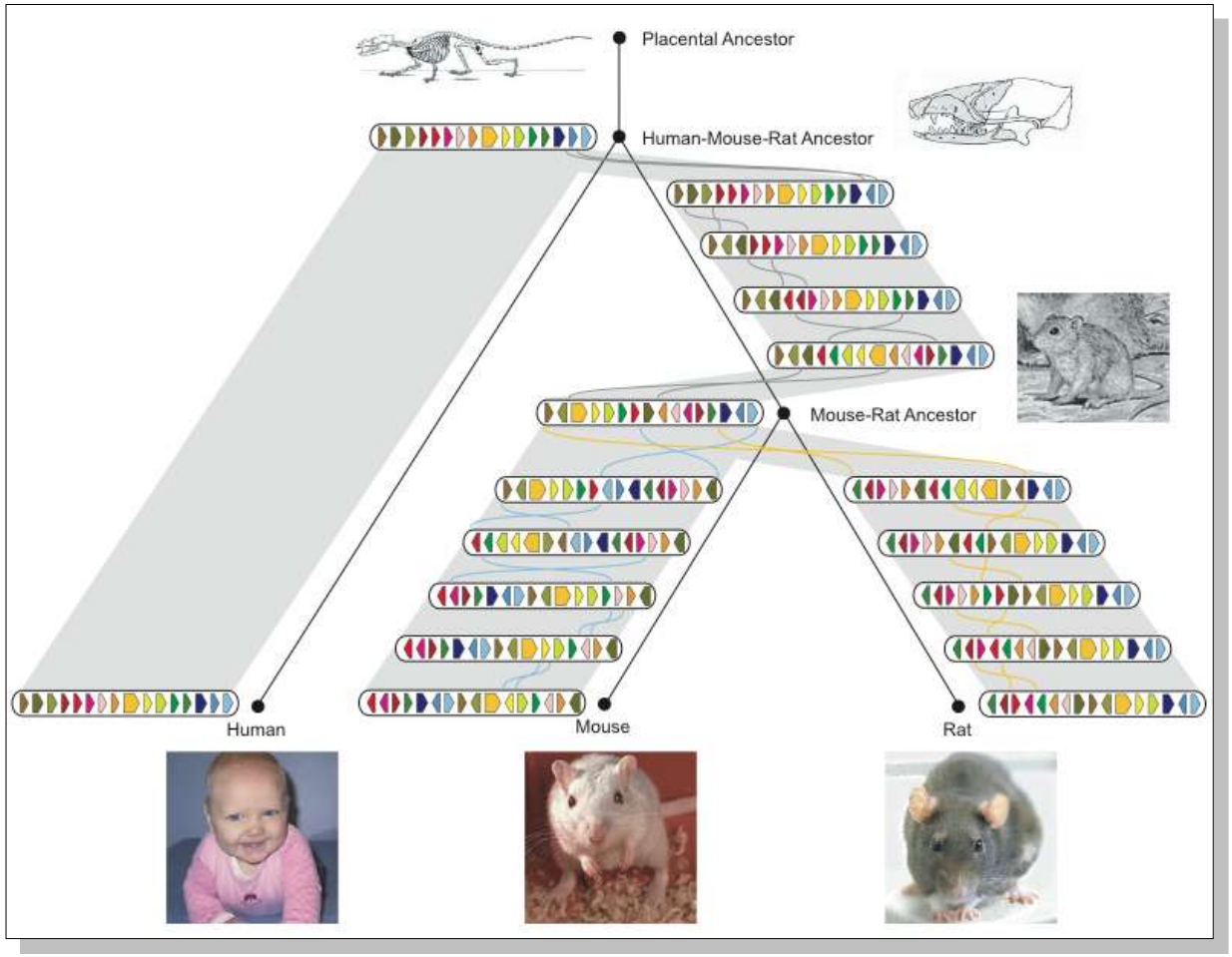
Arrangment of human and mouse synteny blocks



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

History of Chromosome X



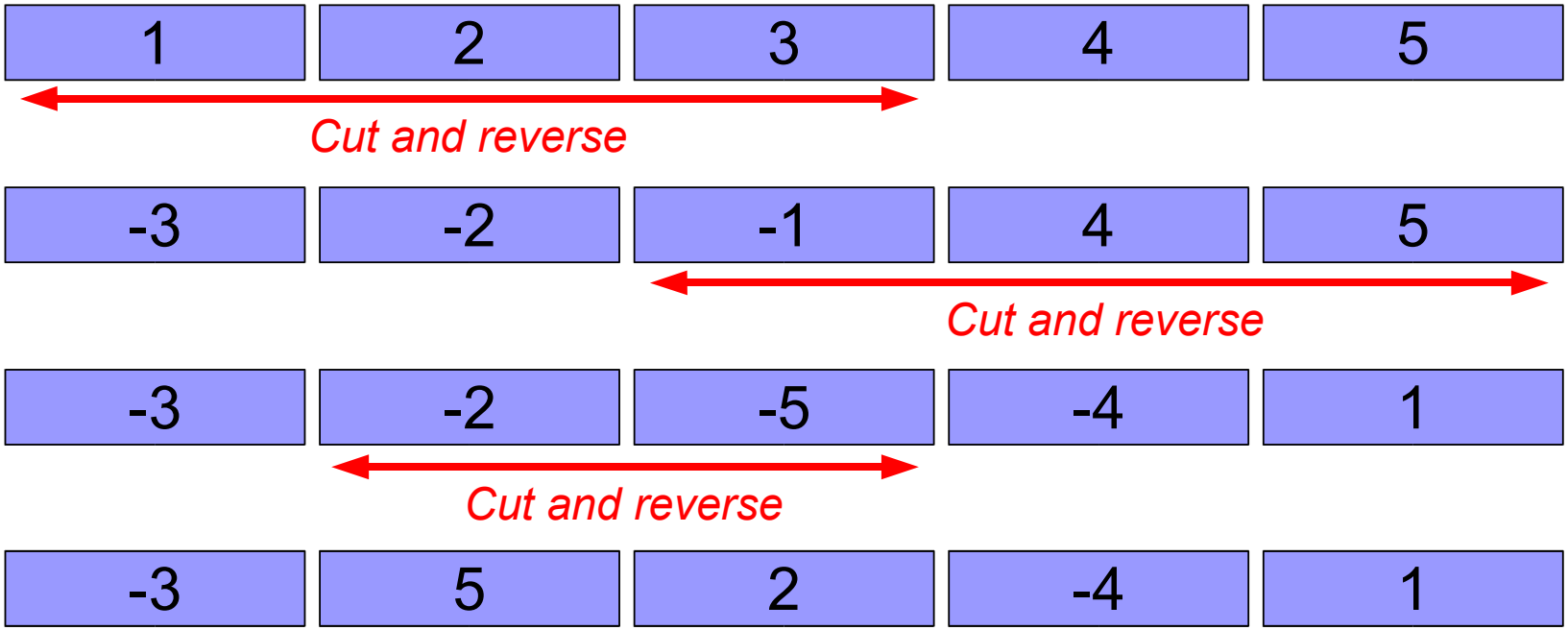
Rat Consortium, Nature, 2004

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Reversal distance

Human Chromosome X



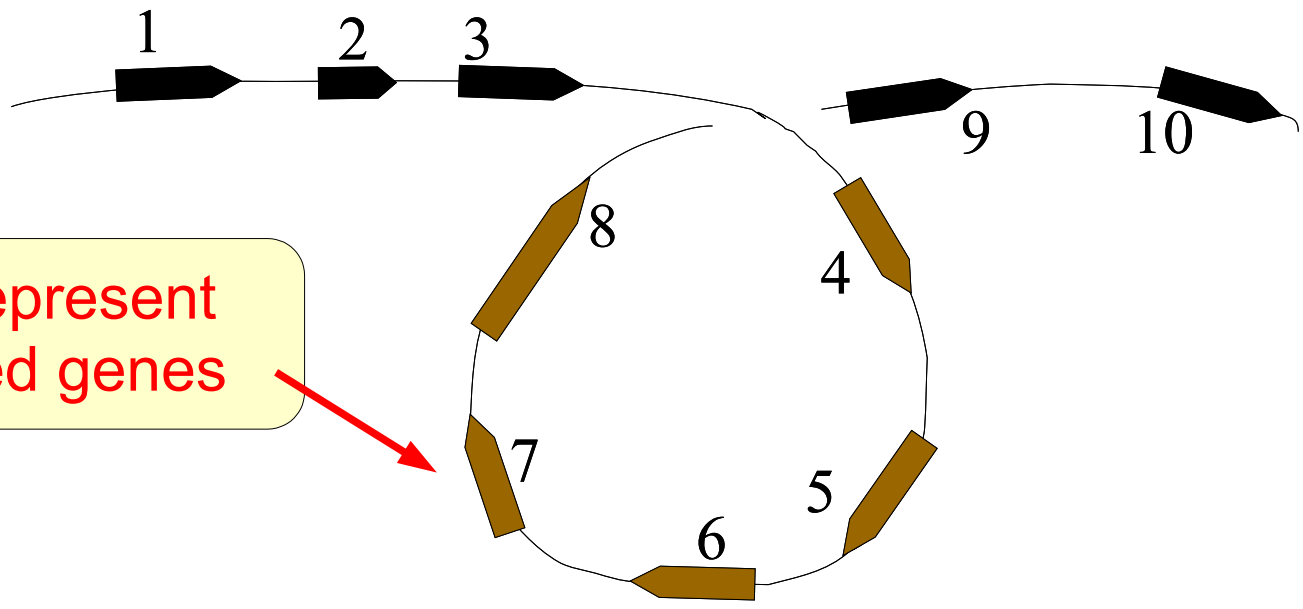
Mouse Chromosome X

Reversal distance is minimum number of such steps needed.

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Reversals



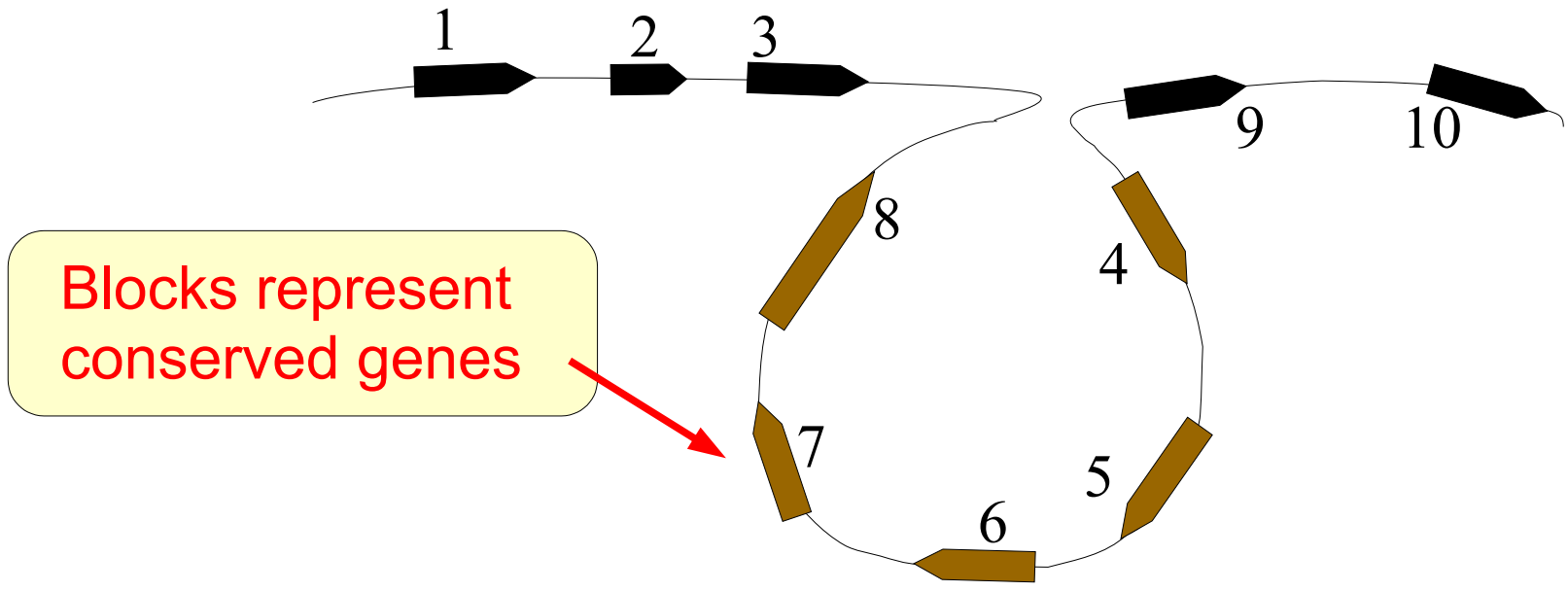
Blocks represent conserved genes

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Reversals



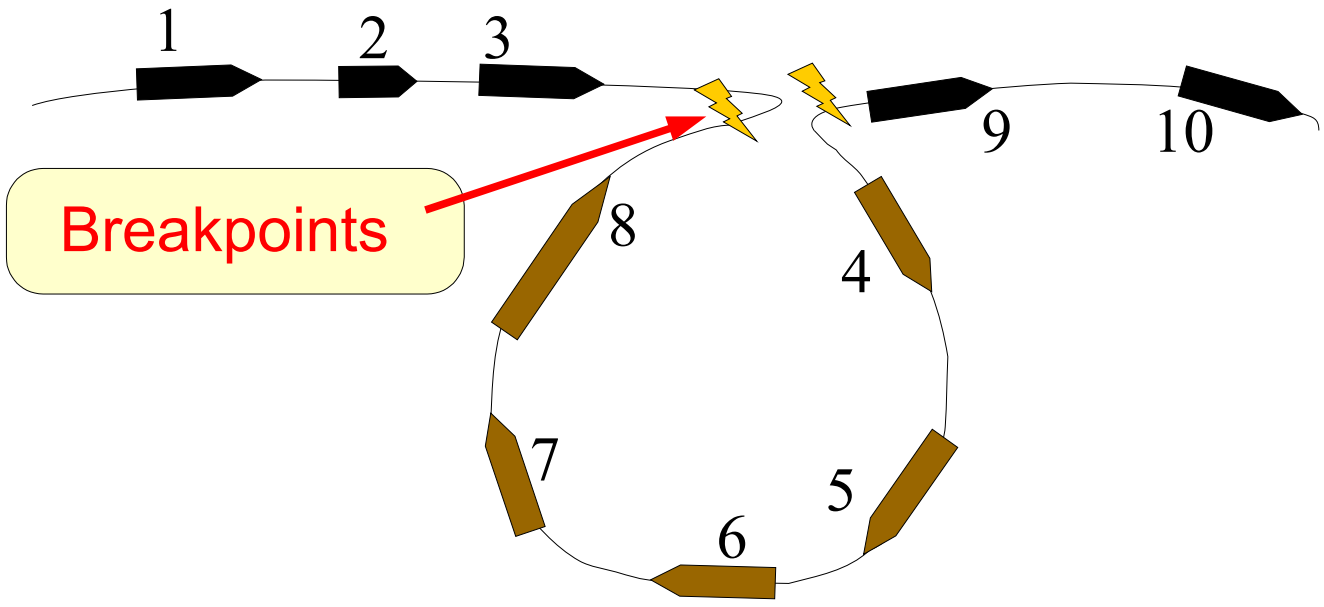
1, 2, 3, -8, -7, -6, -5, -4, 9, 10

In course of evolution or in a clinical context, blocks 1, …, 10 could be misread as 1, 2, 3, -8, -7, -6, -5, -4, 9, 10.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Reversals and breakpoints



1, 2, 3, -8, -7, -6, -5, -4, 9, 10

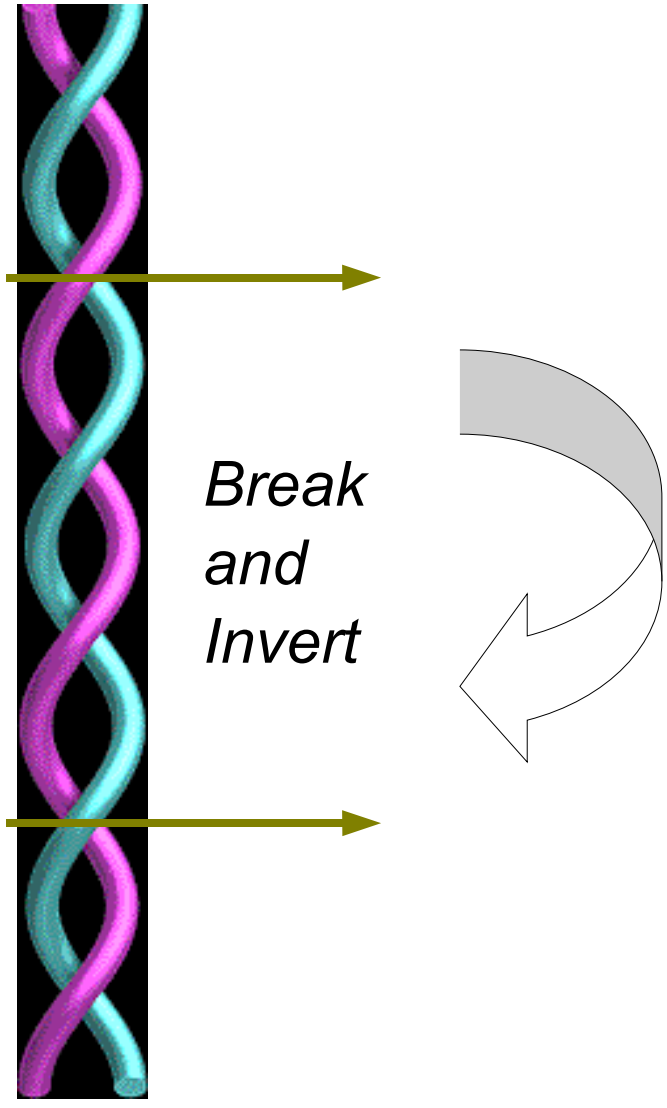
The reversal introduced two breakpoints (disruptions in order).

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Reversals: an example



http://www.bioalgorithms.info

5′ ATGCCTGTACTA 3′

3′ TACGGACATGAT 5′

5′ ATGTACAGGCTA 3′

3′ TACATGTCCGAT 5′

Source: Lehigh Uni. - Daniel Lopresti

Reversals: an example

$\pi = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$

$\rho(3,5)$ ← "Reverse subsequence from index 3 to index 5"

$\pi = 1\ 2\ 5\ 4\ 3\ 6\ 7\ 8$

$\rho(5,6)$ ← "Reverse subsequence from index 5 to index 6"

$\pi = 1\ 2\ 5\ 4\ 6\ 3\ 7\ 8$

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Reversals and gene orders

Gene order is represented by a permutation π.

$$\pi = \pi_1 \ldots \pi_{i-1}\ \pi_i\ \pi_{i+1} \ldots \pi_{j-1}\ \pi_j\ \pi_{j+1} \ldots \pi_n$$

$$\rho(i, j)$$

$$\pi = \pi_1 \ldots \pi_{i-1}\ \pi_j\ \pi_{j-1} \ldots \pi_{i+1}\ \pi_i\ \pi_{j+1} \ldots \pi_n$$

Reversal $\rho(i, j)$ reverses (flips) the elements from i to j in π.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Formal definition of the problem

The Reversal Distance Problem.

Given two permutations, find the shortest series of reversals that transforms one into another.

Input: Permutations π and σ.

Output: A series of reversals $\rho_1,\ldots, \rho_t$ transforming π into σ such that t is a minimum.

Notes:

- t is referred to as the *reversal distance* between π and σ.
- $d(\pi, \sigma)$ = smallest possible value of t, given π and σ.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

A related problem

The Sorting by Reversals Problem.

Given a permutation, find a shortest series of reversals that transforms it into the identity permutation (1 2 … *n*).

Input: Permutation π.

Output: A series of reversals $\rho_1,\ldots,\rho_t$ transforming π into the identity permutation such that *t* is a minimum.

So $d(\pi) = 3$

$$\pi = 3\ 4\ 2\ 1\ 5\ 6\ 7\ 10\ 9\ 8$$
$$4\ 3\ 2\ 1\ 5\ 6\ 7\ 10\ 9\ 8$$
$$4\ 3\ 2\ 1\ 5\ 6\ 7\ 8\ 9\ 10$$
$$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$$

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Sorting by reversals in five steps

Sometimes, we also need to consider directionality of symbols:

Step 0 (π):	2	-4	-3	5	-8	-7	-6	1
Step 1:	2	3	4	5	-8	-7	-6	1
Step 2:	2	3	4	5	6	7	8	1
Step 3:	2	3	4	5	6	7	8	-1
Step 4:	-8	-7	-6	-5	-4	-3	-2	-1
Step 5 (σ):	1	2	3	4	5	6	7	8

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Sorting by reversals in four steps

An even better solution:

Step 0 (π):	2	-4	-3	5	-8	-7	-6	1
Step 1:	2	3	4	5	-8	-7	-6	1
Step 2:	-5	-4	-3	-2	-8	-7	-6	1
Step 3:	-5	-4	-3	-2	-1	6	7	8
Step 4 (σ):	1	2	3	4	5	6	7	8

But is this the best we can do? Is this the reversal distance for this permutation? Or can it be sorted in 3 steps?

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

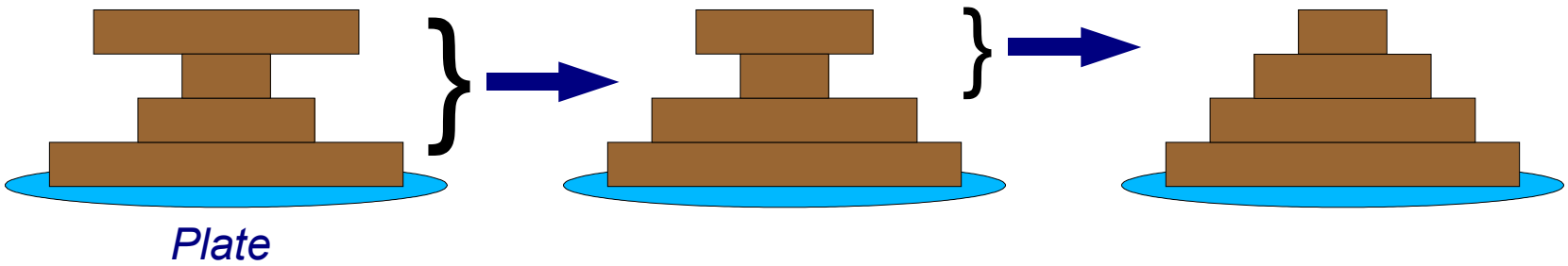
Another related problem

The Pancake Flipping Problem.

Given a stack of *n* pancakes of different sizes, what is the minimum number of flips to rearrange them into perfect stack?

Input: Permutation π.

Output: A series of *t* prefix reversals $\rho_1,\ldots,\rho_t$ transforming π into the identity permutation such that *t* is a minimum.

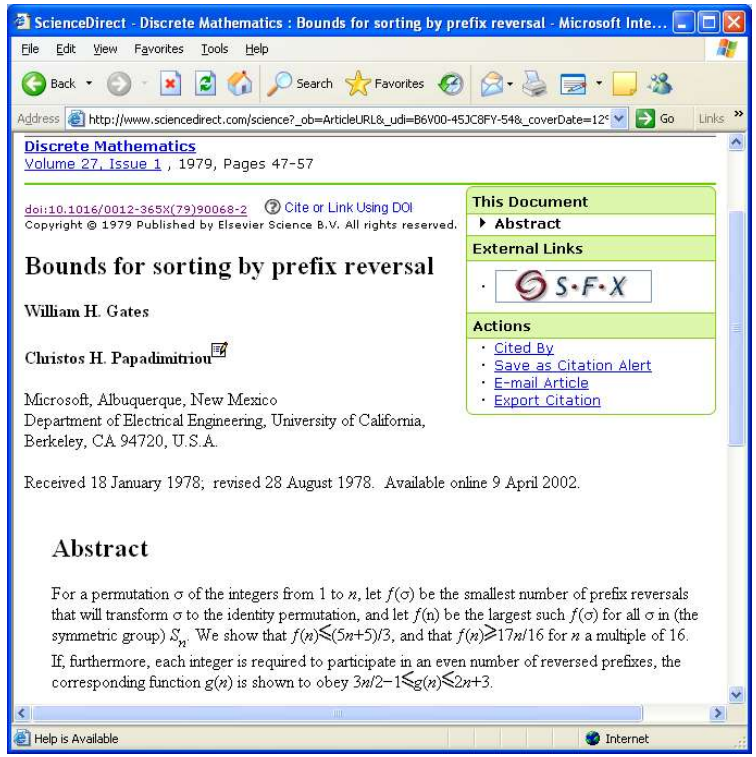


http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Greedy Algorithm for the Pancake Flipping Problem

- Greedy approach: two prefix reversals at most to place a pancake in its right position; at most $2n - 2$ steps in total.
- William Gates and Christos Papadimitriou showed problem can be solved by at most $5/3\ (n + 1)$ prefix reversals.


ScienceDirect - Discrete Mathematics : Bounds for sorting by prefix reversal - Microsoft Inte...

File Edit View Favorites Tools Help

Back Search Favorites

Address http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V00-45JC8FY-54&_coverDate=12

Go Links

Discrete Mathematics
Volume 27, Issue 1 , 1979, Pages 47-57

doi:10.1016/0012-365X(79)90068-2 Cite or Link Using DOI

Copyright © 1979 Published by Elsevier Science B.V. All rights reserved.

This Document
Abstract
External Links
S·F·X
Actions
Cited By
Save as Citation Alert
E-mail Article
Export Citation

Bounds for sorting by prefix reversal

William H. Gates

Christos H. Papadimitriou

Microsoft, Albuquerque, New Mexico
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978; revised 28 August 1978. Available online 9 April 2002.

Abstract

For a permutation σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n. We show that $f(n) \leqslant (5n+5)/3$, and that $f(n) \geqslant 17n/16$ for n a multiple of 16.

