6.047/6.878/HST.507
Computational Biology: Genomes, Networks, Evolution

Lecture 2