If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leqslant g(n) \leqslant 2n+3$.

Help is Available Internet

Yes, that Bill Gates ...

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Greedy Algorithm for the Sorting by Reversals Problem

Observations:

- If sorting permutation π = **1 2 3 6 4 5**, first three elements are already in order so it makes no sense to break them.
- Denote length of already-sorted prefix as prefix(π).

  E.g., prefix(π) = 3
- This gives us an idea for a greedy algorithm: attempt to increase prefix(π) at every step.

For example: **1 2 3 6 4 5**

Put 4 in place → **1 2 3 4 6 5**

Put 5 in place → **1 2 3 4 5 6**

Number of steps to sort permutation of length *n* is at most (*n* – 1)

Source: Lehigh Uni. - Daniel Lopresti

Pseudocode for Greedy Algorithm

```
SimpleReversalSort(π)
for  i ← 1 to n – 1
    j ← position of element i in π (i.e., π_j = i)
    if  j ≠ i
        π ← π • ρ(i, j)
        output π
    if π is the identity permutation
        return
```

Source: Lehigh Uni. - Daniel Lopresti

Analyzing SimpleReversalSort

SimpleReversalSort does not guarantee the smallest number of reversals and takes five steps on π = `6 1 2 3 4 5`:

Step 0:	**6**	**1**	2	3	4	5
Step 1:	1	**6**	**2**	3	4	5
Step 2:	1	2	**6**	**3**	4	5
Step 3:	1	2	3	**6**	**4**	5
Step 4:	1	2	3	4	**6**	**5**
Step 5:	1	2	3	4	5	6

Source: Lehigh Uni. - Daniel Lopresti

Analyzing SimpleReversalSort (continued)

But it can be sorted in two steps:

Step 0:	6	1	2	3	4	5
Step 1:	5	4	3	2	1	6
Step 2:	1	2	3	4	5	6

Hence, SimpleReversalSort(π) is not optimal.

We don't know optimal algorithms for many important problems. Indeed, such an algorithm may not exist in some cases.

When we don't have an optimal algorithm, we can use an *approximation algorithm* instead.

Source: Lehigh Uni. - Daniel Lopresti

Approximation algorithms

The approximation ratio of algorithm A on input π is:

$$A(\pi) / OPT(\pi)$$

where

$A(\pi)$	is solution produced by algorithm A
$OPT(\pi)$	is optimal solution to problem

The approximation ratio (*performance guarantee*) of algorithm A is the largest approximation ratio for all inputs of size *n*.

- For algorithm that minimizes objective function:

$$\max_{|\pi| = n} A(\pi) / OPT(\pi)$$

- For algorithm that maximizes objective function:

$$\min_{|\pi| = n} A(\pi) / OPT(\pi)$$

Source: Lehigh Uni. - Daniel Lopresti

Adjacencies and breakpoints

Consider the following permutation:

$$\pi = \pi_1 \pi_2 \pi_3 \ldots \pi_{n-1} \pi_n$$

A pair of elements π_i and π_{i+1} are *adjacent* if

$$\pi_{i+1} = \pi_i \pm 1$$

For example:

$$\pi = 1 \quad 9 \quad \overline{3 \quad 4} \quad \overline{7 \quad 8} \quad 2 \quad \overline{6 \quad 5}$$

Here (3, 4), (7, 8), and (6, 5) are adjacent pairs.

Source: Lehigh Uni. - Daniel Lopresti

Breakpoints: an example

There is a *breakpoint* between any adjacent elements that are non-consecutive:

π = 1 | 9 | 3 4 | 7 8 | 2 | 6 5

Pairs (1, 9), (9, 3), (4, 7), (8, 2) and (2, 6) are breakpoints.

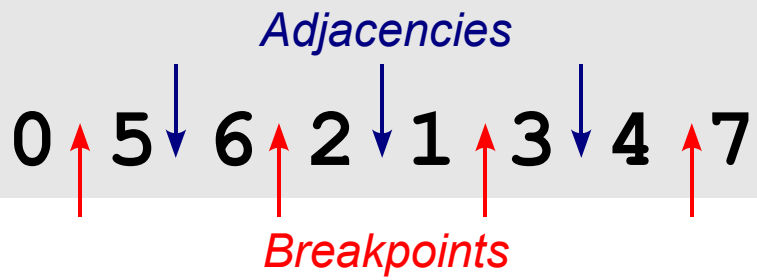
Define $b(\pi)$ as the number of breakpoints in permutation π.

Adjacency = pair of adjacent elements that are consecutive.

Breakpoint = pair of adjacent elements that are not consecutive.

π = 5 6 2 1 3 4

Extend π with 0 and 7 →



Source: Lehigh Uni. - Daniel Lopresti

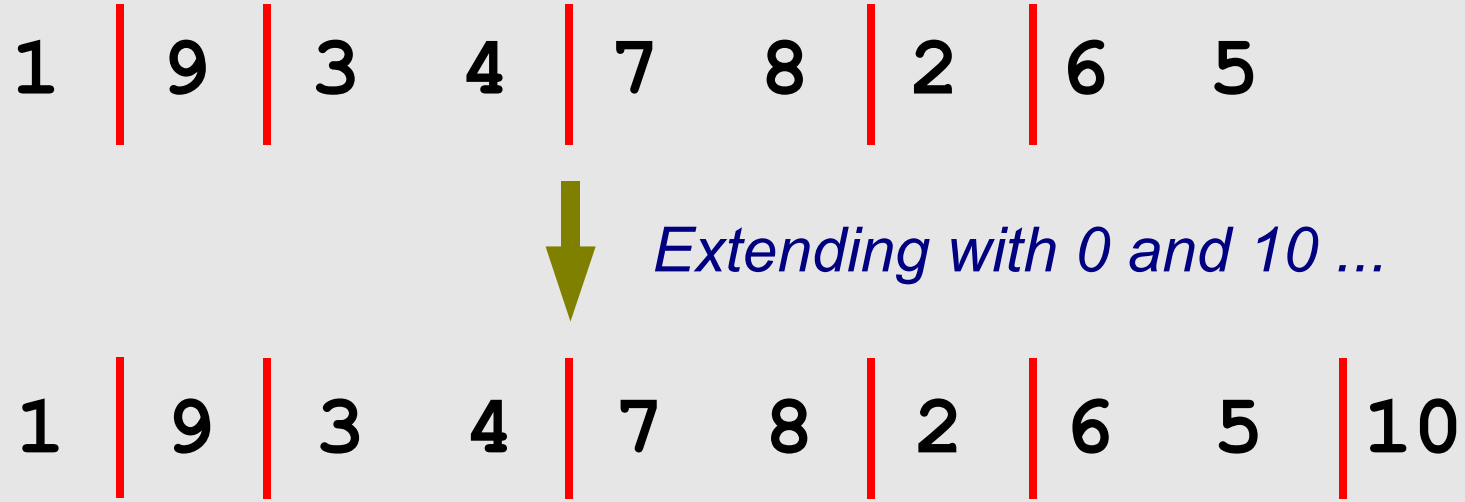
Extending permutations

Given the following permutation:

$$\pi = \pi_1 \pi_2 \pi_3 \ldots \pi_{n-1} \pi_n$$

We put two elements $\pi_0 = 0$ and $\pi_{n+1} = n + 1$ at the ends of π.

For example:



Note: a new breakpoint was created after extending!

Source: Lehigh Uni. - Daniel Lopresti

Reversal distance and breakpoints

Each reversal eliminates at most two breakpoints.

π = 2 3 1 4 6 5

0 | 2 3 | 1 | 4 | 6 5 | 7 $b(\pi) = 5$

0 1 | 3 2 | 4 | 6 5 | 7 $b(\pi) = 4$

0 1 2 3 4 | 6 5 | 7 $b(\pi) = 2$

0 1 2 3 4 5 6 7 $b(\pi) = 0$

Source: Lehigh Uni. - Daniel Lopresti

Reversal distance and breakpoints

Each reversal eliminates at most two breakpoints.

This implies:

reversal distance ≥ #breakpoints / 2

0	2	3	1	4	6	5	7	$b(\pi) = 5$
0	1	3	2	4	6	5	7	$b(\pi) = 4$
0	1	2	3	4	6	5	7	$b(\pi) = 2$
0	1	2	3	4	5	6	7	$b(\pi) = 0$

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

A better greedy algorithm for Sorting by Reversals

```
BreakPointReversalSort(π)
while b(π) > 0
    Among all possible reversals,
        choose reversal ρ minimizing b(π • ρ)
    π ← π • ρ(i, j)
    output π
return
```

Problem: this algorithm may work forever.

Source: Lehigh Uni. - Daniel Lopresti

Strips

Strip: the interval between two consecutive breakpoints.

- *Decreasing strip*: elements in decreasing order (e.g., **6 5**).
- *Increasing strip*: elements in increasing order (e.g., **7 8**).

A single-element strip can be declared either increasing or decreasing. We will choose to declare them as decreasing, with exception of the strips involving 0 and n + 1.

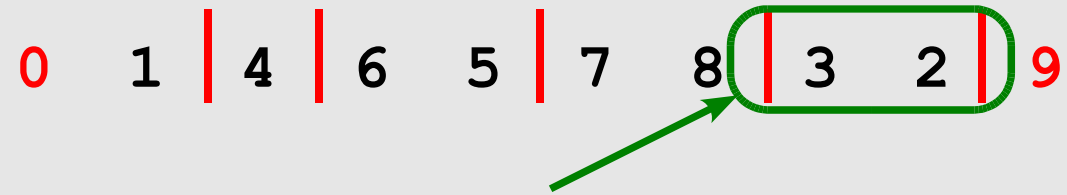
Source: Lehigh Uni. - Daniel Lopresti

Reducing number of breakpoints

Theorem 1:

If permutation π contains at least one decreasing strip, then there exists a reversal ρ which decreases the number of breakpoints (i.e., $b(\pi \cdot \rho) < b(\pi)$).

For π = 1 4 6 5 7 8 3 2



$b(\pi) = 5$

(a) Choose decreasing strip with smallest element *k* in π (*k* = 2 in this case).

Source: Lehigh Uni. - Daniel Lopresti

Things to consider

For π = 1 4 6 5 7 8 3 2

0 1 | 4 | 6 5 | 7 8 | 3 2 | 9 $b(\pi) = 5$

(a) Choose decreasing strip with smallest element k in π (k = 2 in this case).

For π = 1 4 6 5 7 8 3 2

0 1 | 4 | 6 5 | 7 8 | 3 2 | 9 $b(\pi) = 5$

(a) Choose decreasing strip with smallest element k in π (k = 2 in this case).

(b) Find k – 1 in the permutation.

Source: Lehigh Uni. - Daniel Lopresti

Things to consider

For π = 1 4 6 5 7 8 3 2

0 1 | 4 | 6 5 | 7 8 | 3 2 | 9 $b(\pi) = 5$

(a) Choose decreasing strip with smallest element k in π ($k = 2$ in this case).

(b) Find $k - 1$ in the permutation.

(c) Reverse the segment between k and $k - 1$.

0 1 | 4 | 6 5 | 7 8 | 3 2 | 9 $b(\pi) = 5$

0 1 2 3 | 8 7 | 5 6 | 4 | 9 $b(\pi) = 4$

Source: Lehigh Uni. - Daniel Lopresti

Reducing number of breakpoints again

- If there is no decreasing strip, there may be no reversal ρ that reduces number of breakpoints (i.e. $b(\pi \cdot \rho) \geq b(\pi)$ for any reversal ρ).
- By reversing an increasing strip (# of breakpoints stays unchanged), we will create a decreasing strip at next step. Then number of breakpoints will be reduced in next step (by Theorem 1).

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Things to consider

There are no decreasing strips in π for:

π = 0 1 2 | 5 6 7 | 3 4 | 8 $b(\pi) = 3$

$\pi \bullet \rho(6, 7)$ = 0 1 2 | 5 6 7 | 4 3 | 8 $b(\pi) = 3$

- $\rho(6, 7)$ does not change # of breakpoints.
- $\rho(6, 7)$ creates a decreasing strip, thus guaranteeing next step will decrease # of breakpoints.

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Improved BreakpointReversalSort

ImprovedBreakpointReversalSort(π)
while $b(\pi) > 0$
    if π has a decreasing strip
        among all possible reversals, choose reversal ρ
            that minimizes $b(\pi \cdot \rho)$
    else
        choose a reversal ρ that flips an increasing strip in π
    $\pi \leftarrow \pi \cdot \rho$
output π
return

Source: Lehigh Uni. - Daniel Lopresti

Improved BreakpointReversalSort: performance guarantee

ImprovedBreakPointReversalSort is an approximation algorithm with a performance guarantee of at most 4.

- Eliminates at least one breakpoint in every two steps $\Rightarrow$ at most $2b(\pi)$ steps.
- Approximation ratio is $2b(\pi) / d(\pi)$.
- Optimal algorithm eliminates at most 2 breakpoints in every step, so $d(\pi) \geq b(\pi) / 2$.
- Performance guarantee:

$$[\, 2b(\pi) / d(\pi) \,] \leq [\, 2b(\pi) / (b(\pi) / 2) \,] = 4$$

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Sequence comparison and alignment

Consider:

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?

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

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.

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

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.

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

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*.

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Sequence concepts

More basic sequence concepts:

substring	sequence of consecutive symbols appearing in *s* (**ACG** is not 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].

Source: Lehigh Uni. - Daniel Lopresti

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**.

Source: Lehigh Uni. - Daniel Lopresti

DNA sequence comparison

First successes:

- Finding sequence similarities with genes of known function is a common approach to infer function of a newly-sequenced gene.
- In 1984, Russell Doolittle and colleagues found similarities between cancer-causing gene and normal growth factor (PDGF) gene.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Cystic Fibrosis

- Cystic fibrosis (CF) is chronic and often fatal genetic disease of body's mucus glands (abnormally high level of mucus). Primarily affects respiratory systems in children.
- Mucus is slimy material that coats many epithelial surfaces and is secreted into fluids such as saliva.

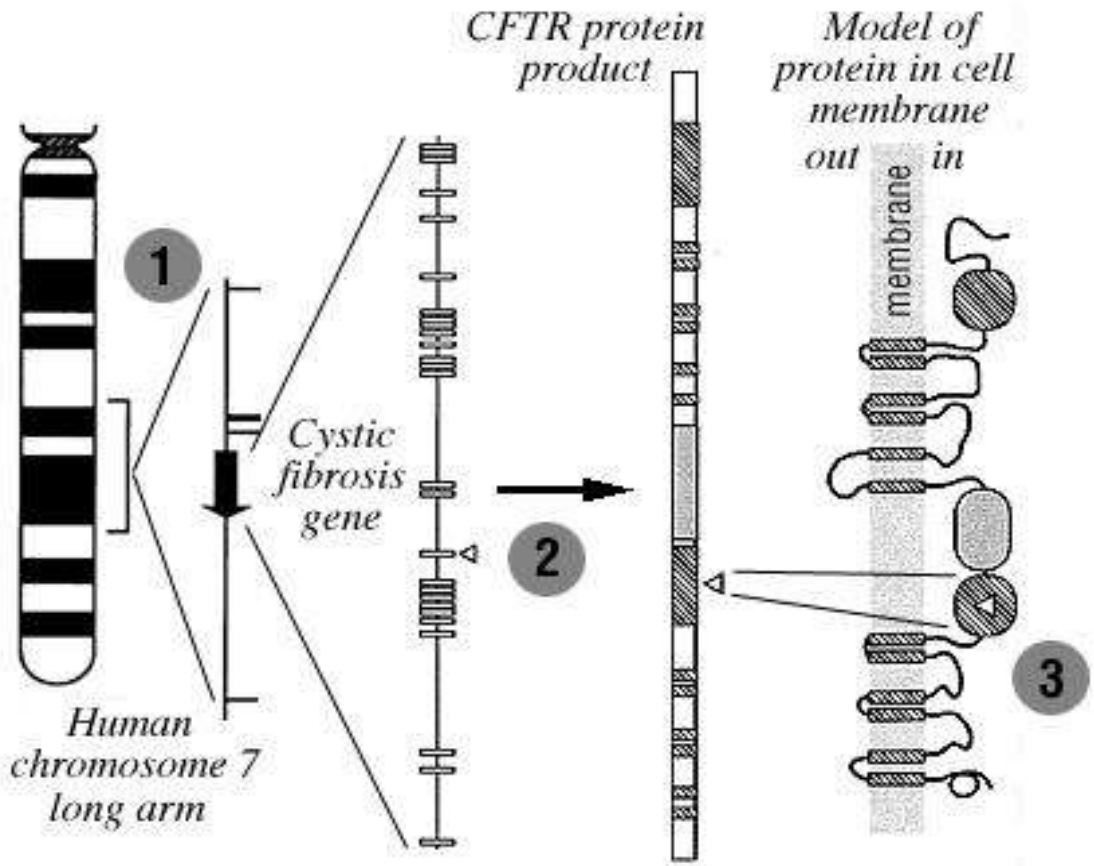
Inheritance of Cystic Fibrosis:

- In early 1980's, biologists hypothesized that CF is an autosomal recessive disorder caused by mutations in a gene that remained unknown until 1989.
- Heterozygous carriers are asymptomatic.
- Must be homozygously recessive in this gene in order to be diagnosed with CF.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Cystic Fibrosis: finding the gene



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Cystic Fibrosis

Finding Similarities between Cystic Fibrosis Gene and ATP binding proteins:

- ATP binding proteins are present on cell membrane and act as transport channel.
- In 1989, biologists found similarity between cystic fibrosis gene and ATP binding proteins.
- A plausible function for cystic fibrosis gene, given that CF involves sweet secretion with abnormally high sodium level.

http://www.bioalgorithms.info

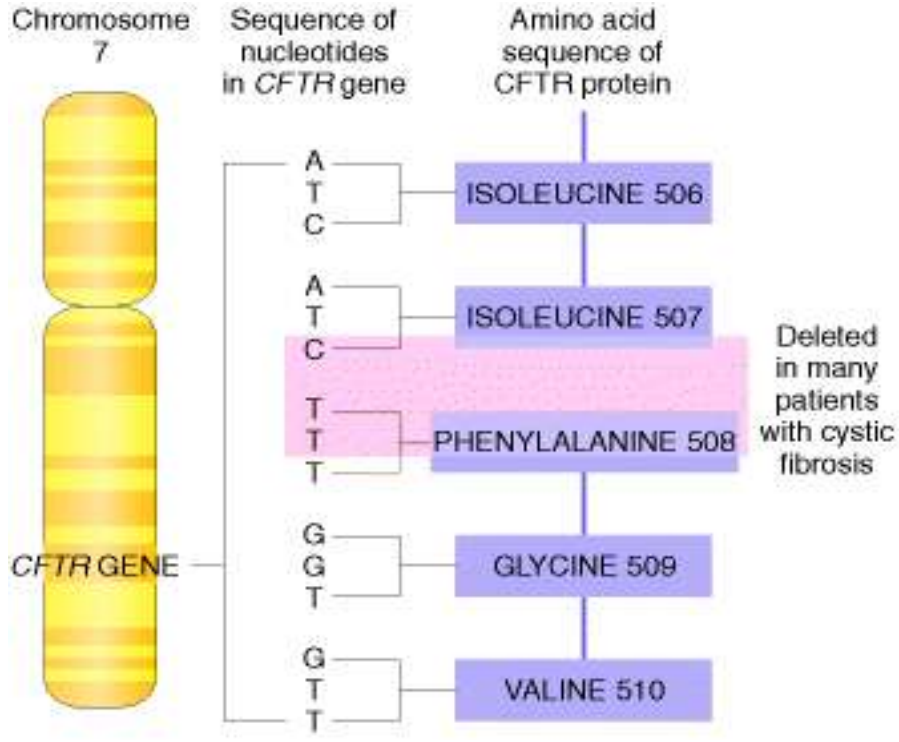
Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Cystic Fibrosis: mutation analysis

If a high percentage of cystic fibrosis (CF) patients have a certain mutation in the gene, while unaffected patients don't, that could be an indicator of a mutation that is related to CF.

Indeed, a certain mutation was found in 70% of CF patients; this is convincing evidence of a predominant genetic diagnostics marker for CF.

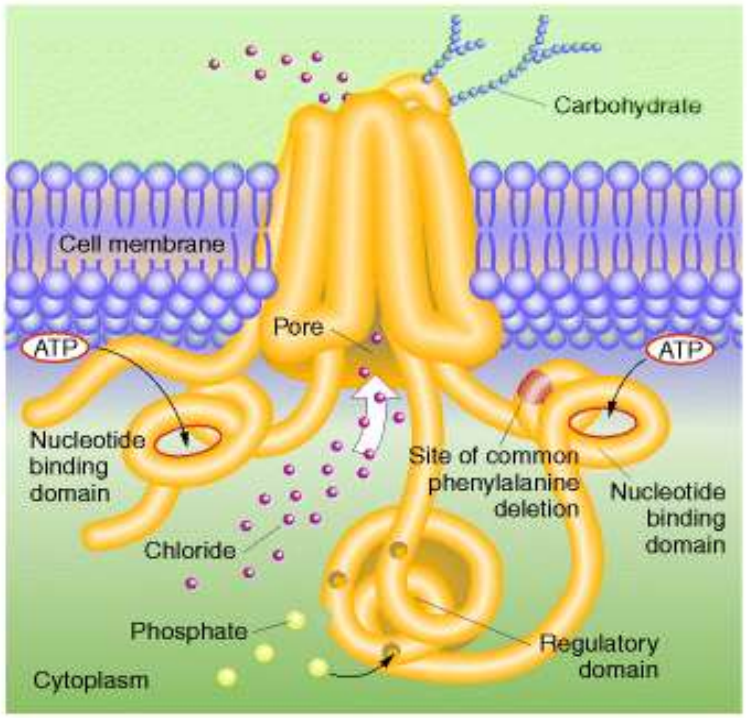


http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Cystic Fibrosis and the CFTR protein

- CFTR (Cystic Fibrosis Transmembrane conductance Regulator) protein acts in cell membrane of epithelial cells that secrete mucus.
- These cells line airways of nose, lungs, stomach wall, etc.



- CFTR protein (1480 amino acids) regulates a chloride ion channel: adjusts "wateriness" of fluids secreted by cell.
- CF sufferers are missing a single amino acid in their CFTR.
- Mucus ends up being too thick, affecting many organs.

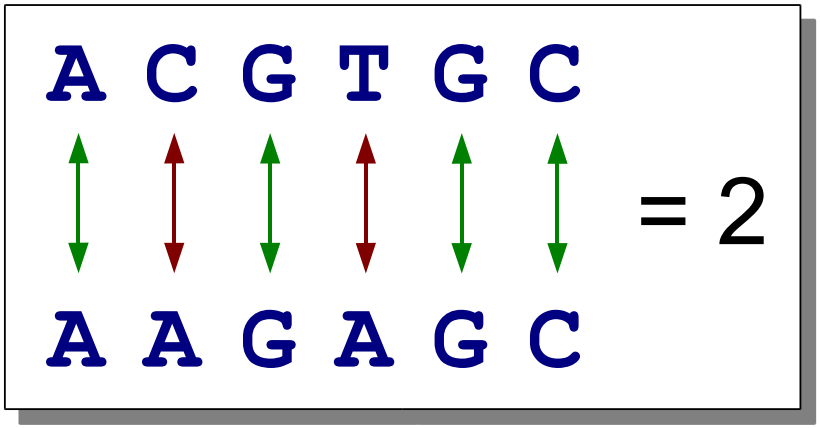
http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

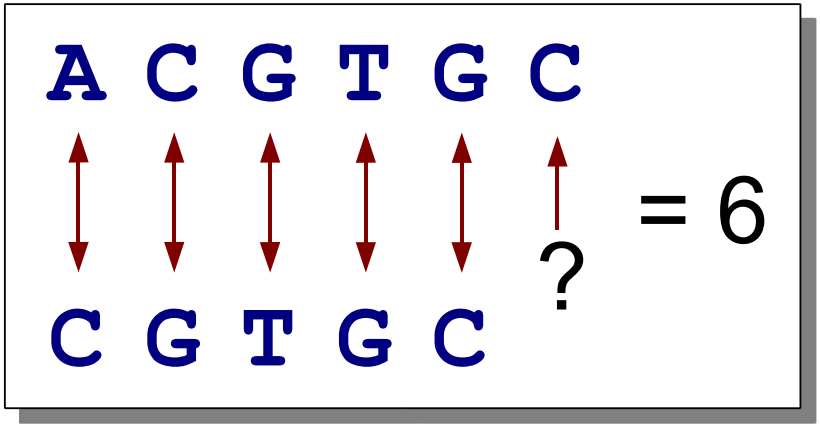
Sequence comparison: a false start

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



As we know, this is Hamming distance and forms the basis for most error correcting codes, as well as the motif-finding problem we saw earlier.

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

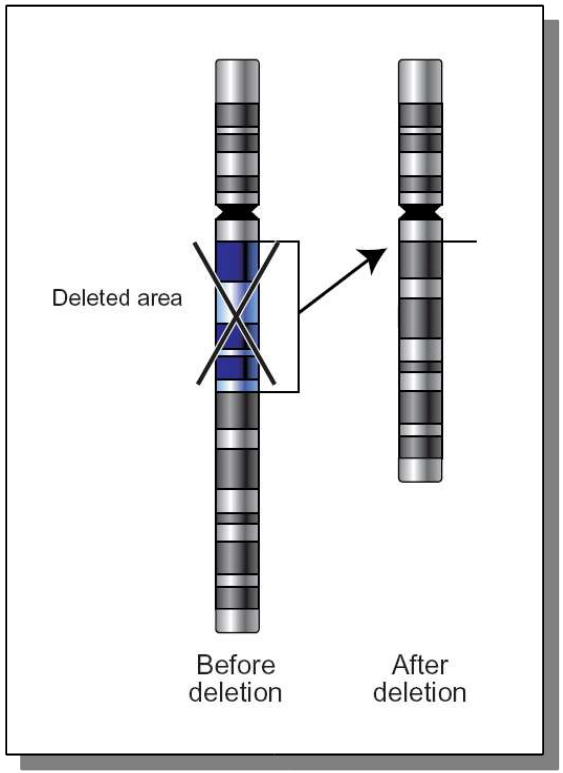


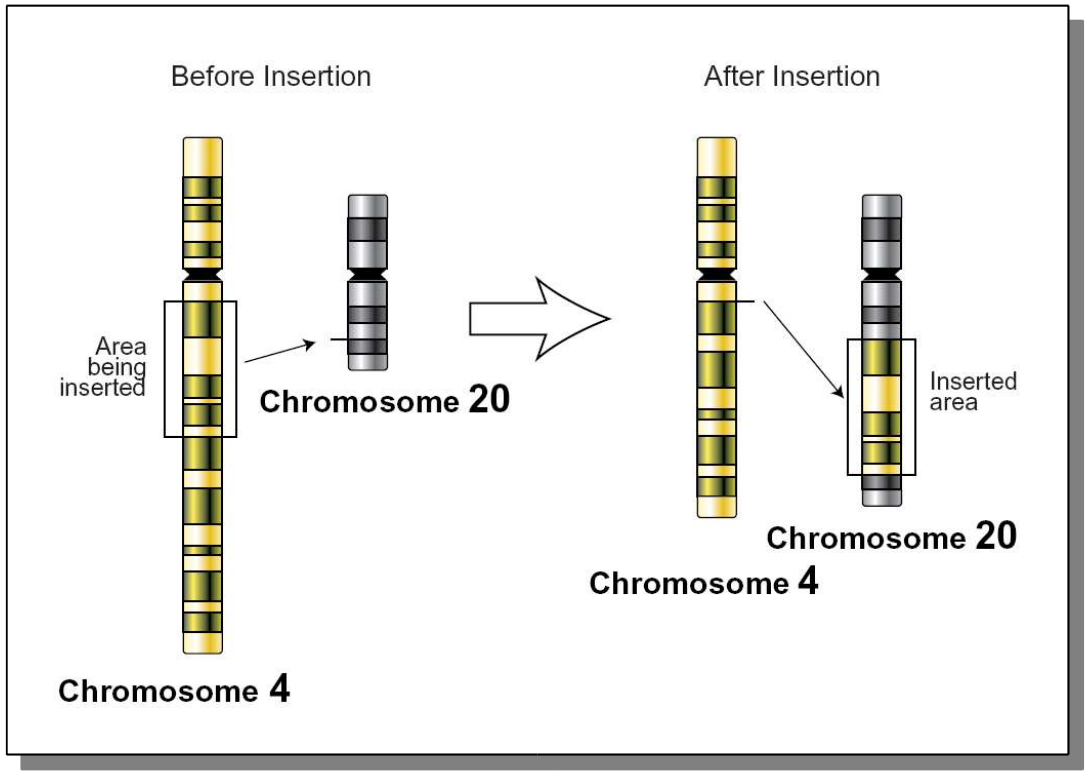
Just one missing symbol at the start of the second sequence leads to a large distance.

Source: Lehigh Uni. - Daniel Lopresti

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

Source: Lehigh Uni. - Daniel Lopresti

The human factor

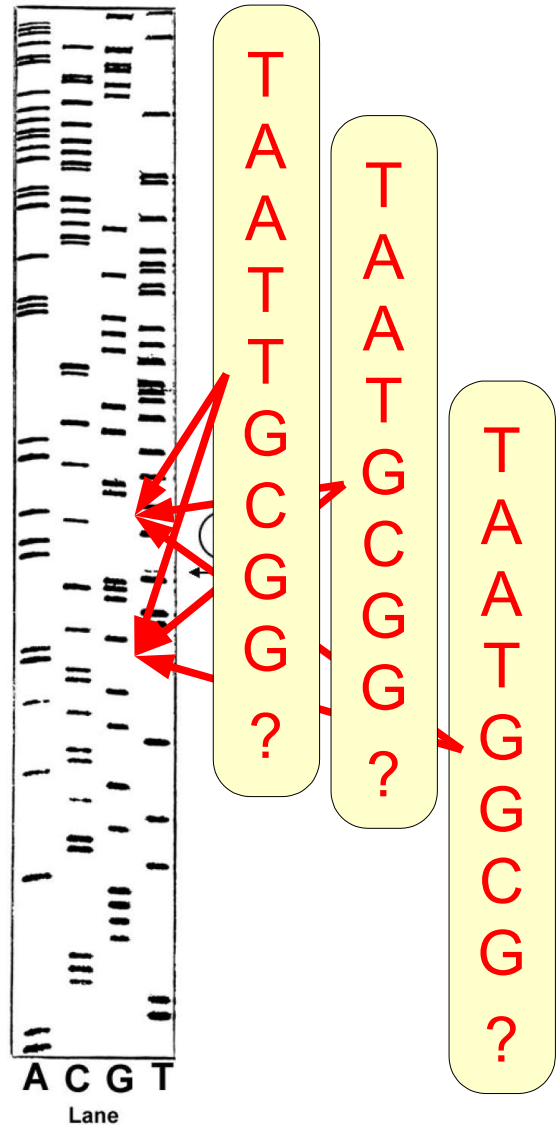
In addition, errors can arise during the sequencing process:

"...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

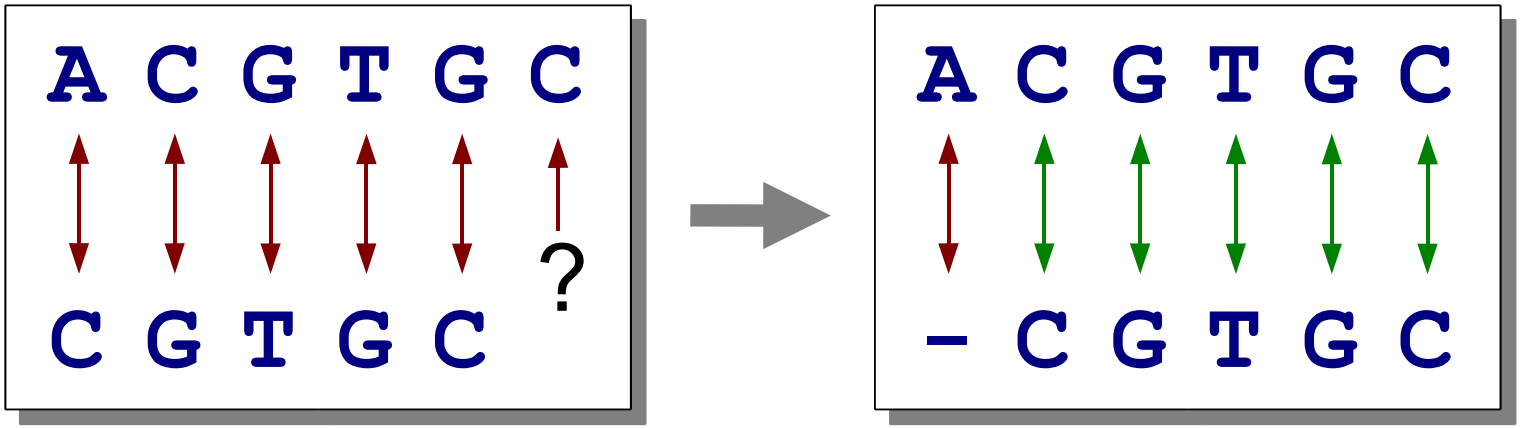


Source: Lehigh Uni. - Daniel Lopresti

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.

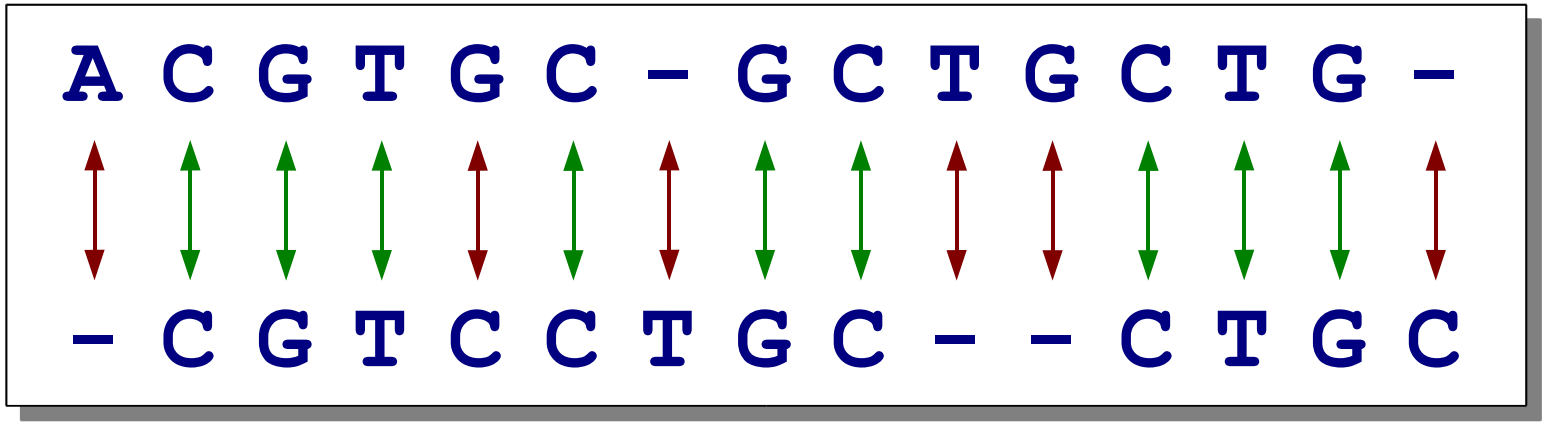
The notion of an *alignment* helps us visualize this:



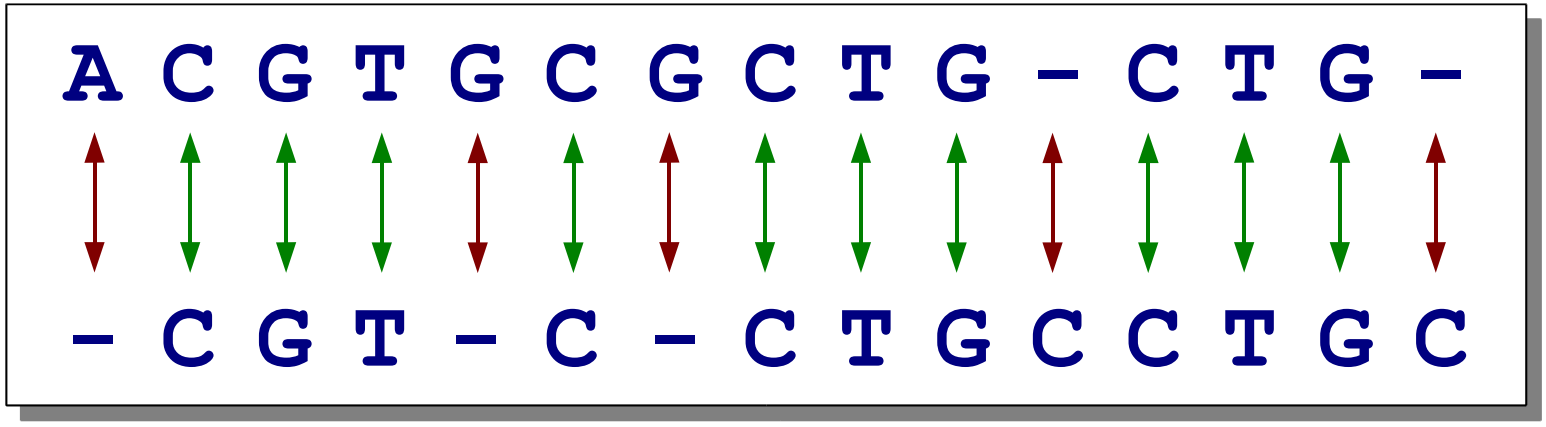
Alignment permits us to incorporate "spaces" (represented by a dash) in one or both sequences to make them the same length.

Source: Lehigh Uni. - Daniel Lopresti

Sequence alignment



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



No – this one has 5! How can we find the best alignment?

Source: Lehigh Uni. - Daniel Lopresti

Alignment: two-row representation

Given 2 DNA sequences *v* and *w*:

v : A T C T G A T ***m*** = 7

w : T G C A T A ***n*** = 6

Alignment : 2 * *k* matrix (*k* > *m*, *n*)

letters of ***v***

letters of ***w***

A	T	–	C	–	T	G	A	T
–	T	G	C	A	T	–	A	–

4 matches | 2 insertions | 3 deletions

http://www.bioalgorithms.info

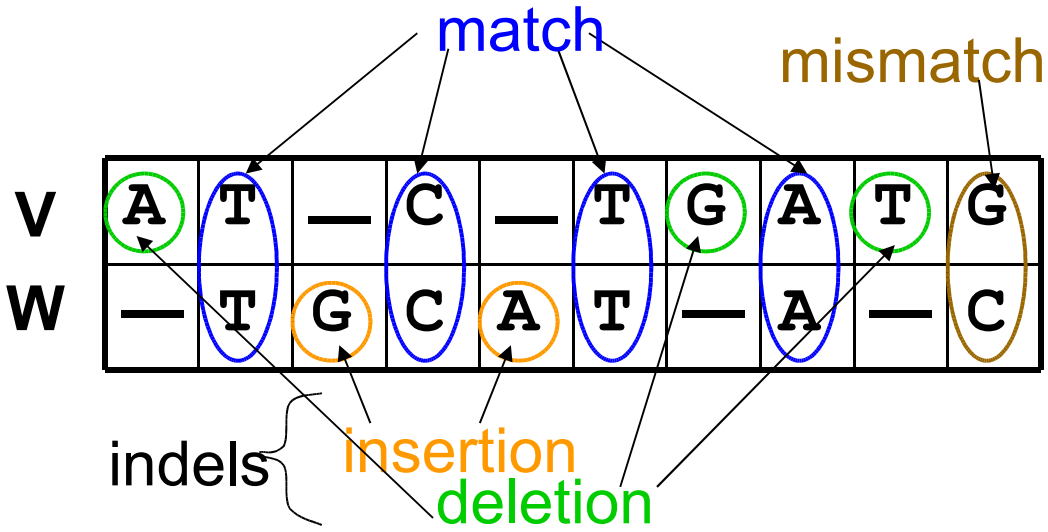
Source: Lehigh Uni. - Daniel Lopresti

Aligning DNA sequences

v = ATCTGATG $n = 8$

w = TGCATAC $m = 7$

4 matches
1 mismatches
3 deletions
2 insertions



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

The Longest Common Subsequence Problem (LCS)

The Longest Common Subsequence Problem.

Find the longest common subsequence of two sequences.

Input: Two sequences:

$$v = v_1\, v_2 \ldots v_m \quad \text{and} \quad w = w_1\, w_2 \ldots w_n$$

Output: A sequence of positions in

$$v: \; 1 \le i_1 < i_2 < \ldots < i_t \le m$$

and a sequence of positions in

$$w: \; 1 \le j_1 < j_2 < \ldots < j_t \le n$$

such that symbol i_k of v matches j_k of w and t is maximal.

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

LCS example

i coords:	0	1	2	2	3	3	4	5	6	7	8
elements of *v*		A	T	–	C	–	T	G	A	T	C
elements of *w*		–	T	G	C	A	T	–	A	–	C
j coords:	0	0	1	2	3	4	5	5	6	6	7

(0,0)→ (1,0)→ (2,1)→ (2,2)→ (3,3)→ (3,4)→ (4,5)→ (5,5)→ (6,6)→ (7,6)→ (8,7)

Matches shown in red

positions in *v*: 2 < 3 < 4 < 6 < 8

positions in *w*: 1 < 3 < 5 < 6 < 7

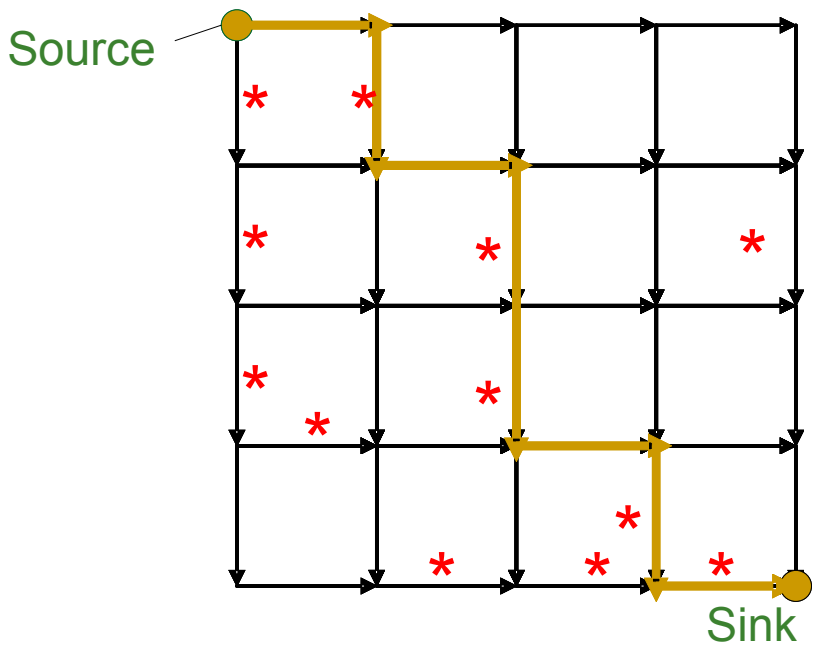
Every common subsequence is a path in 2-D grid.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

The Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid.

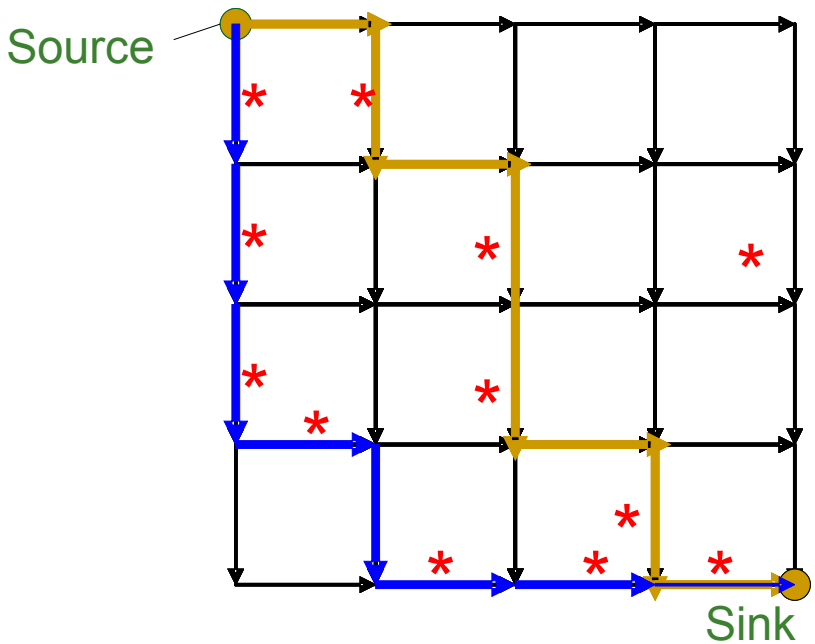


http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

The Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid.



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

The Manhattan Tourist Problem (MTP)

The Manhattan Tourist Problem.

Find the longest path in a weighted grid.

Input: A weighted grid *G* with two distinct vertices, one labeled "source" and the other labeled "sink."

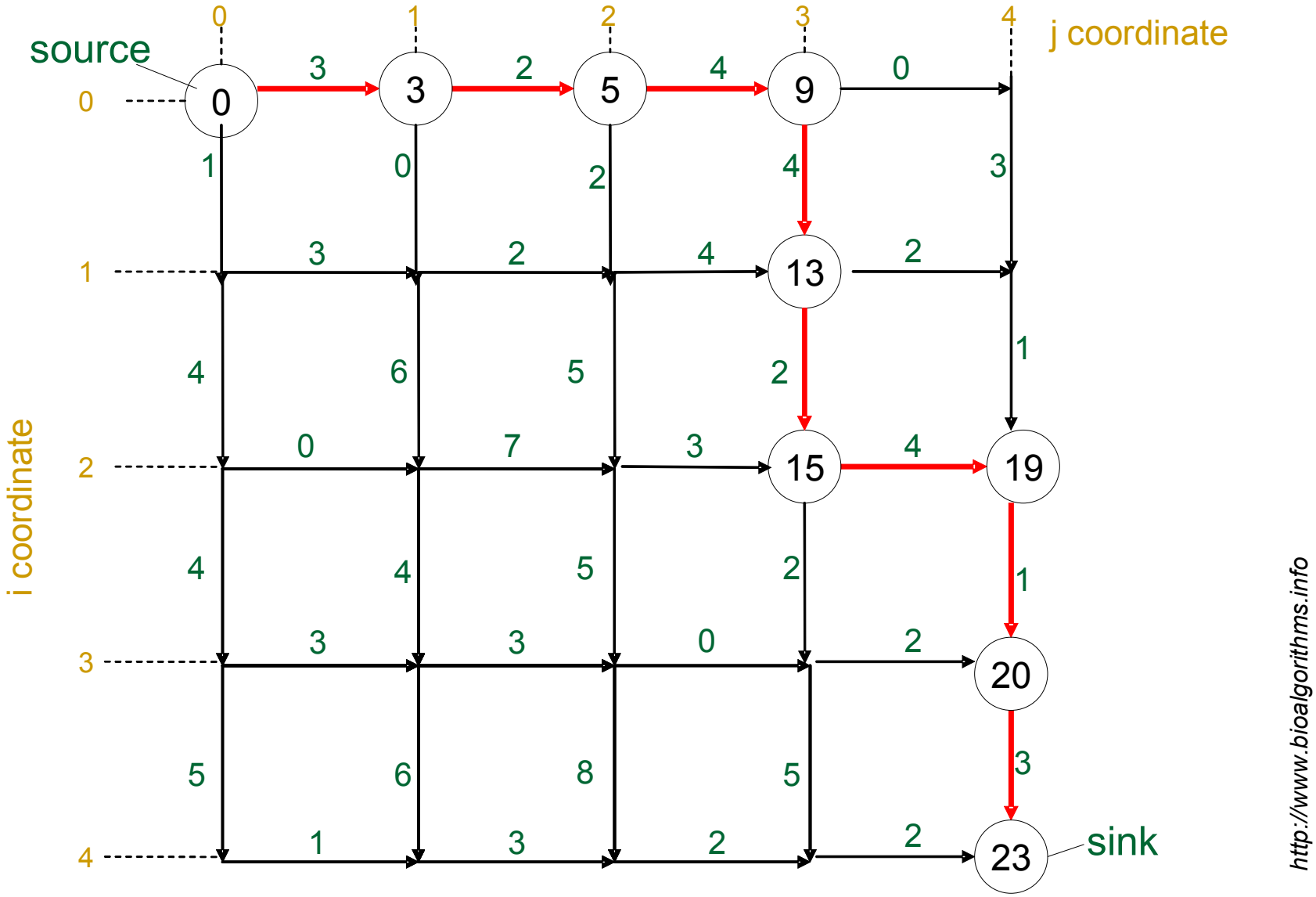
Output: A longest path in *G* from "source" to "sink."

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

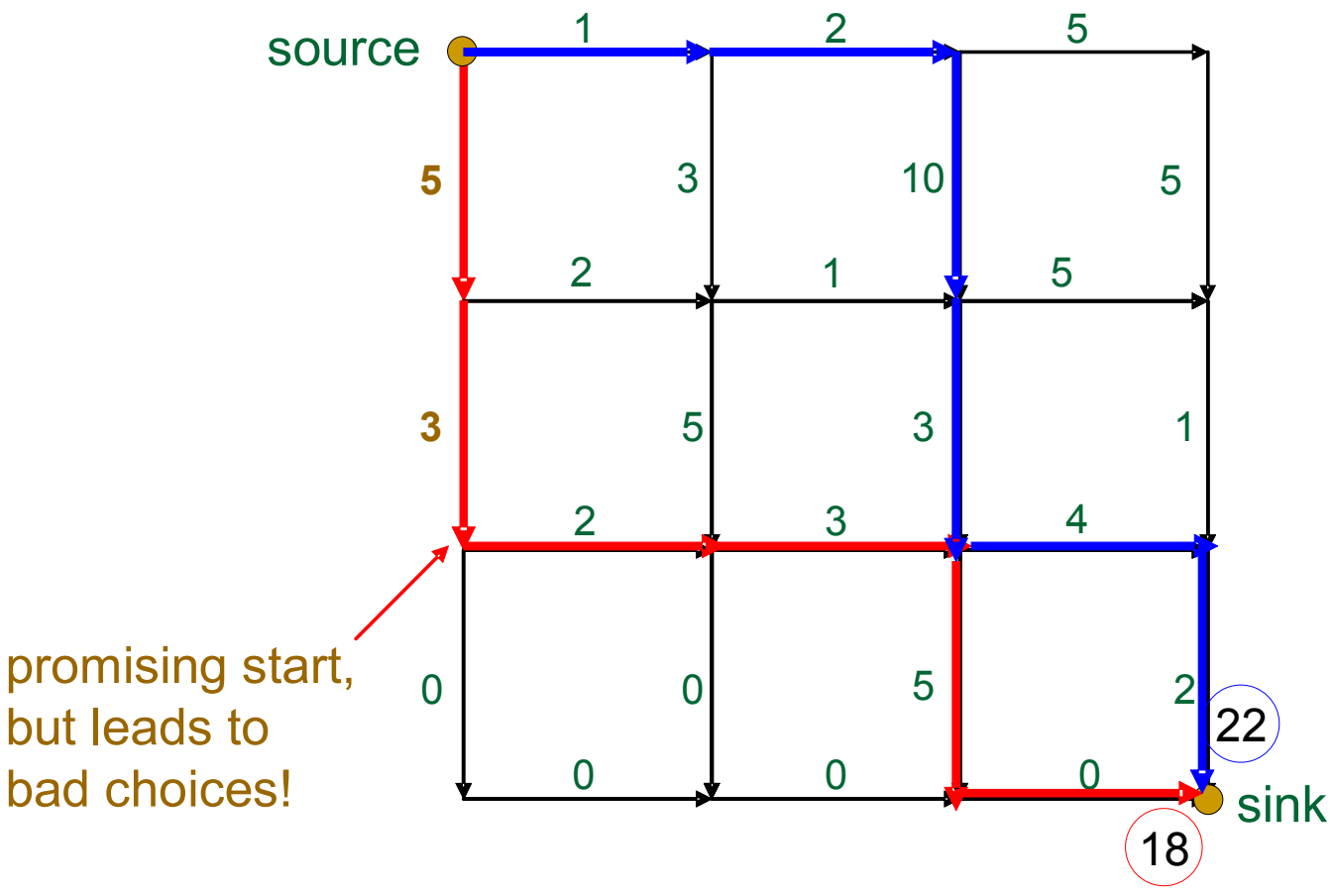
LEHIGH UNIVERSITY

MTP: an example



Source: Lehigh Uni. - Daniel Lopresti

MTP: greedy is not optimal



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

MTP: a simple recursive program

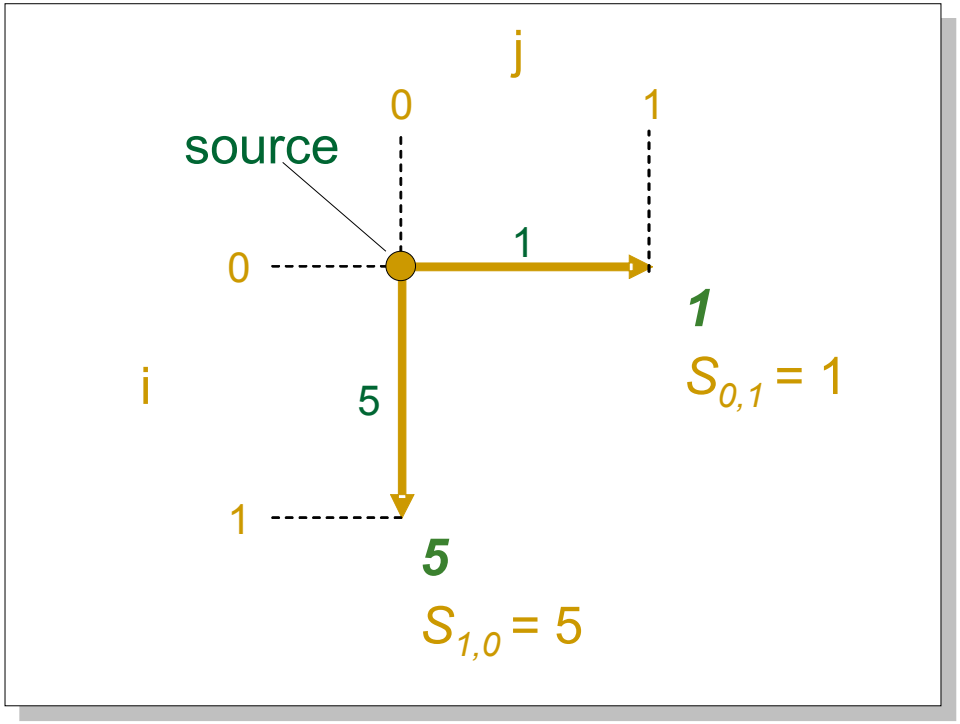
```
MT(n, m)
    if n = 0 and m = 0
        return 0
    if n > 0
        x ← MT(n-1, m) + length of edge from (n-1, m) to (n, m)
    else
        x ← 0
    if m > 0
        y ← MT(n, m-1) + length of edge from (n, m-1) to (n, m)
    else
        y ← 0
    return max(x, y)
```

What's wrong with this approach?

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

MTP: Dynamic Programming

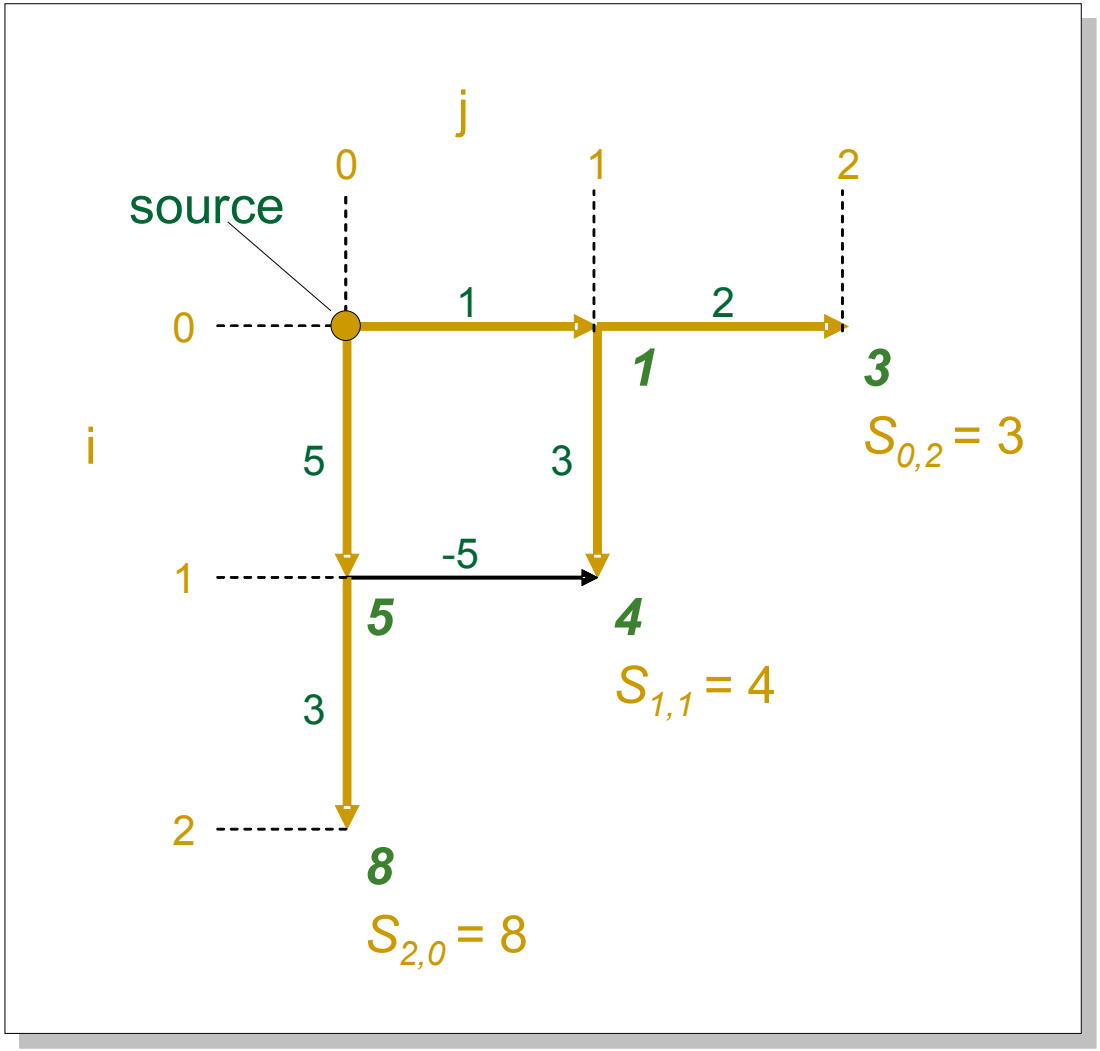


- Calculate optimal path score for each vertex in graph.
- Each vertex's score is maximum of prior vertices' scores plus weight of respective edges in between.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

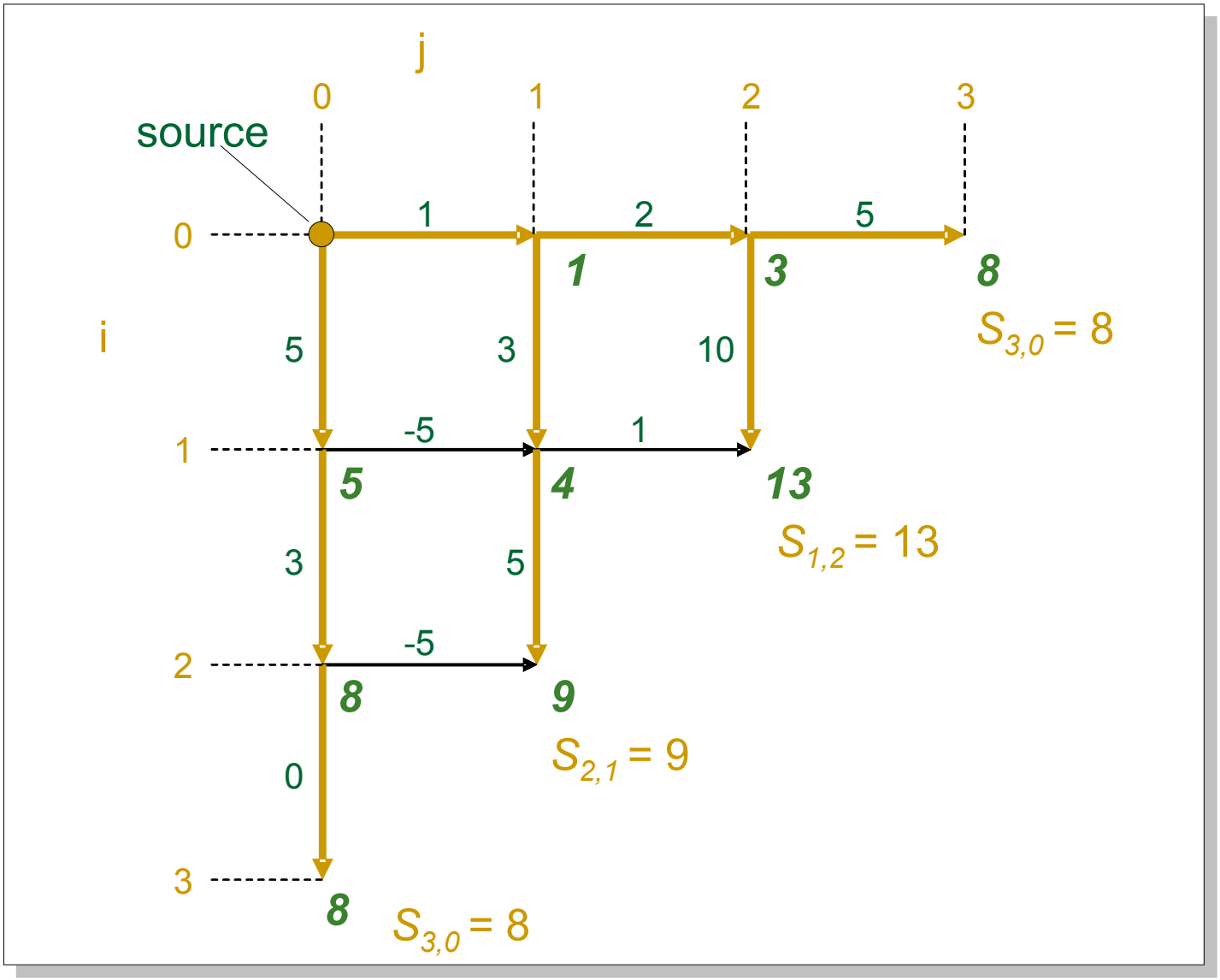
MTP: Dynamic Programming (continued)



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

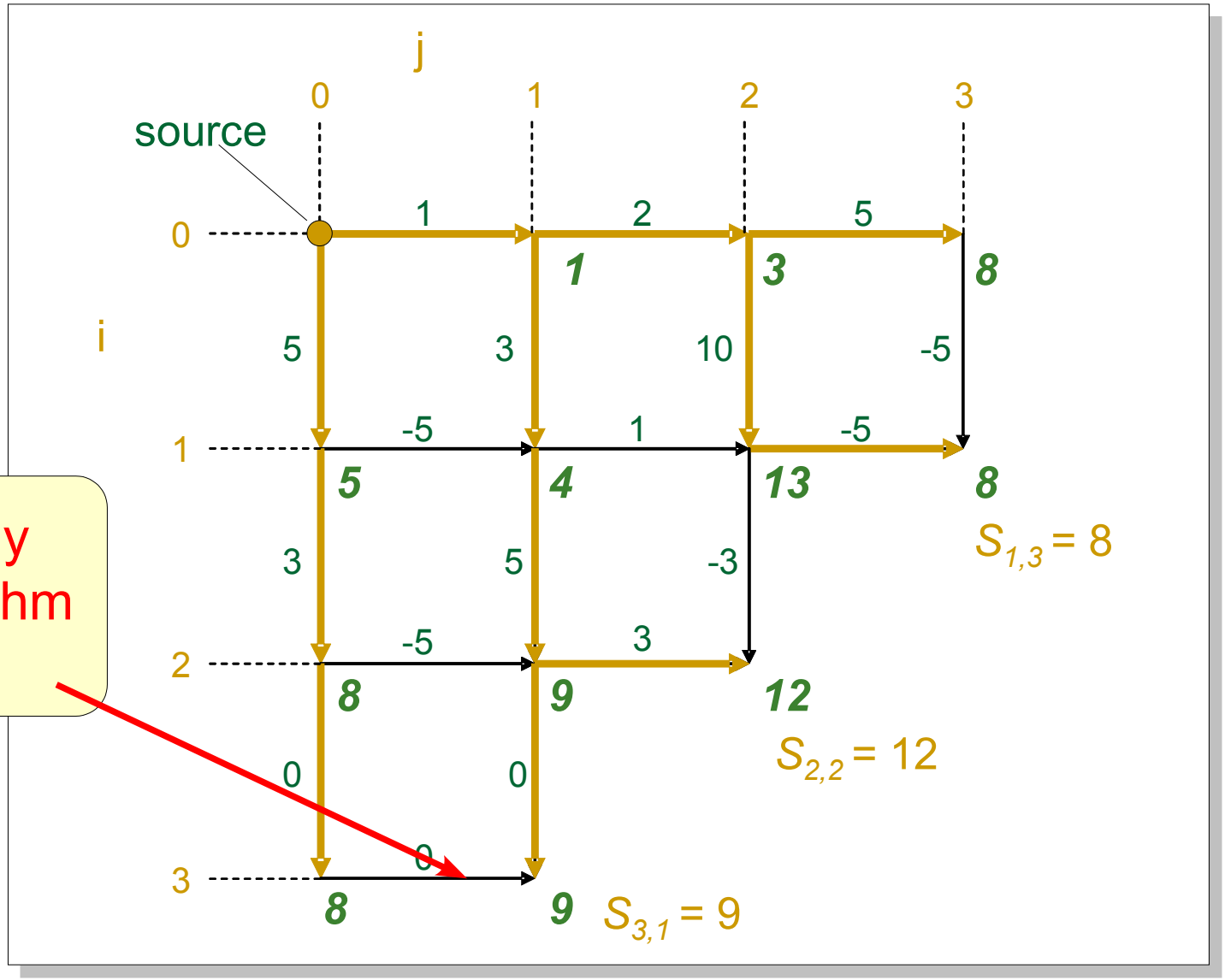
MTP: Dynamic Programming (continued)



http://www.bioalgorithms.info

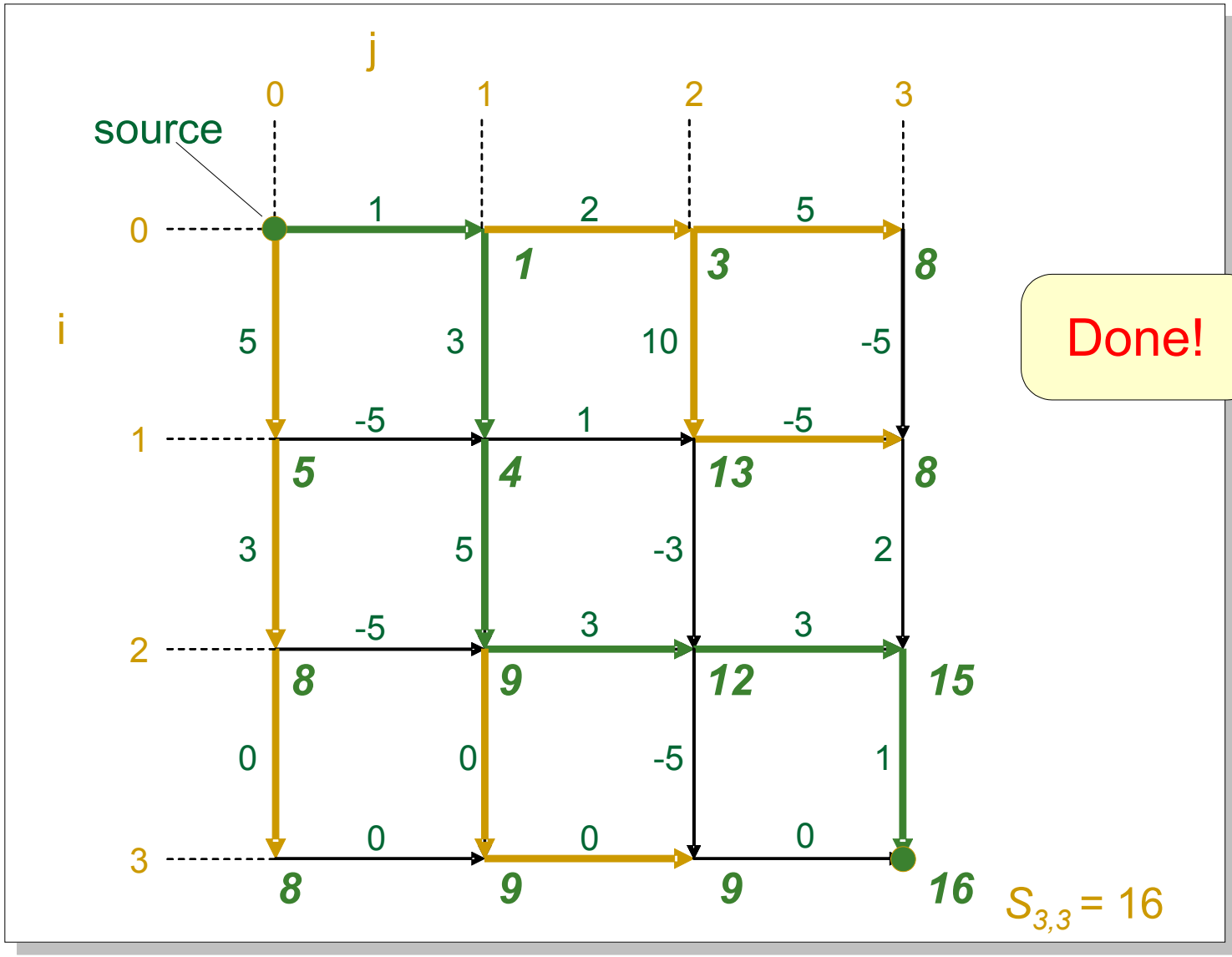
Source: Lehigh Uni. - Daniel Lopresti

MTP: Dynamic Programming (continued)



Source: Lehigh Uni. - Daniel Lopresti

MTP: Dynamic Programming (continued)



Done!

Source: Lehigh Uni. - Daniel Lopresti

MTP: recurrence

Computing score for a point (i, j) by recurrence relation:

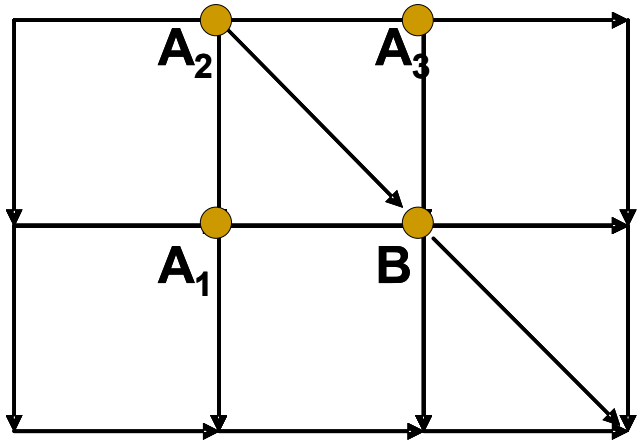
$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of the edge between } (i-1, j) \text{ and } (i, j) \\ s_{i,j-1} + \text{weight of the edge between } (i, j-1) \text{ and } (i, j) \end{cases}$$

The running time is $n \times m$ for a n-by-m grid, where n is the number of rows and m is the number of columns.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

But Manhattan is not a perfect grid ...



What about diagonals?

Score at point B is given by:

$$s_B = \max \text{ of } \begin{cases} s_{A1} + \text{weight of the edge } (A_1, B) \\ s_{A2} + \text{weight of the edge } (A_2, B) \\ s_{A3} + \text{weight of the edge } (A_3, B) \end{cases}$$

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

But Manhattan is not a perfect grid ...

Computing score s_x for point x is given by recurrence relation:

$$s_x = \max \text{ of } \begin{cases} s_y + \text{weight of edge } (y, x) \text{ where} \\ \qquad y \in \text{Predecessors}(x) \end{cases}$$

Predecessors(x) = set of vertices that have edges leading to x

Running time for a graph $G(V, E)$ is $O(E)$ since each edge is evaluated once

Note: V is set of all vertices and E is set of all edges.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

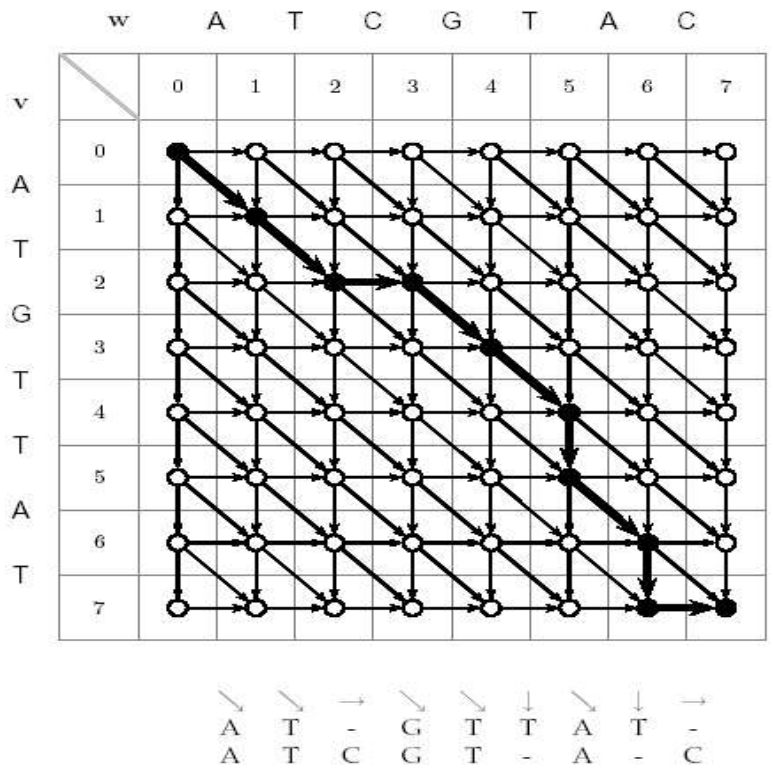
LCS Dynamic Programming

Find LCS of two strings.

Input: a weighted graph G with two distinct vertices, one labeled "source" and one labeled "sink."

Output: a longest path in G from "source" to "sink."

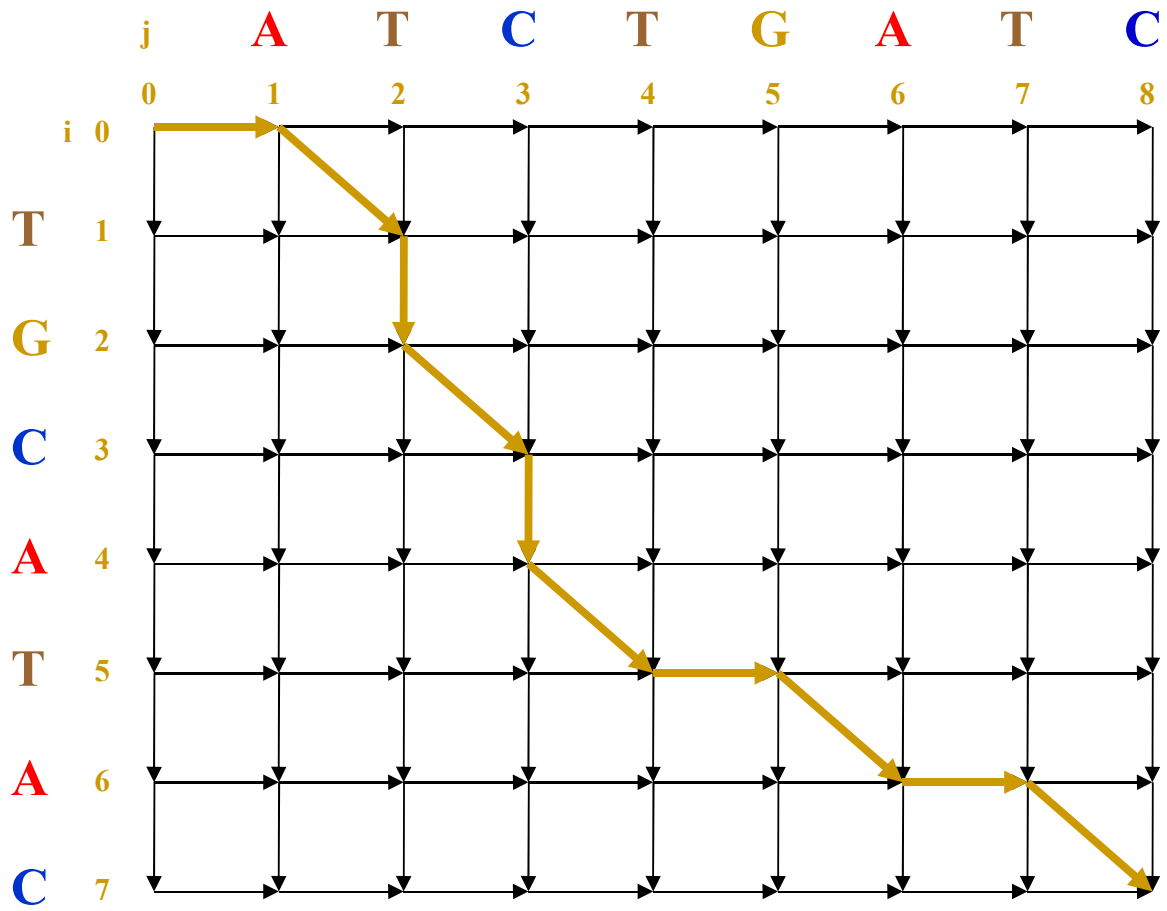
Solve using an MTP graph with diagonals replaced by +1 edges.



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

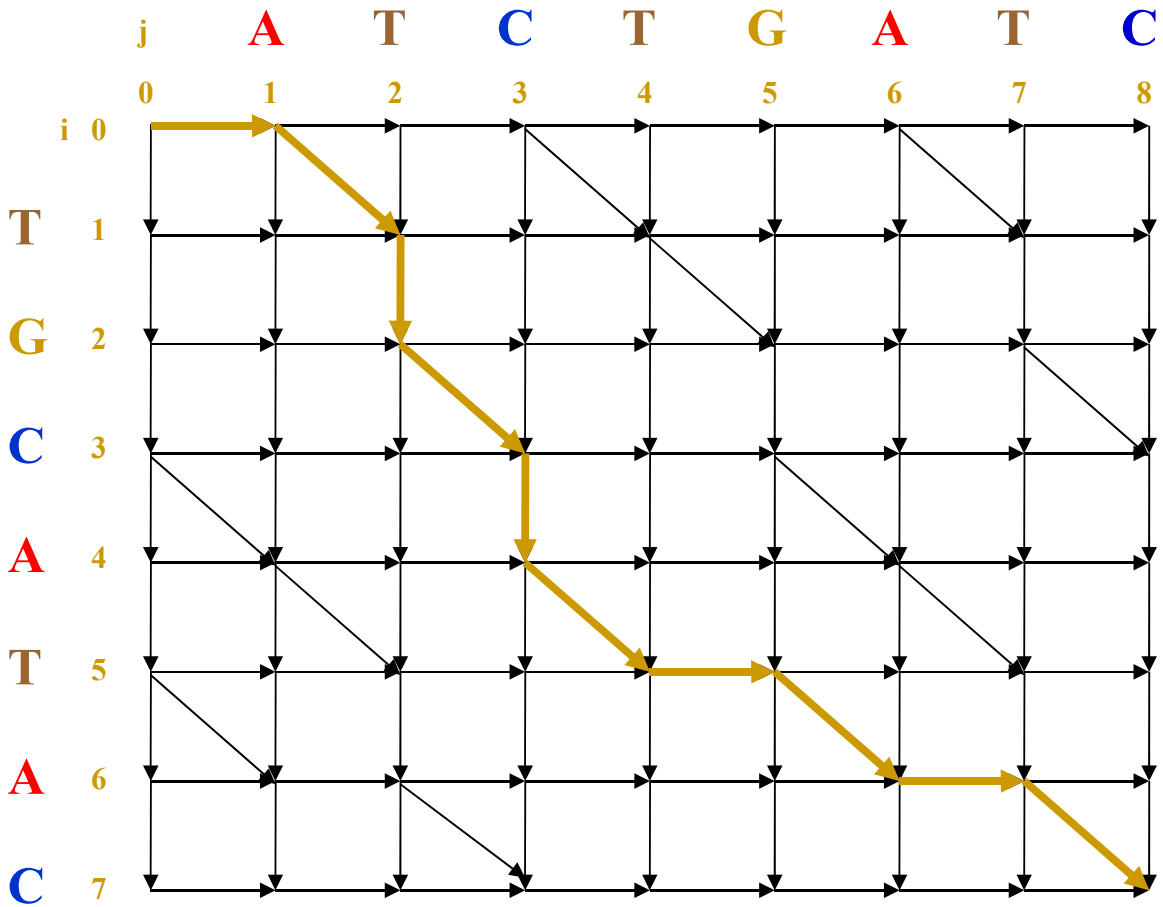
LCS Problem as Manhattan Tourist Problem



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

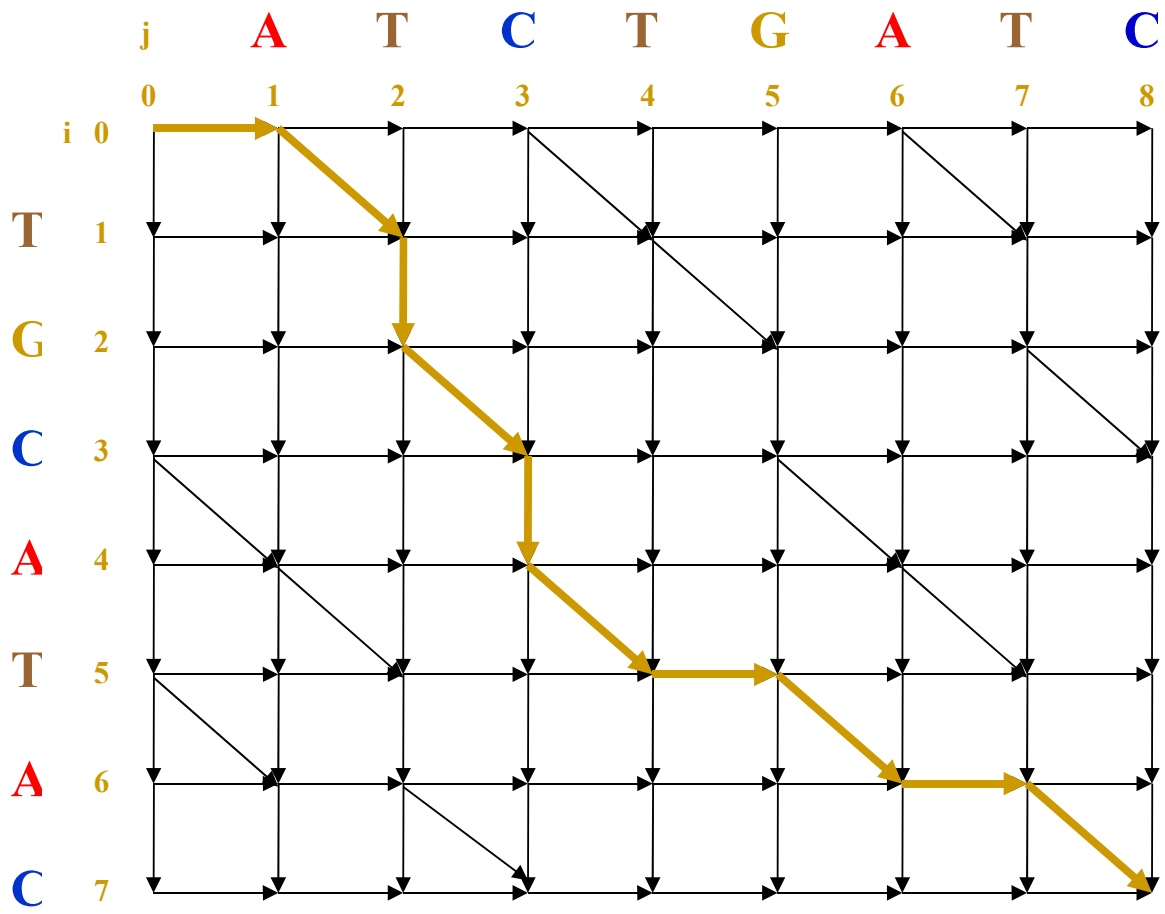
MTP Graph for LCS Problem



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

MTP Graph for LCS Problem



Every path is common subsequence.

Diagonal edge adds extra element to subsequence.

LCS Problem: **f**ind path with maximum diagonal edges.

http://www.bioalgorithms.info

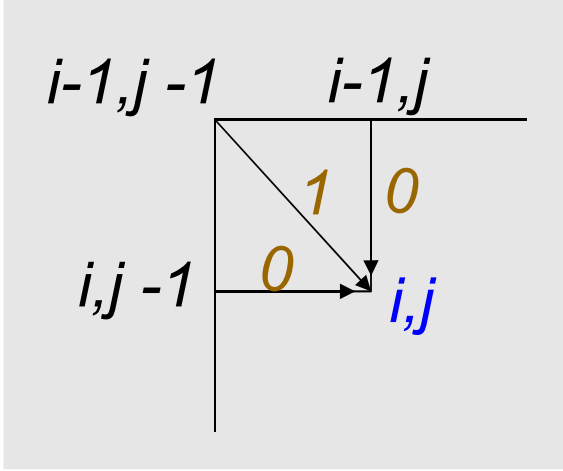
Source: Lehigh Uni. - Daniel Lopresti

Computing LCS

Let v_i = prefix of v of length i : $v_1 \ldots v_i$
and w_j = prefix of w of length j : $w_1 \ldots w_j$

The length of LCS(v_i, w_j) is computed by:

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

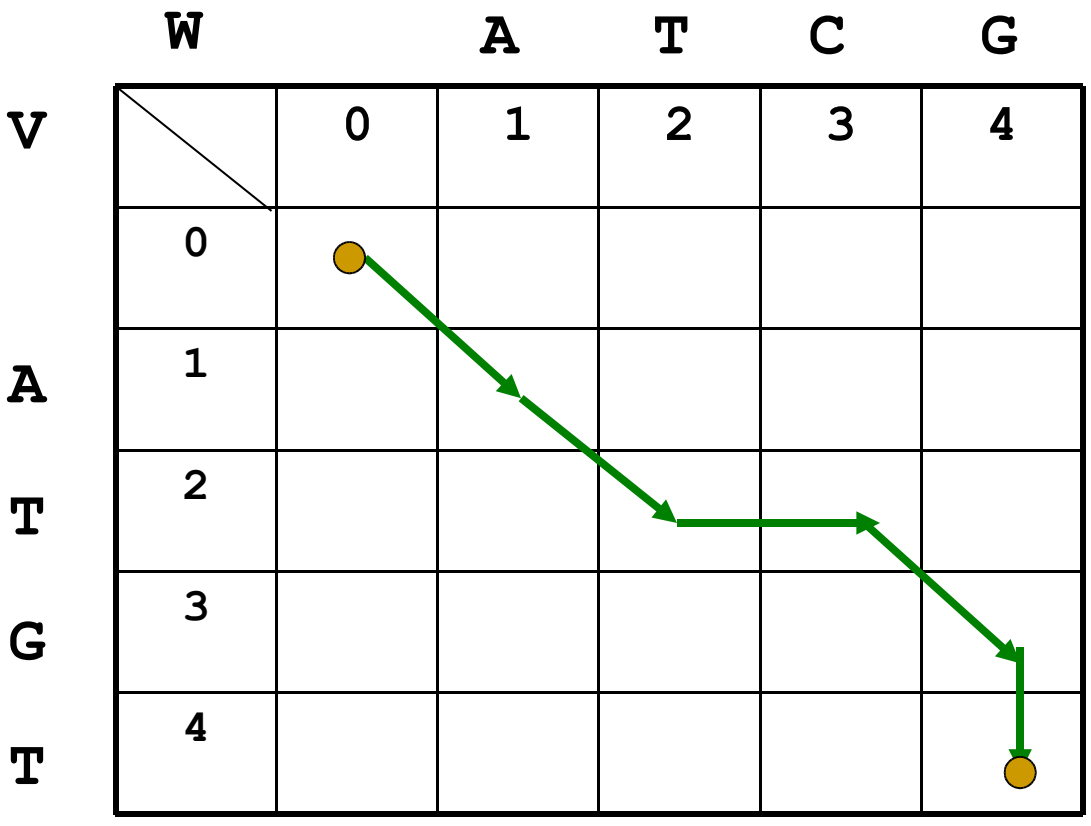


http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LCS Alignments

Every path in grid corresponds to an alignment:




```
    ↘ ↘ → ↘ ↓
  0 1 2 2 3 4
V = A T - G T
    | |   |
W = A T C G -
  0 1 2 3 4 4
```

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Edit distance

In 1966, Levenshtein introduced *edit distance* between two strings as minimum number of elementary operations (insertions, deletions, and substitutions) needed to transform one string into the other.

$d(v, w)$ = MIN number of elementary operations needed to transform $v \rightarrow w$.

Keep in mind that LCS computation uses a MAX.

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

The Edit Distance Problem

The Edit Distance Problem.

Given two sequences, find the optimal series of deletions, insertions, and substitutions to transform one into the other.

Input: Two sequences:

$$v = v_1\, v_2 \ldots v_m \quad \text{and} \quad w = w_1\, w_2 \ldots w_n$$

Output: An optimal series of basic editing operations:

$$e_1, e_2, \ldots, e_t$$

such then, when applied to one of the sequences, say v, it is transformed into the other sequence, w.

Here "optimal" can mean any of a number of things, including "fewest" or "lowest- / highest-cost."

Source: Lehigh Uni. - Daniel Lopresti

Edit distance vs. Hamming distance

Hamming distance always compares $i^{\text{-th}}$ letter of v with $i^{\text{-th}}$ letter of w

```
v = ATATATAT
    |||||||||
w = TATATATA
```

Just one shift

Make it all line up

```
v = -ATATATAT
     |||||||
w = TATATATA
```

Hamming distance:

$d(\boldsymbol{v}, \boldsymbol{w}) = 8$

Computing Hamming distance is trivial task.

Edit distance may compare $i^{\text{-th}}$ letter of v with $j^{\text{-th}}$ letter of w

Edit distance:

$d(\boldsymbol{v}, \boldsymbol{w}) = 2$

Computing edit distance is non-trivial task.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Edit distance vs. Hamming distance

Hamming distance always compares $i^{\text{-th}}$ letter of v with $i^{\text{-th}}$ letter of w

Edit distance may compare $i^{\text{-th}}$ letter of v with $j^{\text{-th}}$ letter of w

```
v = ATATATAT
    |||| |||
w = TATATATA
```

Just one shift

Make it all line up

```
v = -ATATATAT
     ||| ||||
w = TATATATA-
```

Hamming distance:

$d(\boldsymbol{v}, \boldsymbol{w}) = 8$

Edit distance:

$d(\boldsymbol{v}, \boldsymbol{w}) = 2$

How to find which j goes with which i ???

One insertion and one deletion.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Edit distance example

TGCATAT → ATCCGAT in 5 steps:

TGCATAT	→	(delete last T)
TGCATA	→	(delete last A)
TGCAT	→	(insert A at front)
ATGCAT	→	(substitute C for 3rd G)
ATCCAT	→	(insert G before last A)
ATCCGAT		(Done)

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Edit distance example

TGCATAT → ATCCGAT in 5 steps:

TGCATAT → (delete last T)
TGCATA → (delete last A)
TGCAT → (insert A at front)
ATGCAT → (substitute C for 3rd G)
ATCCAT → (insert G before last A)
ATCCGAT (Done)

What is the edit distance? 5?

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Edit distance example

TGCATAT → ATCCGAT in 4 steps:

TGCATAT → (insert A at front)
ATGCATAT → (delete 8th T)
ATGCATA → (substitute G for 5th A)
ATGCGTA → (substitute C for 3rd G)
ATCCGAT (Done)

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Edit distance example

TGCATAT → ATCCGAT in 4 steps:

TGCATAT → (insert A at front)

ATGCATAT → (delete 8th T)

ATGCATA → (substitute G for 5th A)

ATGCGTA → (substitute C for 3rd G)

ATCCGAT (Done)

Can it be done in 3 steps???

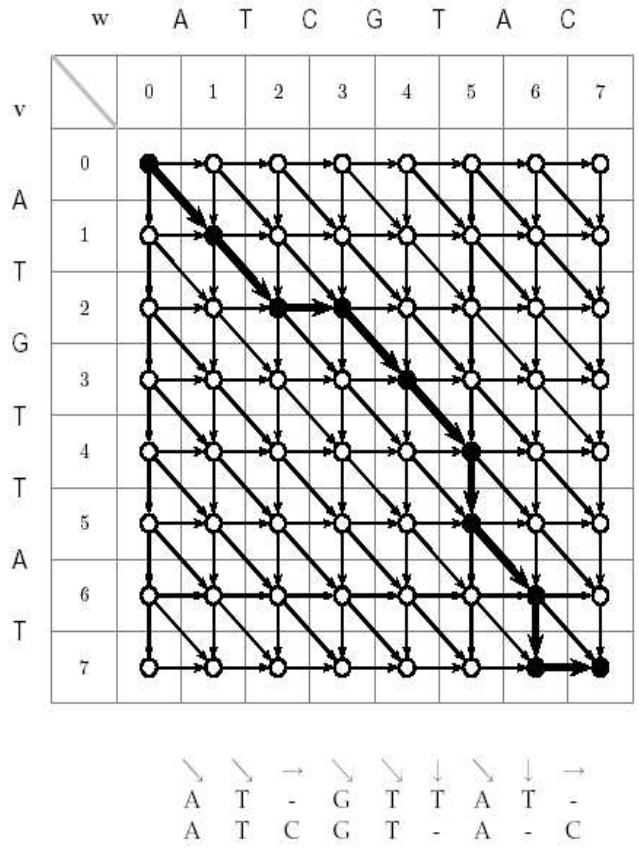
http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Alignment grid

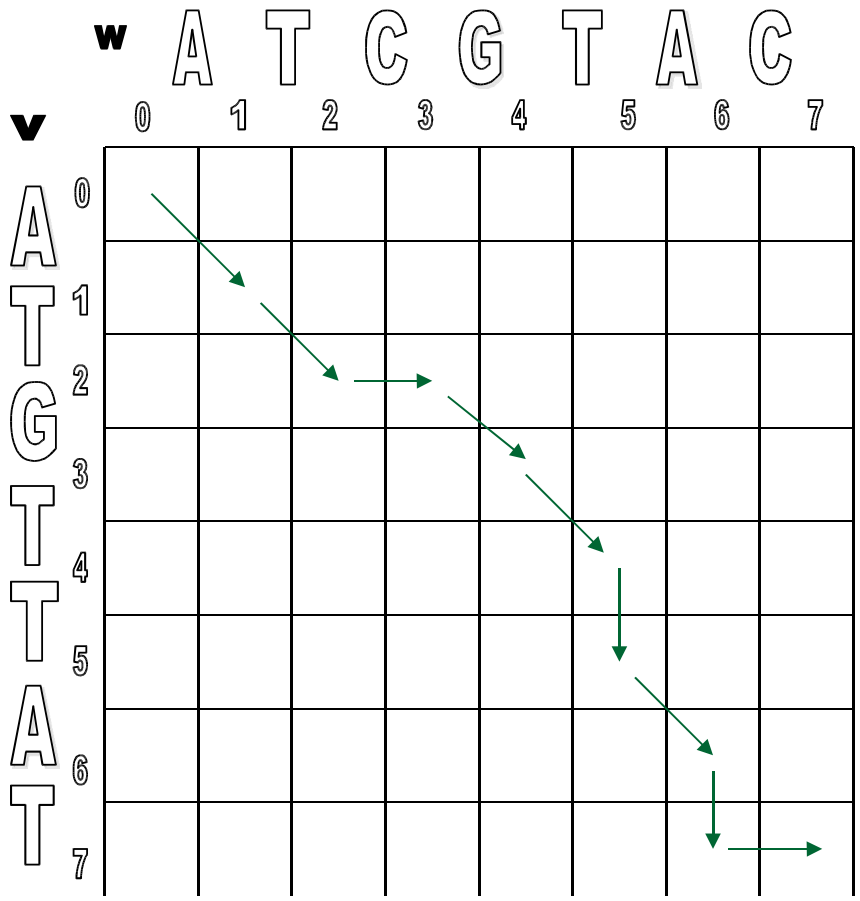
		0	1	2	2	3	4	5	6	7	7
v	=		A	T	-	G	T	T	A	T	-
w	=		A	T	C	G	T	-	A	-	C
		0	1	2	3	4	5	5	6	6	7

Every alignment path is from source to sink.



Source: Lehigh Uni. - Daniel Lopresti

Alignment as a path in edit graph




```
0 1 2 2 3 4 5 6 7 7
  A T _ G T T A T _
  A T C G T _ A _ C
0 1 2 3 4 5 5 6 6 7
```

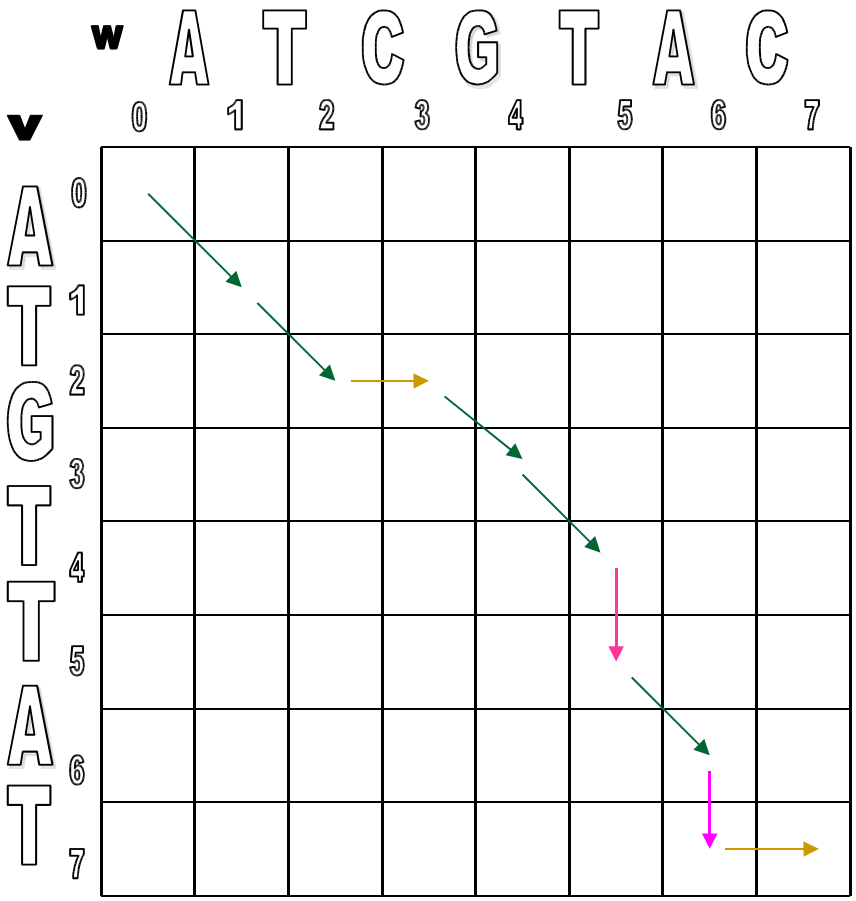
Corresponding path:

```
(0,0), (1,1), (2,2),
(2,3), (3,4), (4,5),
(5,5), (6,6), (7,6),
(7,7)
```

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Alignment as a path in edit graph



↓ and → represent indels in v and w with score 0.

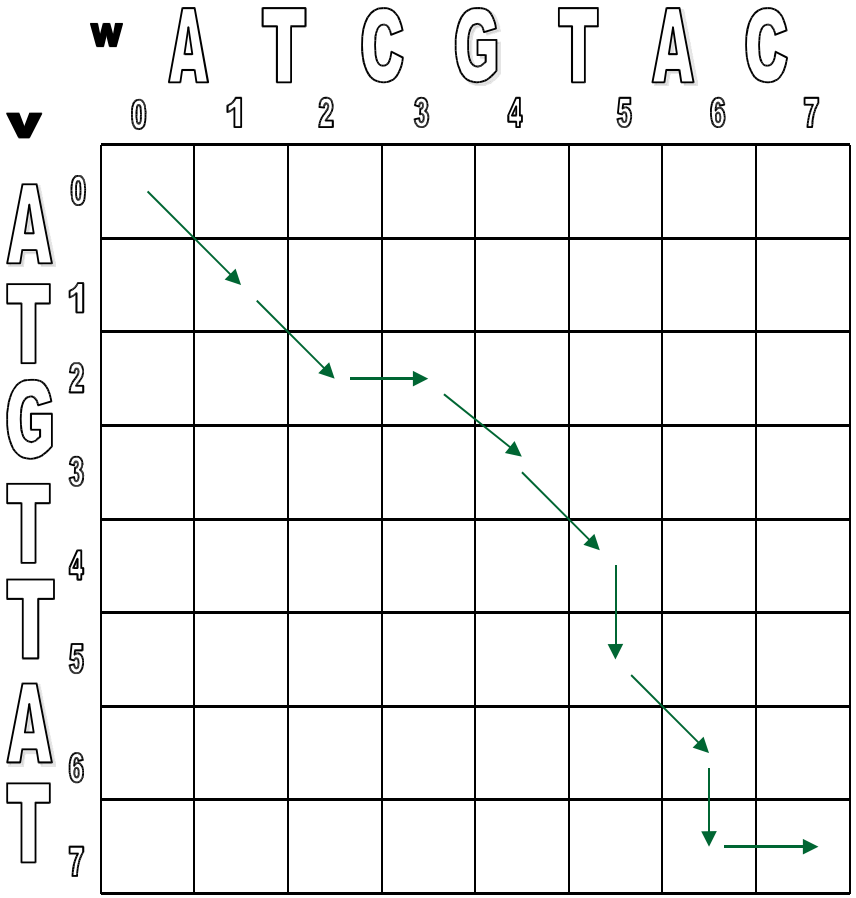
↘ represent matches with score 1.

The score of the alignment path is 5.

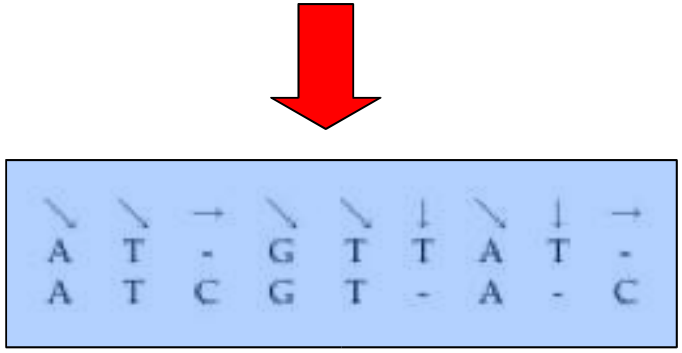
http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Alignment as a path in edit graph



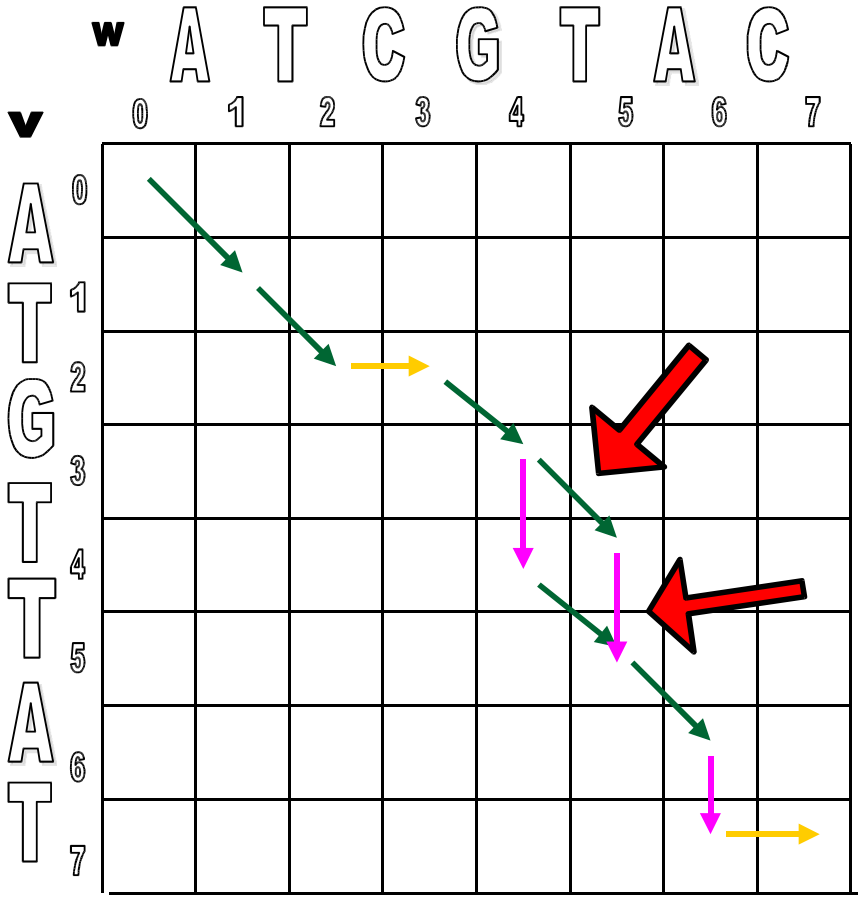
Every path in edit graph orresponds to an alignment:



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Alignment as a path in edit graph



Old Alignment

```
      01223456 77
v =   AT_GTTAT_
w =   ATCGT_A_C
      01234556 67
```

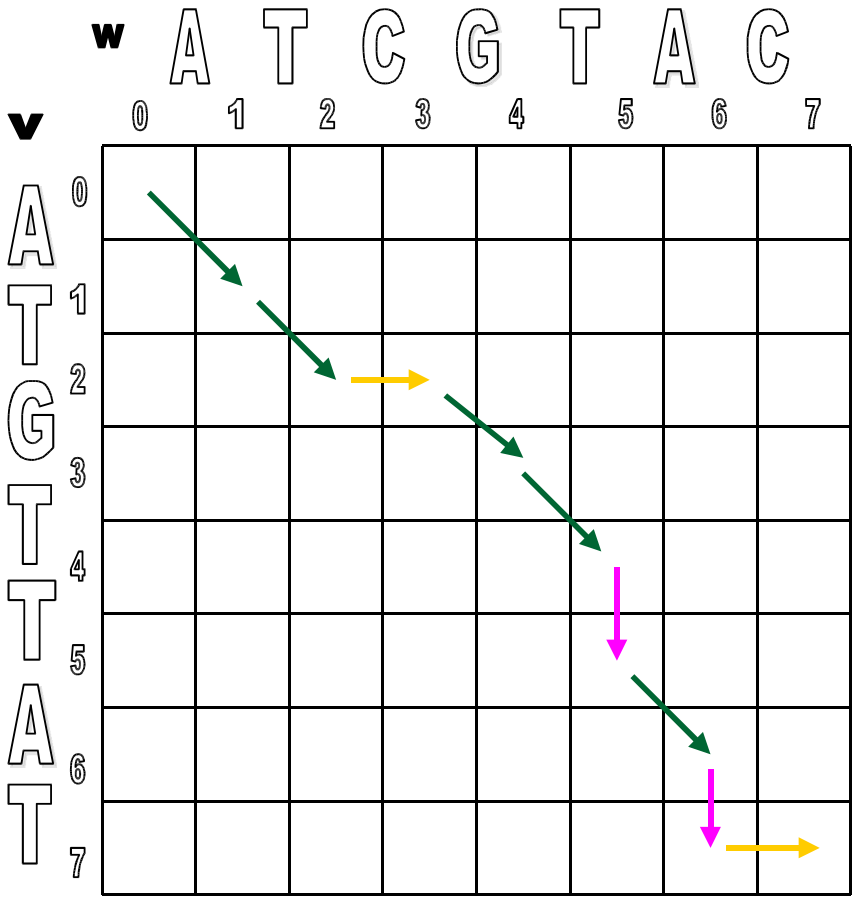
New Alignment

```
      01223456 77
v =   AT_GTTAT_
w =   ATCG_TA_C
      01234456 67
```

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Alignment as a path in edit graph




```
        0122345677
v =     AT_GTTAT_
w =     ATCGT_A_C
        0123455667
```

(0,0), (1,1), (2,2), (2,3), (3,4), (4,5), (5,5), (6,6), (7,6), (7,7)

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

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 dynamic programming, which we've just seen.

Dynamic programming is based on premise of computing 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.)

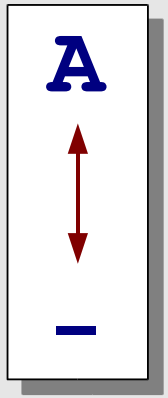
http://fens.sabanciuniv.edu/msie/operations_research_50_years/anniversary/or50/1526-5463-2002-50-01-0048.pdf

Source: Lehigh Uni. - Daniel Lopresti

Sequence alignment ...

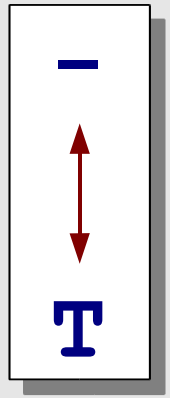
First some ground-rules.

Legal:



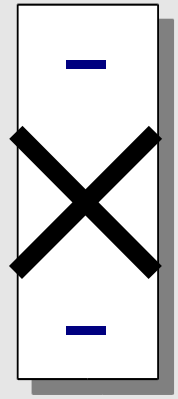
deletion

Legal:

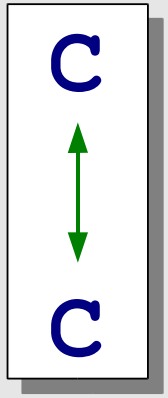


insertion

Not legal:

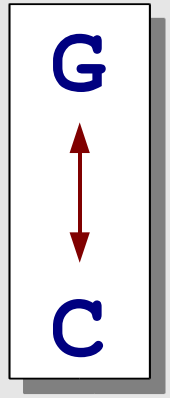


Legal:



match

Legal:



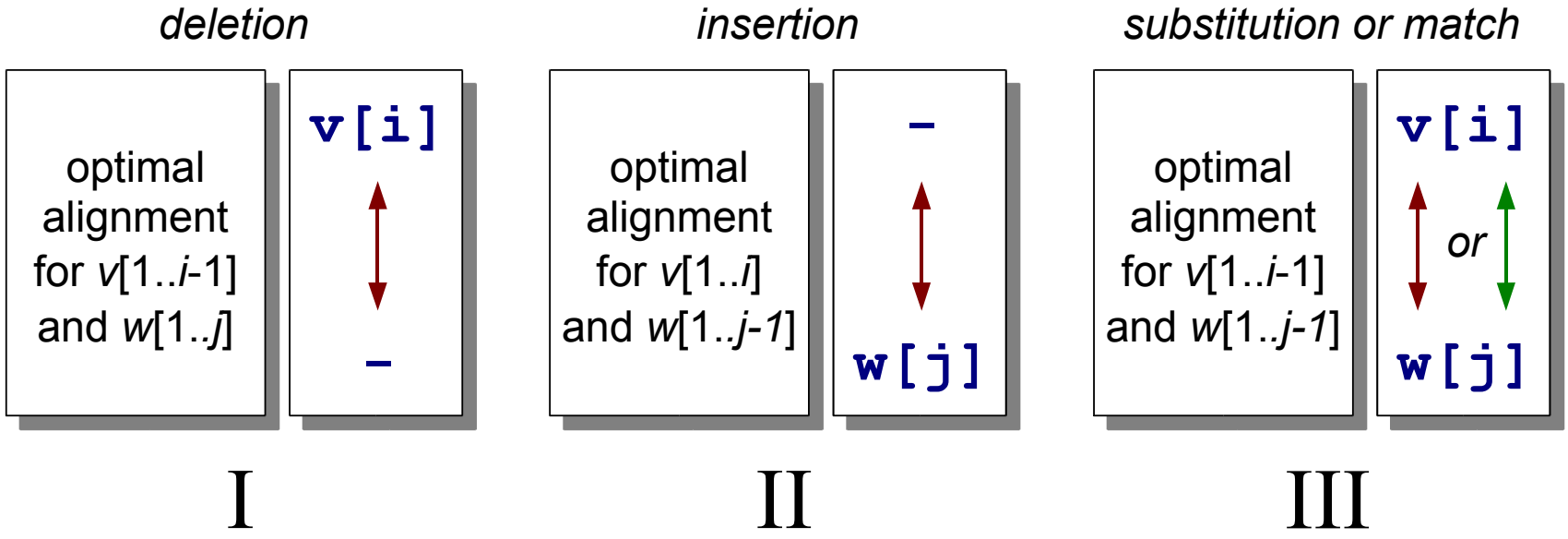
mismatch

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Sequence comparison: the basic algorithm

Given two sequences *v* and *w*, consider what's required to compute optimal alignment for prefixes *v*[1..*i*] and *w*[1..*j*]. Based on our rules for alignments, there are three possible cases:



So, assuming we've already computed solutions for all shorter prefixes, we can compute the alignment for *v*[1..*i*] and *w*[1..*j*].

Source: Lehigh Uni. - Daniel Lopresti

Sequence comparison: the basic algorithm

Conceptually, this might look something like this:

optimal alignment at $v[1..i]$ and $w[1..j]$ = max

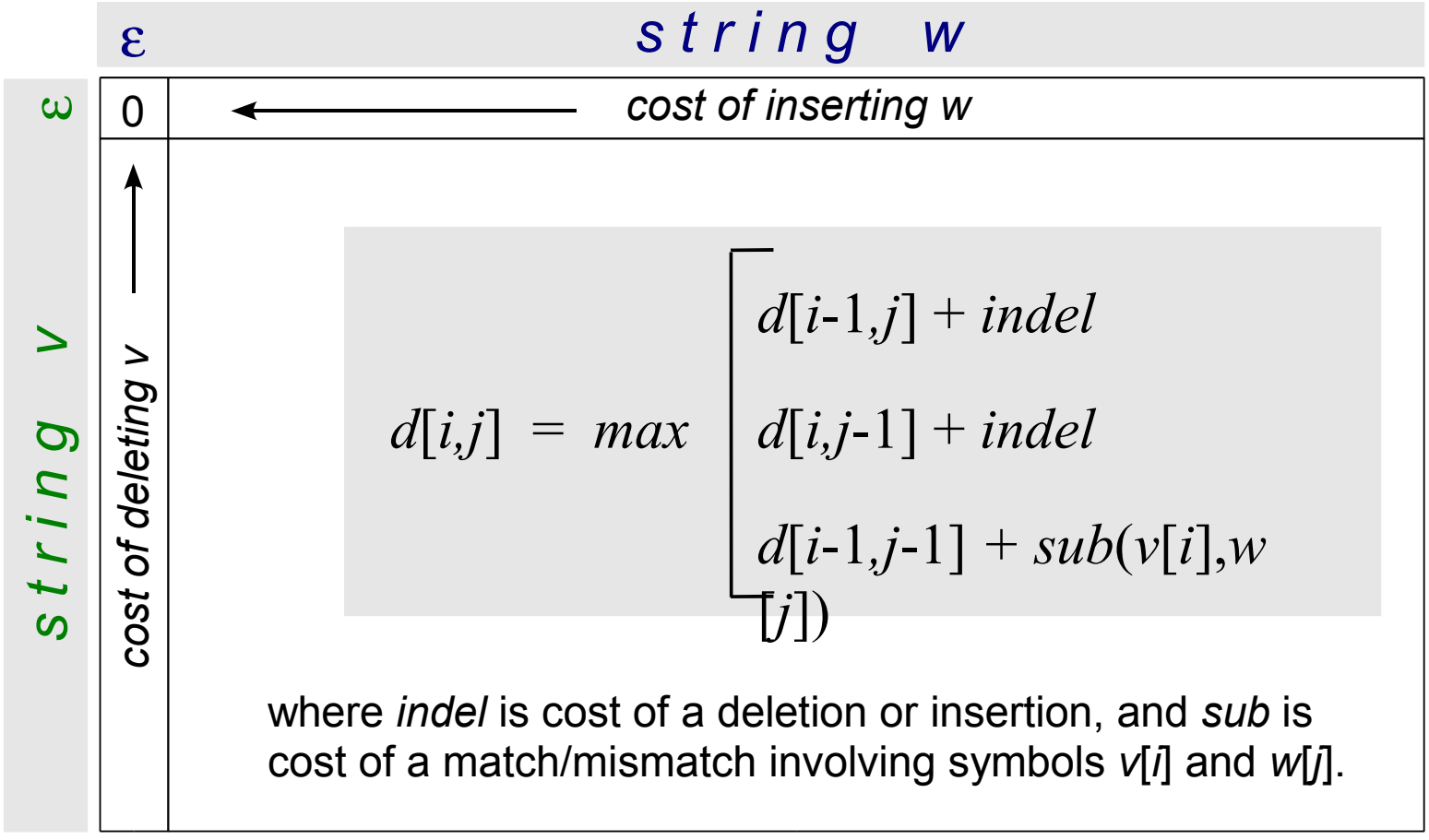
- optimal alignment at $v[1..i\text{-}1]$ and $w[1..j]$ + cost of deleting $v[i]$
- optimal alignment at $v[1..i]$ and $w[1..j\text{-}1]$ + cost of inserting $w[j]$
- optimal alignment at $v[1..i\text{-}1]$ and $w[1..j\text{-}1]$ + cost of matching $v[i]$ and $w[j]$

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

Source: Lehigh Uni. - Daniel Lopresti

Sequence comparison: the basic algorithm

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



Source: Lehigh Uni. - Daniel Lopresti

Sequence comparison: the basic algorithm

Stated more generally, say that our two sequences are:

$$v[1]v[2]v[3]...v[m] \qquad w[1]w[2]w[3]...w[n]$$

Then:

$$d[0,0] = 0$$

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

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

initial conditions

And:

$$d[i,j] = max \begin{cases} d[i\text{-}1,j] + c_{del}(v[i]) \\ d[i,j\text{-}1] + c_{ins}(w[j]) \\ d[i\text{-}1,j\text{-}1] + c_{sub}(v[i], w[j]) \end{cases} \qquad 1 \le i \le m, \ 1 \le j \le n$$

recurrence

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

Source: Lehigh Uni. - Daniel Lopresti

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

Source: Lehigh Uni. - Daniel Lopresti

Algorithm for computing edit distance

EditDistance1(v, w)

for $i \leftarrow 1$ **to** n

$d_{i,0} \leftarrow d_{i-1,0} + c_{del}(v_i)$

for $j \leftarrow 1$ **to** m

$d_{0,j} \leftarrow d_{0,j-1} + c_{ins}(w_j)$

for $i \leftarrow 1$ **to** n

for $j \leftarrow 1$ **to** m

$$d_{i,j} \leftarrow \max \begin{cases} d_{i-1,j} + c_{del}(v_i) \\ d_{i,j-1} + c_{ins}(w_j) \\ d_{i-1,j-1} + c_{sub}(v_i, w_j) \end{cases}$$

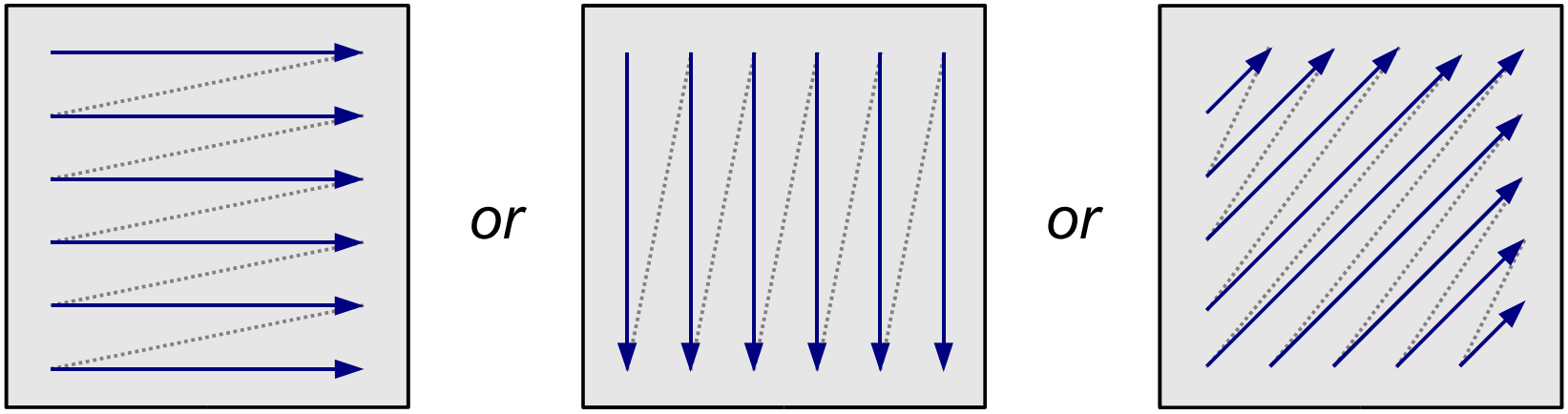
return $(d_{n,m})$

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Sequence comparison: some observations

Computation can progress in a number of ways:



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

$$v[1]v[2]v[3]...v[m] \qquad w[1]w[2]w[3]...w[n]$$

What is the computation time? $O(mn)$

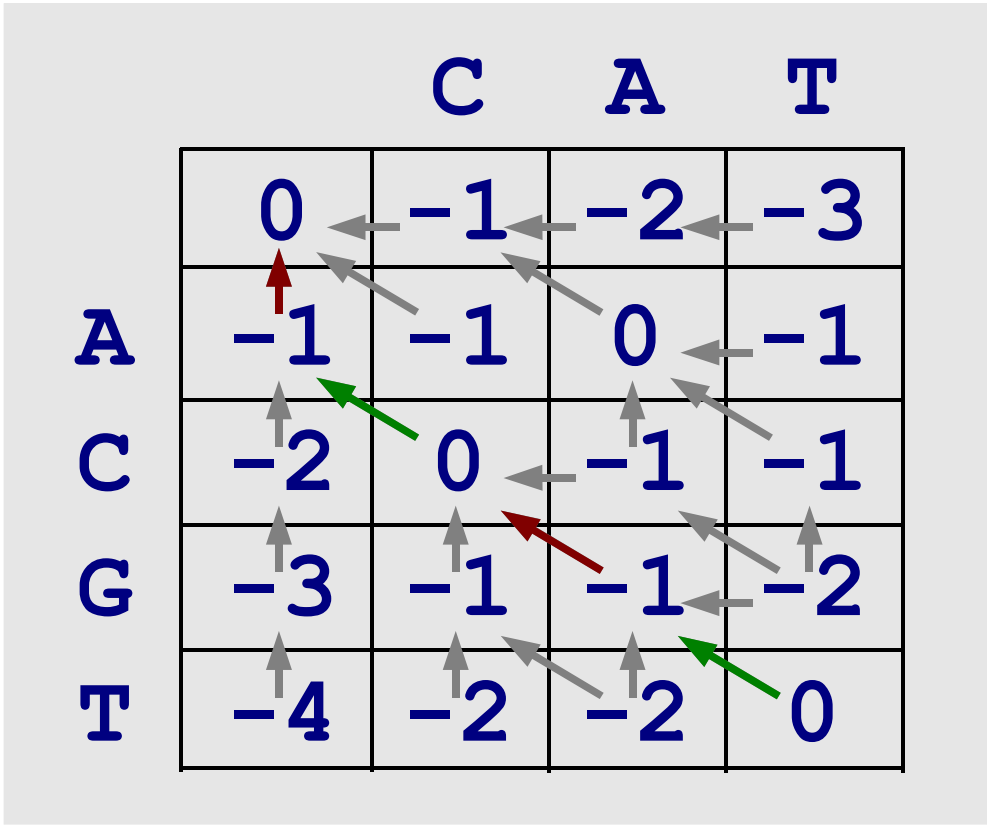
How much memory (storage) is required? $O(mn)$

Source: Lehigh Uni. - Daniel Lopresti

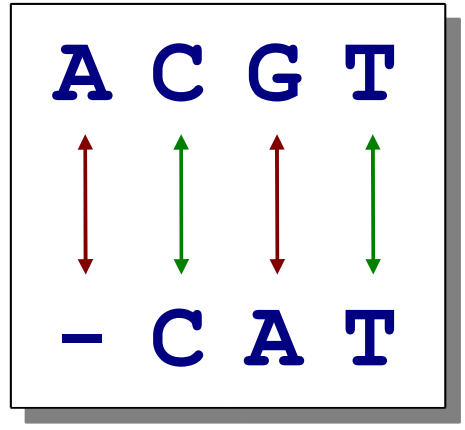
Sequence comparison: getting an alignment

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

By keeping track of optimal decisions made during algorithm,



... and then tracing back optimal path.



(May not be unique).

Source: Lehigh Uni. - Daniel Lopresti

Maintaining the traceback

EditDistance1(v, w)

...

for i ← 1 **to** n

  for j ← 1 **to** m

$$d_{i,j} \leftarrow \max \begin{cases} d_{i-1,j} + c_{del}(v_i) \\ d_{i,j-1} + c_{ins}(w_j) \\ d_{i-1,j-1} + c_{sub}(v_i, w_j) \end{cases}$$

$$b_{i,j} \leftarrow \begin{cases} \uparrow & \text{if } d_{i,j} = d_{i-1,j} + c_{del}(v_i) \\ \leftarrow & \text{if } d_{i,j} = d_{i,j-1} + c_{ins}(w_j) \\ \nwarrow & \text{if } d_{i,j} = d_{i-1,j-1} + c_{sub}(v_i, w_j) \end{cases}$$

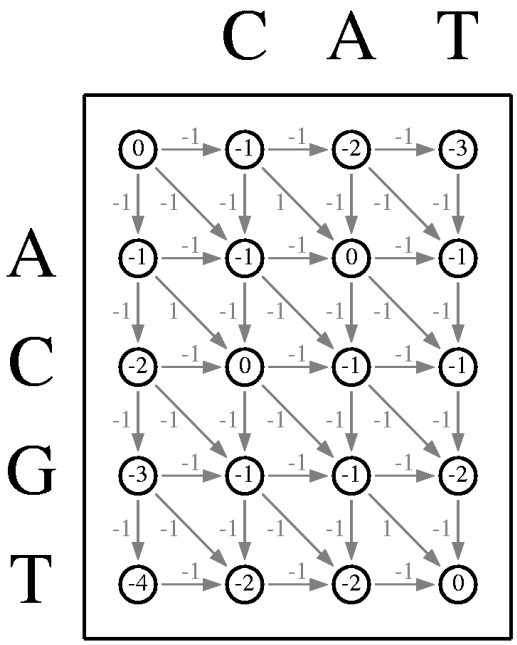
return $(d_{n,m}, b)$

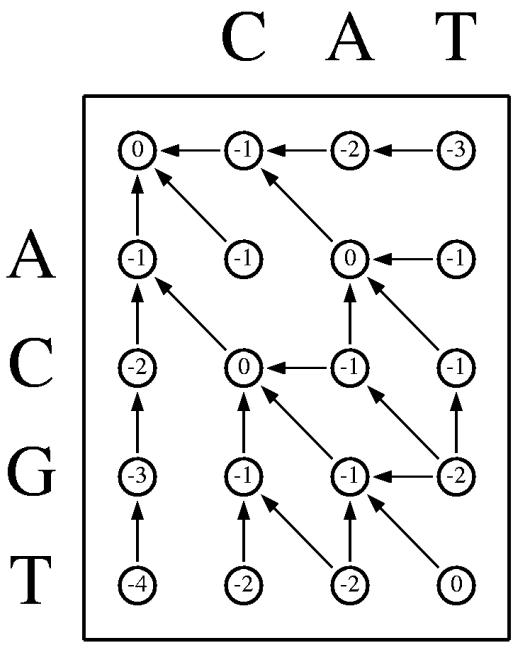
http://www.bioalgorithms.info

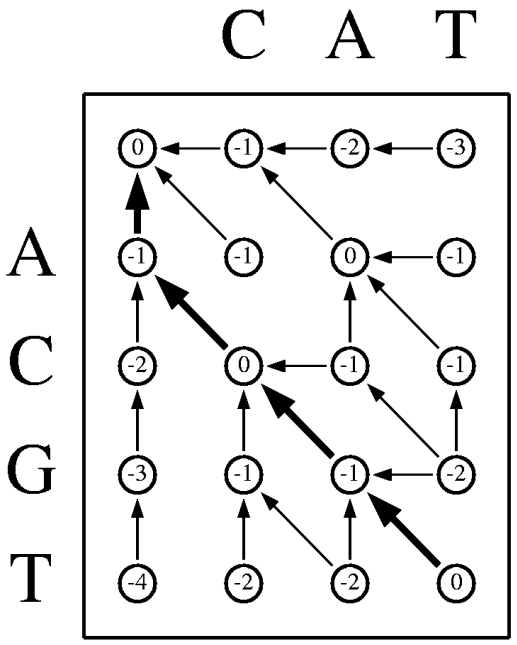
Source: Lehigh Uni. - Daniel Lopresti

More examples

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



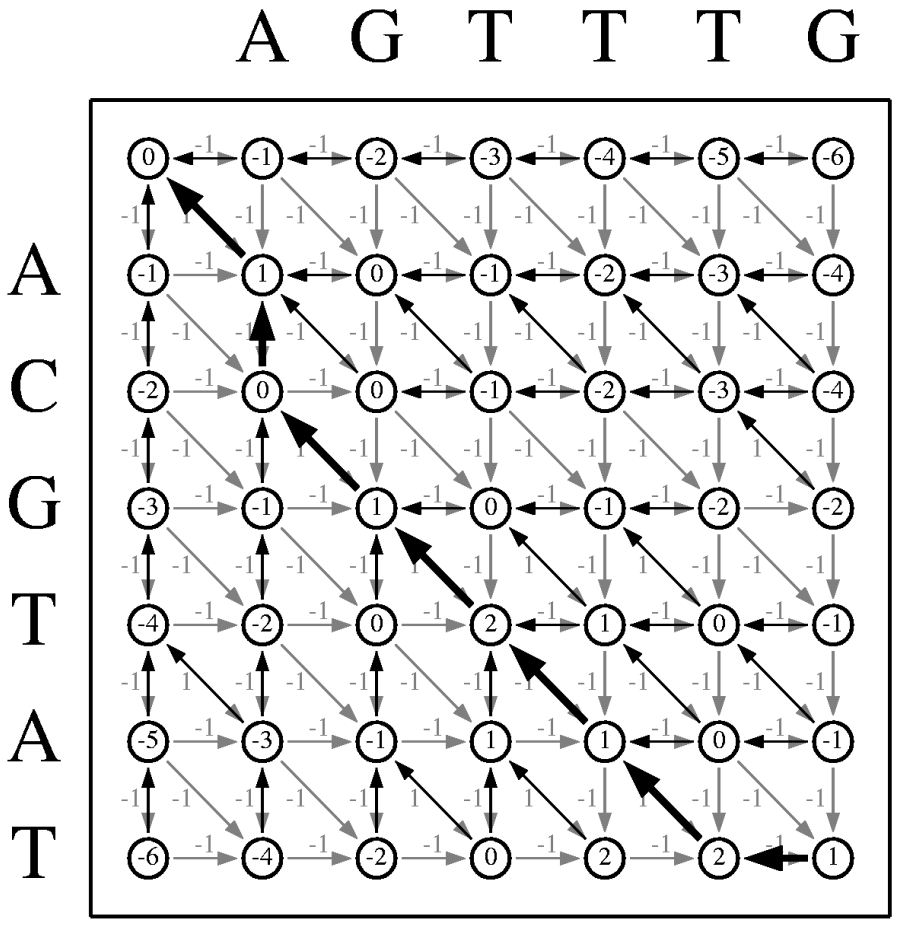




Source: Lehigh Uni. - Daniel Lopresti

More examples

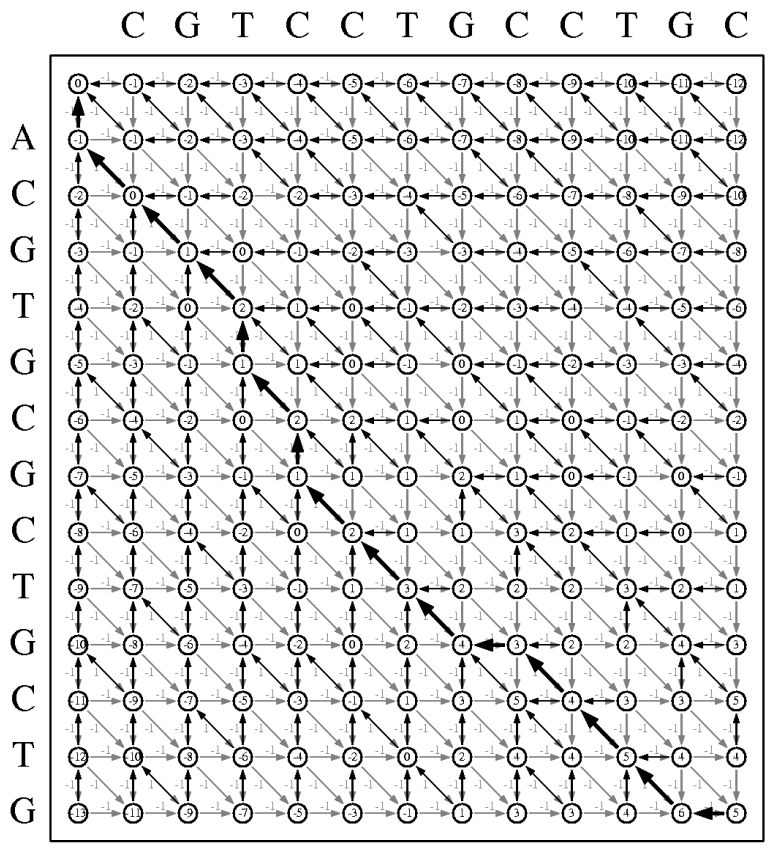
Comparing **ACGTAT** vs. **AGTTTG**:



Source: Lehigh Uni. - Daniel Lopresti

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

Source: Lehigh Uni. - Daniel Lopresti

From LCS to alignment: change scoring

The Longest Common Subsequence (LCS) problem – the simplest form of sequence alignment – allows only insertions and deletions (no mismatches).

- In LCS Problem, we scored 1 for matches and 0 for indels.
- Now penalize indels and mismatches with negative scores.
- Simplest scoring schema:
  - +1 = match premium
  - –μ = mismatch penalty
  - –σ = indel penalty

score = #matches – μ (#mismatches) – σ (#indels)

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

The Global Alignment Problem.

Find the best alignment between two strings under a given scoring schema.

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

Output: Alignment of maximum score.

$$d[i,j] = \max \begin{cases} d[i-1,j] - \sigma & \text{indel} \\ d[i,j-1] - \sigma & \text{indel} \\ d[i-1,j-1] + 1 & \text{if } v[i]=w[j] \text{ match} \\ d[i-1,j-1] - \mu & \text{if } v[i]\neq w[j] \text{ mismatch} \end{cases}$$

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Scoring matrices

To generalize scoring, consider a (4+1) x (4+1) matrix δ.

In the case of an amino acid sequence alignment, the scoring matrix would be (20+1) x (20+1). The extra 1 is to include score for comparison of a gap character "–".

This will simplify the algorithm as follows:

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

http://www.bioalgorithms.info

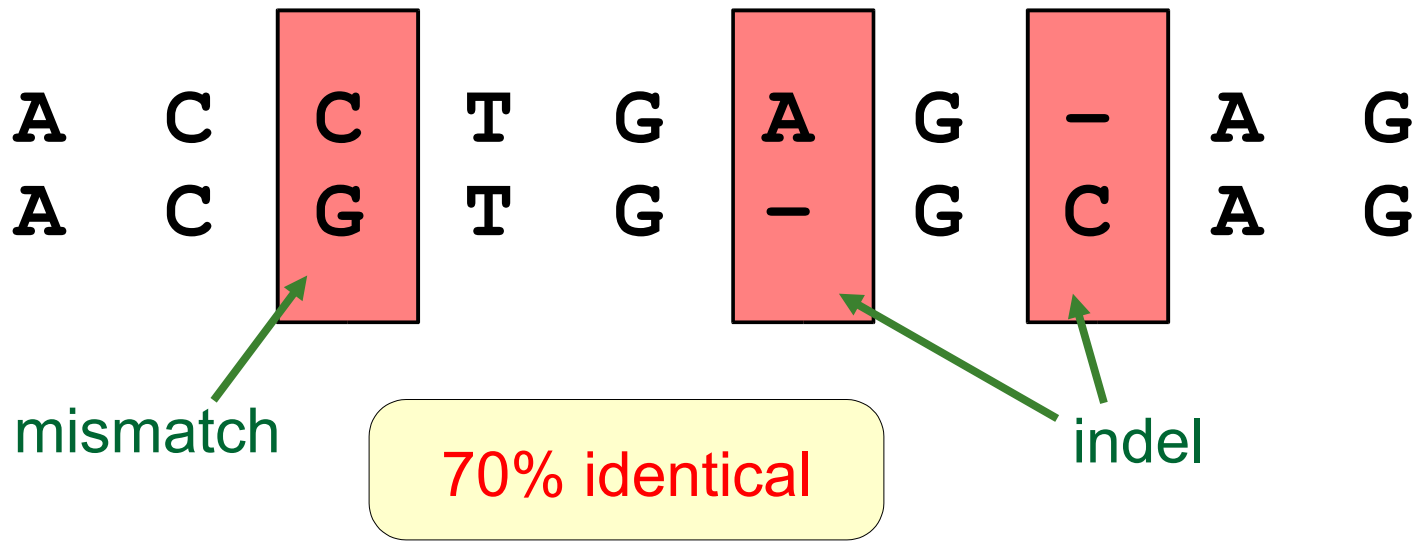
Source: Lehigh Uni. - Daniel Lopresti

Measuring similarity

Measuring the extent of similarity between two sequences:

- Based on percent sequence identity.
- Based on conservation.

Percent sequence identity: extent to which two nucleotide or amino acid sequences are invariant.



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Making 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.

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

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

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Conservation

Amino acid changes that tend to preserve the physico-chemical properties of the original residue:

- Polar to polar

  aspartate → glutamate

- Nonpolar to nonpolar

  alanine → valine

- Similarly-behaving residues

  leucine → isoleucine

Amino acid substitution matrices include PAM and BLOSUM.

DNA is less conserved than protein sequences, so it is less effective to compare coding regions at nucleotide level.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

PAM matrices

One important measure of mutual substitution used by biologists is *Point Accepted Mutations* (or *Percent of Accepted Mutations*) matrices, often called *PAM* matrices.

PAM matrices are computed in terms of evolutionary "units." E.g., a 1-PAM matrix reflects evolution leading to an average of one mutation per hundred amino acids. Higher level PAM matrices are computed from this.

To build a 1-PAM matrix, M^1, we need:

- list of accepted mutations, $a \rightarrow b$,
- probability of occurrence for each amino acid, p_a.

This allows us to compute mutability of amino acid *a* into *b* as:

$$M^1_{ab} = Pr(a \rightarrow b) = Pr(a \rightarrow b \mid a \text{ changed}) Pr(a \text{ changed})$$

Source: Lehigh Uni. - Daniel Lopresti

PAM and scoring matrices

Given the 1-PAM matrix, we can compute mutability of amino acid *a* into *b* though *k* units of evolution as a *k*-PAM matrix:

$$M^k_{ab} \text{ is the } (a, b) \text{ entry in } \underbrace{M^1 M^1 \ \ldots \ M^1}_{k \text{ times}}$$

PAM matrices are used to build scoring matrices whose entries measure the probability an amino acid is present due to a mutation versus a random event:

$$\frac{M_{ab}}{p_b}$$

For convenience, actual score is 10 times logarithm of ratio.

Source: Lehigh Uni. - Daniel Lopresti

250-PAM matrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	12															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-4	-4	-4	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	-1	-1	-3	0	-1	-1	0	-2	-3	-1	-2	-5	6					
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

Source: Lehigh Uni. - Daniel Lopresti

BLOSUM

- Blocks Substitution Matrix.
- Scores derived from frequencies of substitutions in blocks of local alignments in related proteins.
- Matrix name indicates evolutionary distance.
- BLOSUM62 was created using sequences sharing no more than 62% identity.

Blosum50 Scoring Matrix

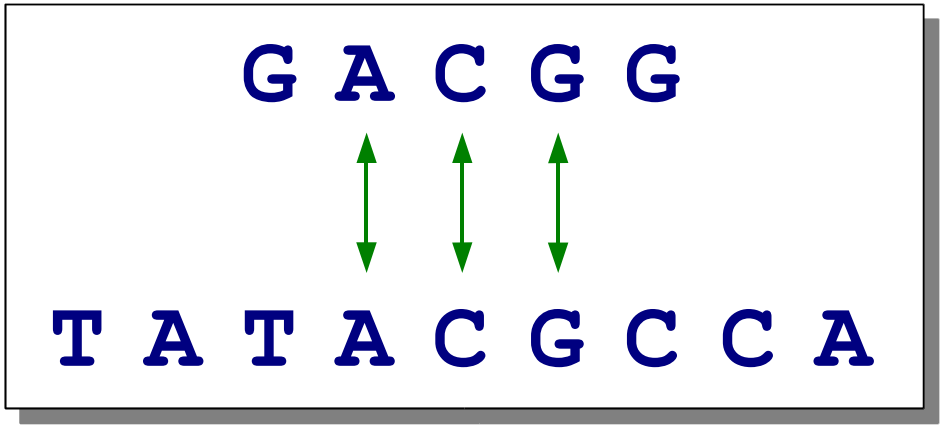
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0	-2	-1	-1	-5
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3	-1	0	-1	-5
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3	4	0	-1	-5
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4	5	1	-1	-5
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-3	-3	-2	-5
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3	0	4	-1	-5
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3	1	5	-1	-5
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4	-1	-2	-2	-5
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4	0	0	-1	-5
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4	-4	-3	-1	-5
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1	-4	-3	-1	-5
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3	0	1	-1	-5
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1	-3	-1	-1	-5
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1	-4	-4	-2	-5
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3	-2	-1	-2	-5
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2	0	0	-1	-5
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0	0	-1	0	-5
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3	-5	-2	-3	-5
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1	-3	-2	-1	-5
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5	-4	-3	-1	-5
B	-2	-1	4	5	-3	0	1	-1	0	-4	-4	0	-3	-4	-2	0	0	-5	-3	-4	5	2	-1	-5
Z	-1	0	0	1	-3	4	5	-2	0	-3	-3	1	-1	-4	-1	0	-1	-2	-2	-3	2	5	-1	-5
X	-1	-1	-1	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-1	0	-3	-1	-1	-1	-1	-1	-5
*	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	1

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Local alignment

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.

Source: Lehigh Uni. - Daniel Lopresti

Local vs. global alignment

The Global Alignment Problem tries to find longest path between vertices (0, 0) and (*n*, *m*) in edit graph.

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

In edit graph with negatively-scored edges, local alignmet may score higher than global alignment.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Local vs. global alignment

Global alignment:

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

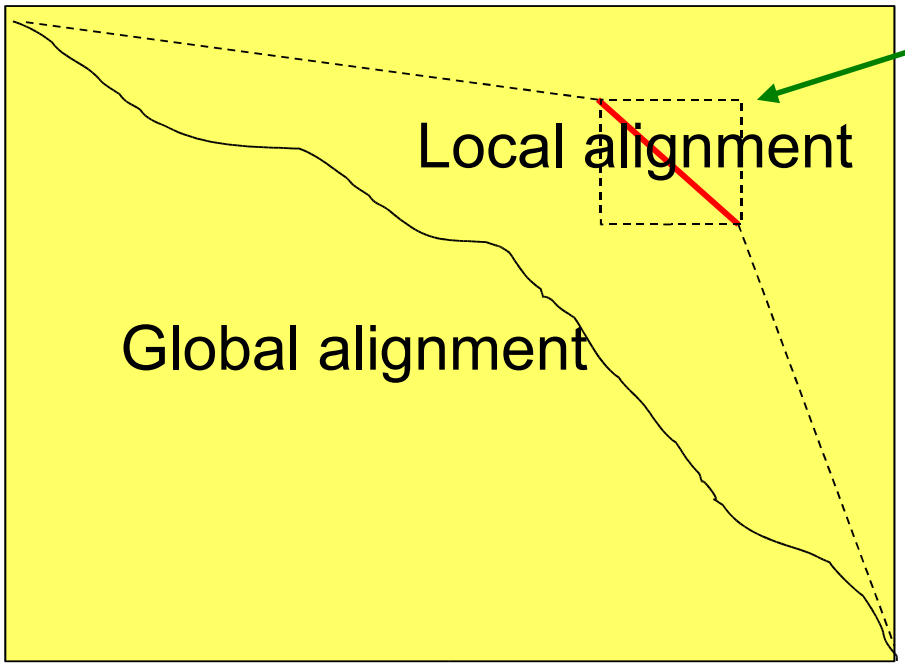
Local alignment – better alignment to find conserved segment:

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

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Local vs. global alignment



Compute a "mini" global alignment to get local

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Local alignments: 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 between species.
- A global alignment would not find the homeodomain because it would try to align the ENTIRE sequence.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

The Local Alignment Problem

The Local Alignment Problem.

Find the best local alignment between two strings.

Input: Strings v and w and a scoring matrix δ.

Output: Alignment of substrings of v and w whose score is maximum over all possible alignments of all possible substrings.

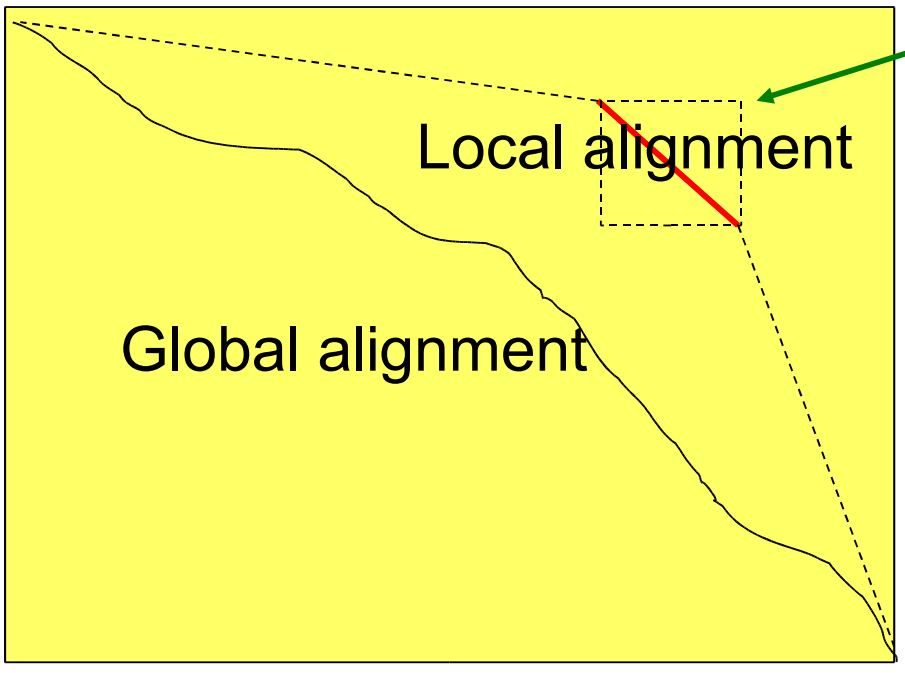
Hmmm ... run time looks to be $O(n^4)$:

- In 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.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Local alignment: example



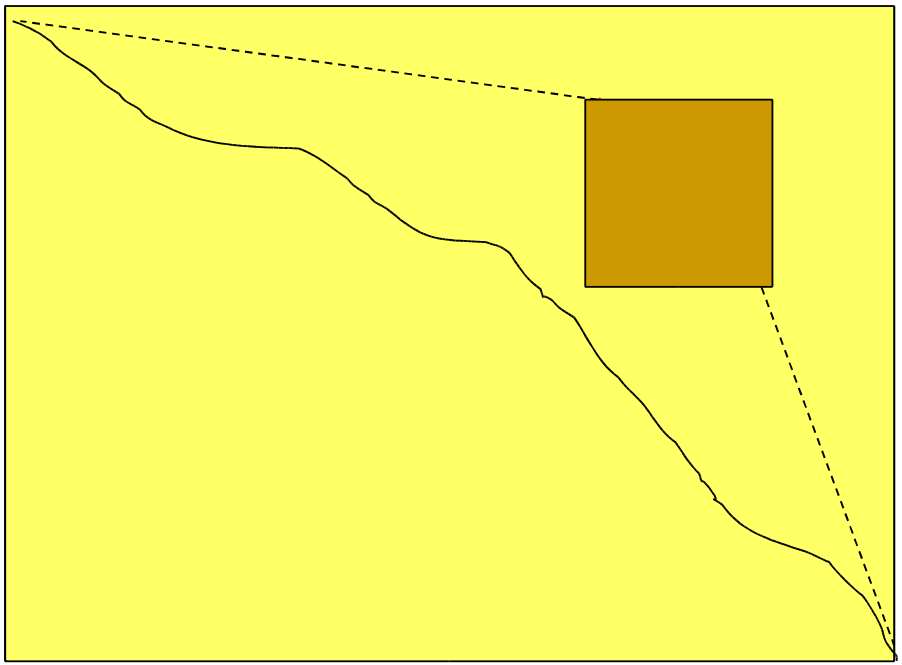
Compute a "mini" global alignment to get local

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

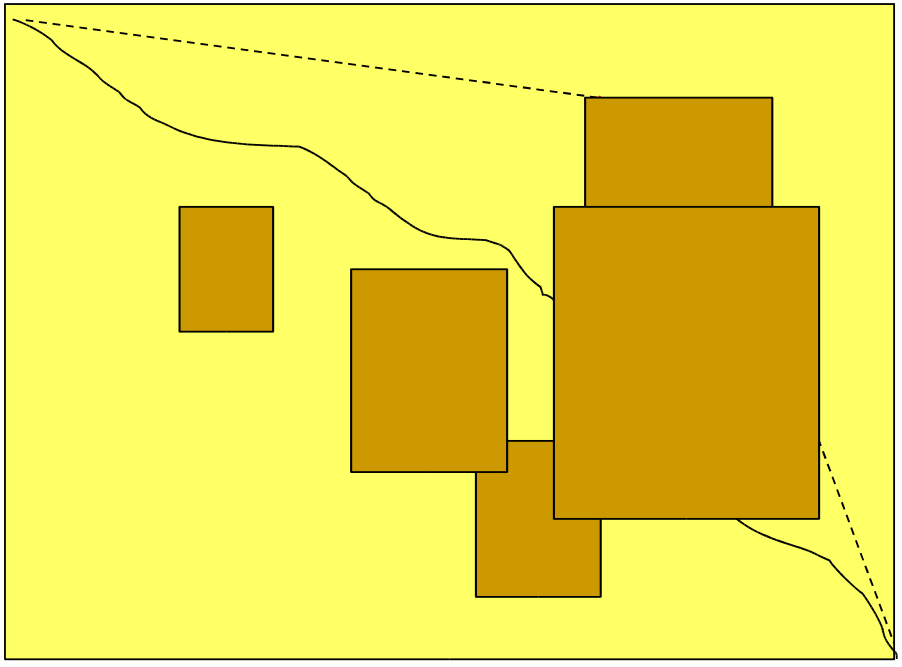
Local alignment: example



http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Local alignment: example

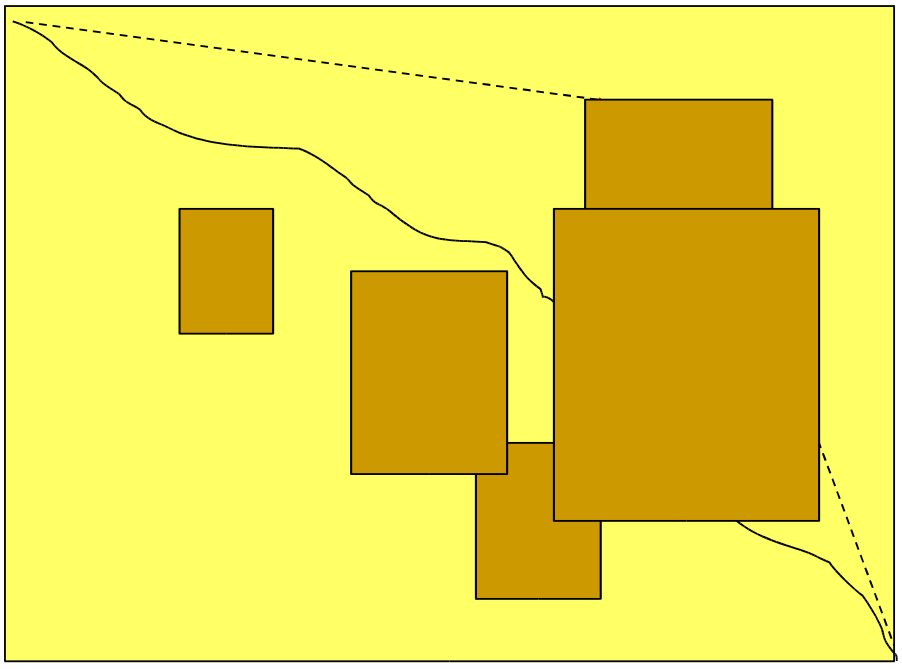


http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

Local alignment: running time

This can be remedied by giving "*free rides*."



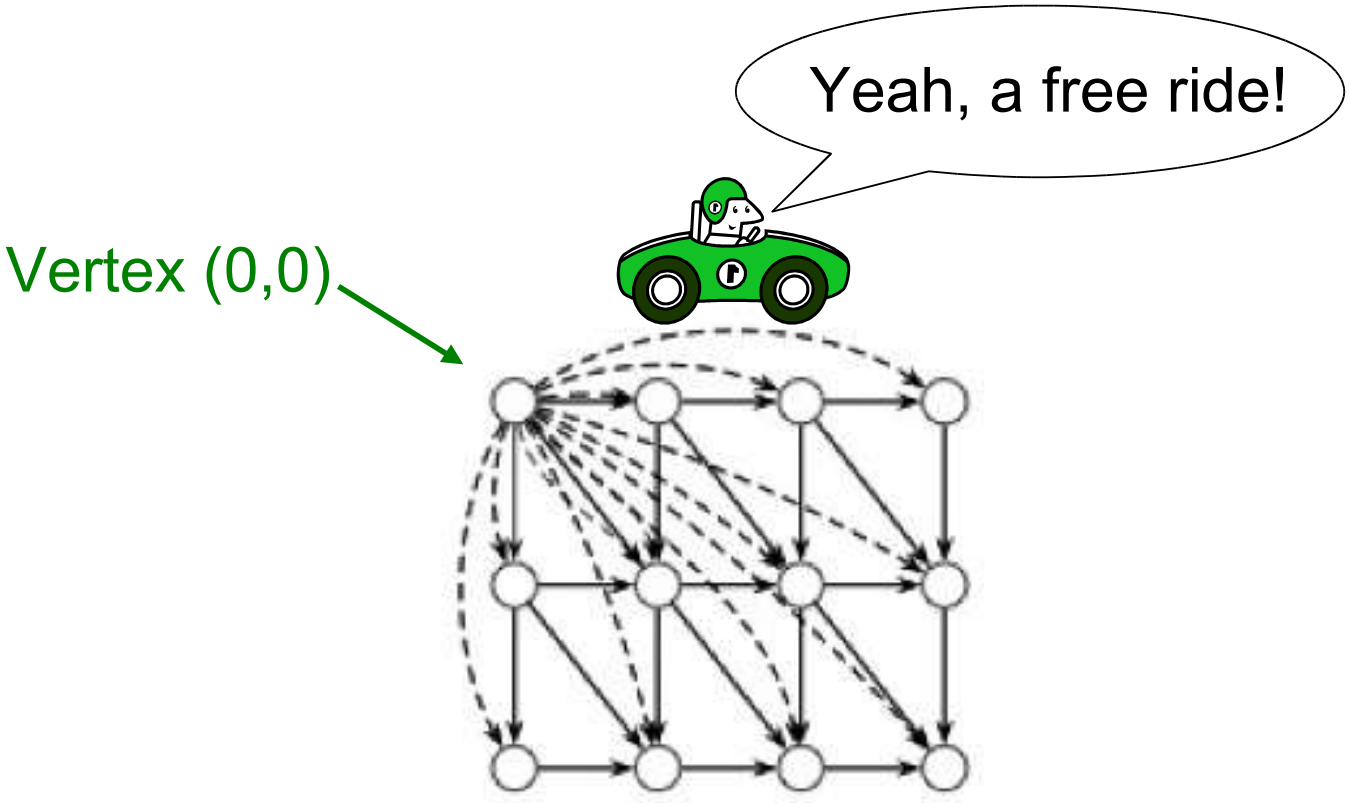
Run time looks like $O(n^4)$:

- In 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.

http://www.bioalgorithms.info

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY



Dashed edges represent free rides from (0,0) to other nodes.

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Local comparison

As before, say that our two sequences are:

$$v[1]v[2]v[3]...v[m] \qquad w[1]w[2]w[3]...w[n]$$

Then:

$$a[0,0] = 0$$
$$a[i,0] = 0 \qquad 1 \le i \le m$$
$$a[0,j] = 0 \qquad 1 \le j \le n$$

initial conditions

And:

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

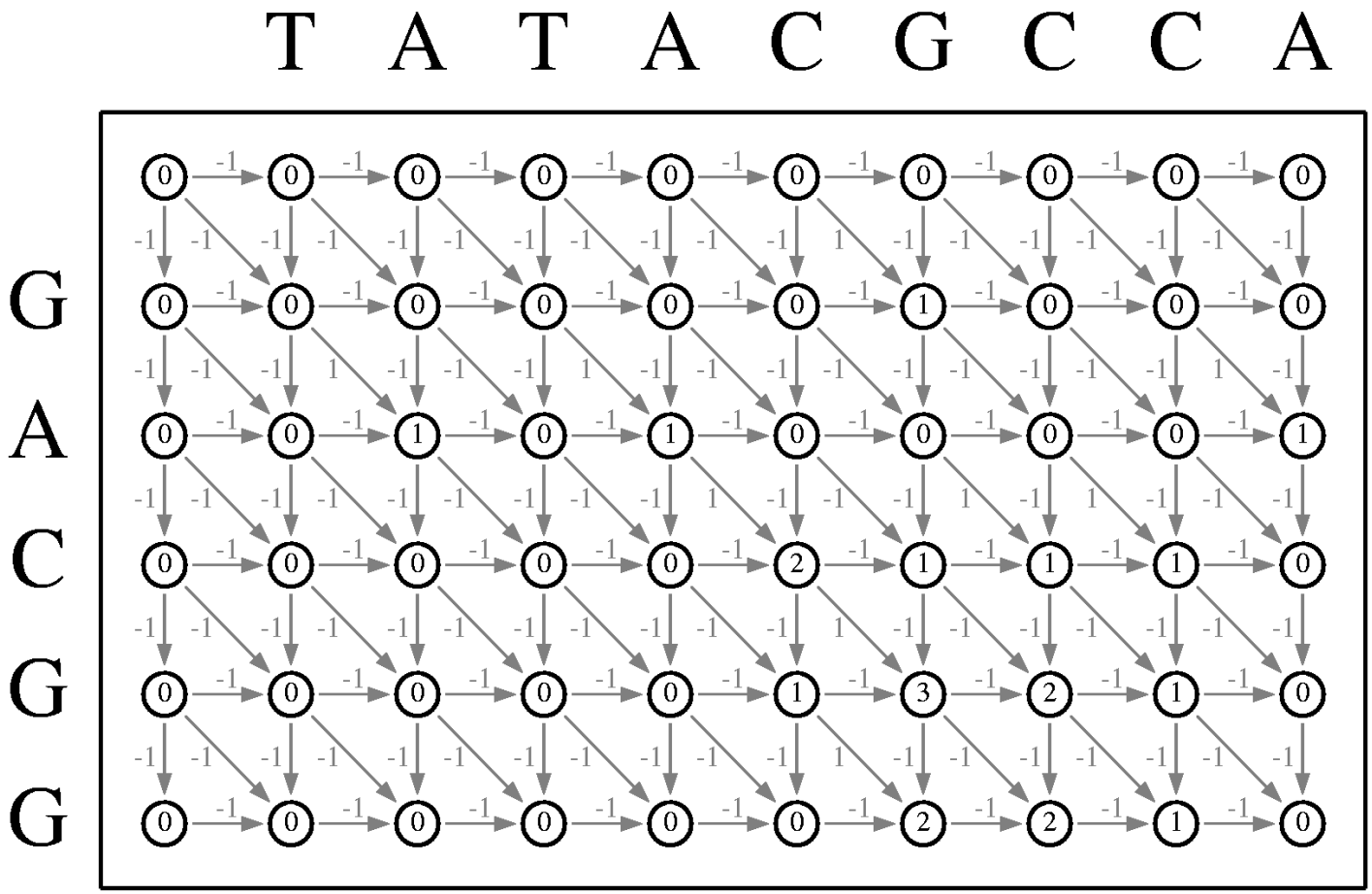
recurrence

Power of ZERO: only change from original recurrence for global alignment. There is one "free ride" edge entering every vertex

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

Source: Lehigh Uni. - Daniel Lopresti

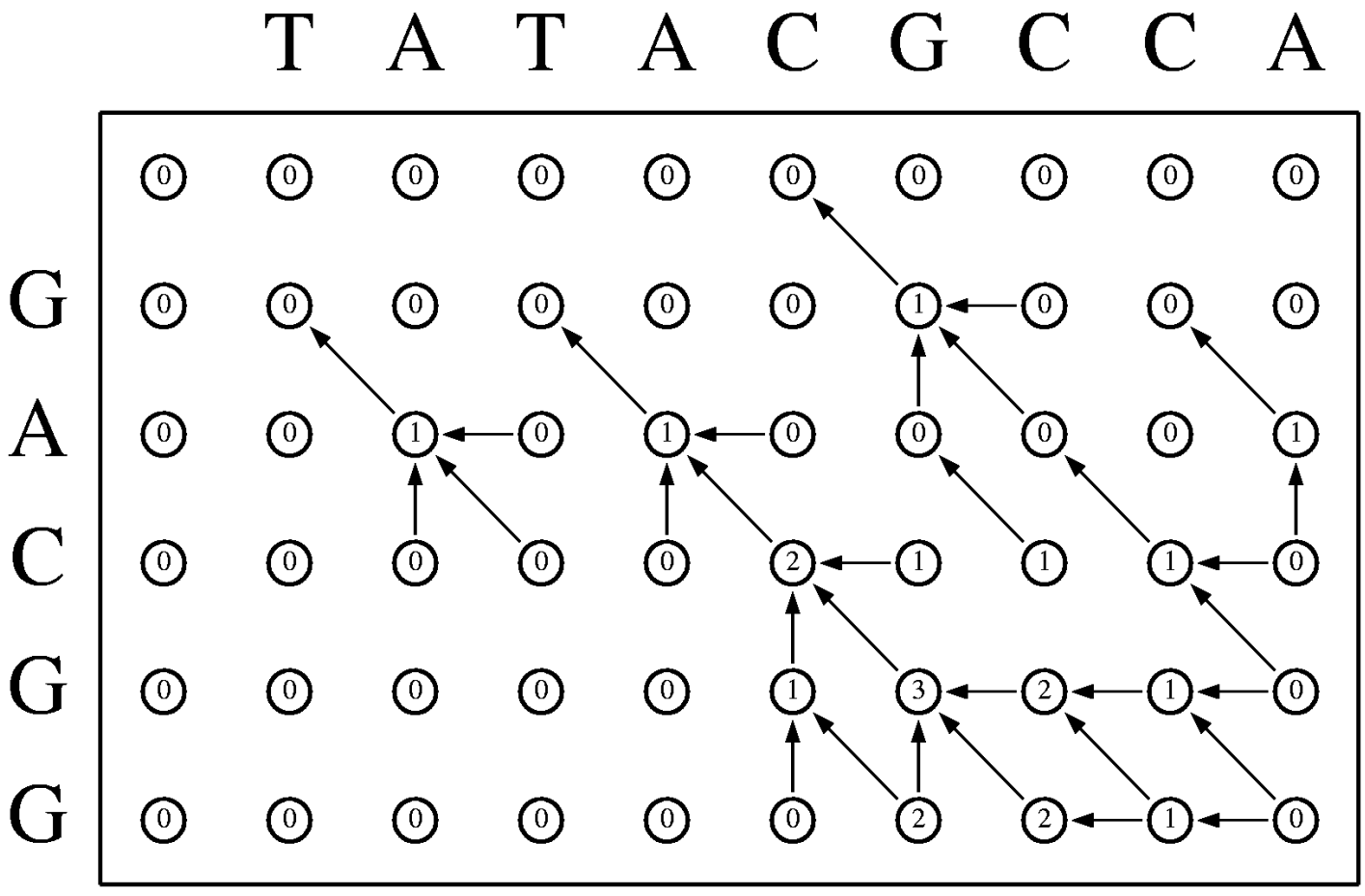
Local comparison



Source: Lehigh Uni. - Daniel Lopresti

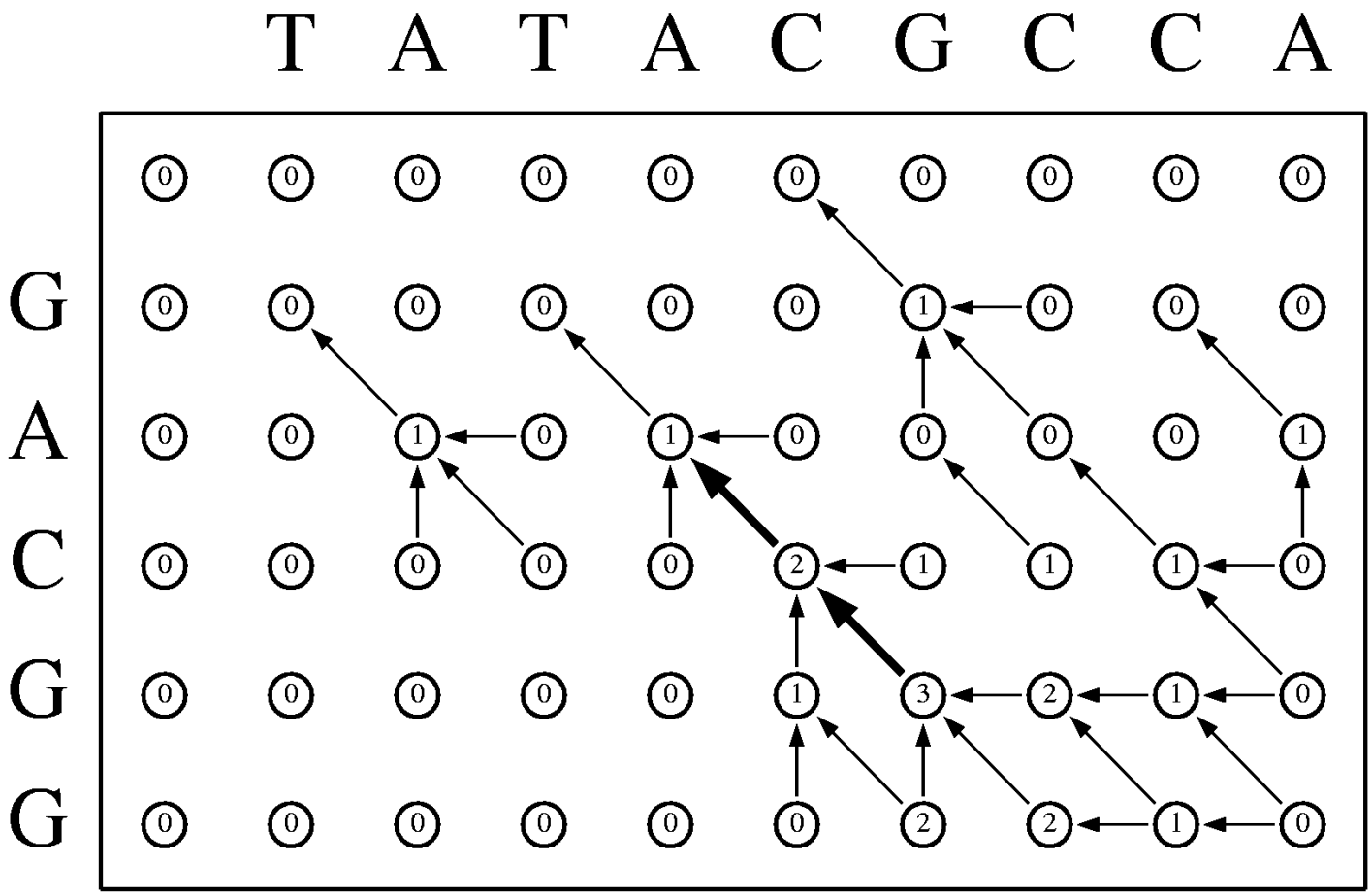
LEHIGH UNIVERSITY

Local comparison



Source: Lehigh Uni. - Daniel Lopresti

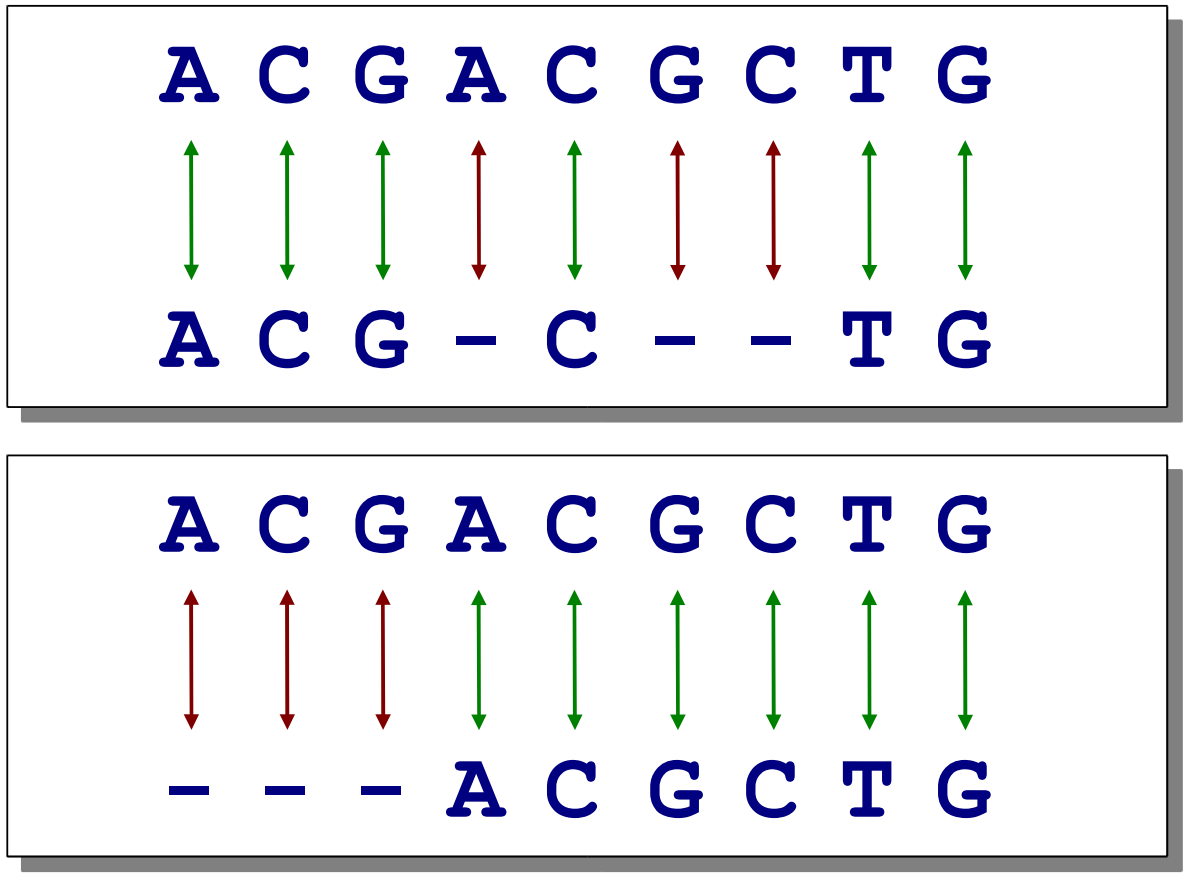
Local comparison



Source: Lehigh Uni. - Daniel Lopresti

General gap penalties

What is the difference between these two alignments?



While scores are the same, second is preferable (fewer gaps).

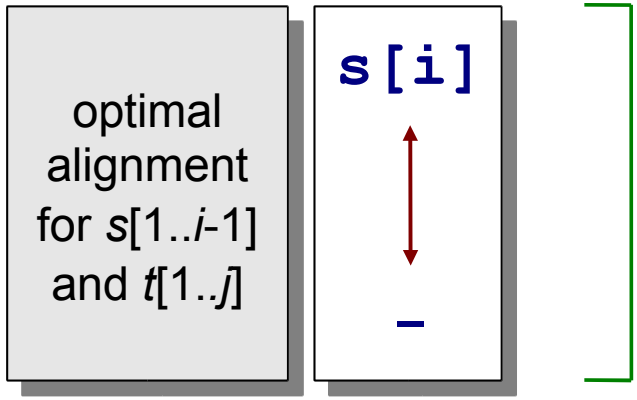
Source: Lehigh Uni. - Daniel Lopresti

General gap penalties

A *gap* is a series of $k > 1$ consecutive spaces. All else being equal, we prefer one gap of k spaces over k gaps of one space.

Let $w(k)$ be our gap penalty. Before, we had $w(k) = bk$ where b was a constant (the deletion and insertion cost). This is an *additive* gap penalty, which may not be true in general.

Assumptions we used to partition problem no longer valid. E.g.,



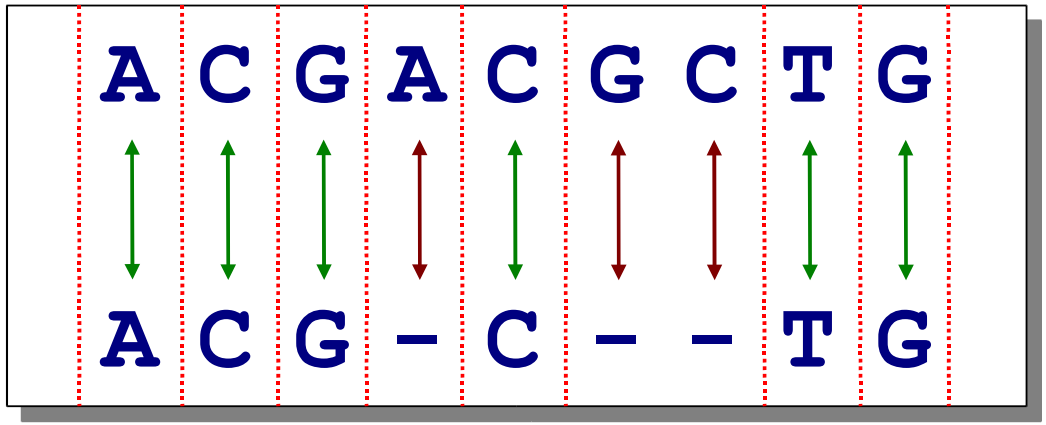
Cost of this alignment depends on whether or not optimal alignment for *s* [1..*i*-1] and *t*[1..*j*] has gap in *t* and, if so, its length.

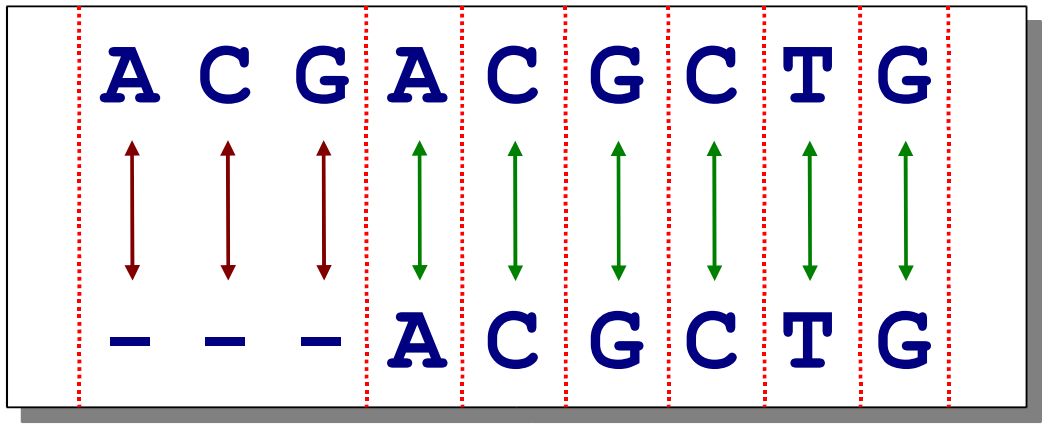
So we can't just call this: $a[i-1, j] + c_{\text{del}} s([i])$

Source: Lehigh Uni. - Daniel Lopresti

General gap penalties

We must now think of computing alignments in terms of *blocks*, maximal substrings where score additivity still holds.





6 matches + w(1) + w(2)

6 matches + w(3)

In general,
$w(3) \neq w(1) + w(2)$

Source: Lehigh Uni. - Daniel Lopresti

Affine gap penalties

Assuming less generality about $w(k)$, we can do better.

Say that $w(k) = h + gk$. This is known as an *affine* gap penalty.

Now we only need to keep track of whether given space is first space in gap (in which case cost is $h + g$), or continuation of existing gap (in which case cost is g).

need to maintain three arrays, a, b, and c:

- $a[i,j]$ assumes $s[i]$ aligns with $t[j]$.
- $b[i,j]$ assumes $t[j]$ aligns with space in s.
- $c[i,j]$ assumes $s[i]$ aligns with space in t.

Source: Lehigh Uni. - Daniel Lopresti

Affine gap penalties

Recurrences:

$$a[i,j] = c_{\text{sub}}(s[i],t[j]) + \max\begin{cases} a[i-1,j-1] \\ b[i-1,j-1] \\ c[i-1,j-1] \end{cases}$$

$$b[i,j] = \max\begin{cases} a[i,j-1]-(h+g) \\ b[i,j-1]-g \\ c[i,j-1]-(h+g) \end{cases}$$

$$c[i,j] = \max\begin{cases} a[i-1,j]-(h+g) \\ b[i-1,j]-(h+g) \\ c[i-1,j]-g \end{cases}$$

Time complexity here is $O(mn)$.

Source: Lehigh Uni. - Daniel Lopresti

LEHIGH UNIVERSITY

Algorithms, programming and data structures in Bioinformatics

Subject	4 Exhaustive Search	5 Greedy Algorithms	6 Dynamic Programming Algorithms	7 Divide-and-Conquer Algorithms	8 Graph Algorithms	9 Combinatorial Pattern Matching	10 Clustering and Trees	11 Hidden Markov Models	12 Randomized Algorithms
Mapping DNA	o								
Sequencing DNA					o				
Comparing Sequences			o	o		o			
Predicting Genes			o						
Finding Signals	o	o						o	o
Identifying Proteins					o				
Repeat Analysis						o			
DNA Arrays					o				
Genome Rearrangements		o							
Molecular Evolution							o		

- Read chapters 8 & 9 for the next class (and 7 if you have time)

Source: An Introduction to Bioinformatics Algorithms, N.C. Jones and P.A. Pevzner, The MIT Press Cambridge, 2004

http://bioinformaticsinstitute.ru/sites/default/files/an_introduction_to_bioinformatics_algorithms_-_jones_pevzner.pdf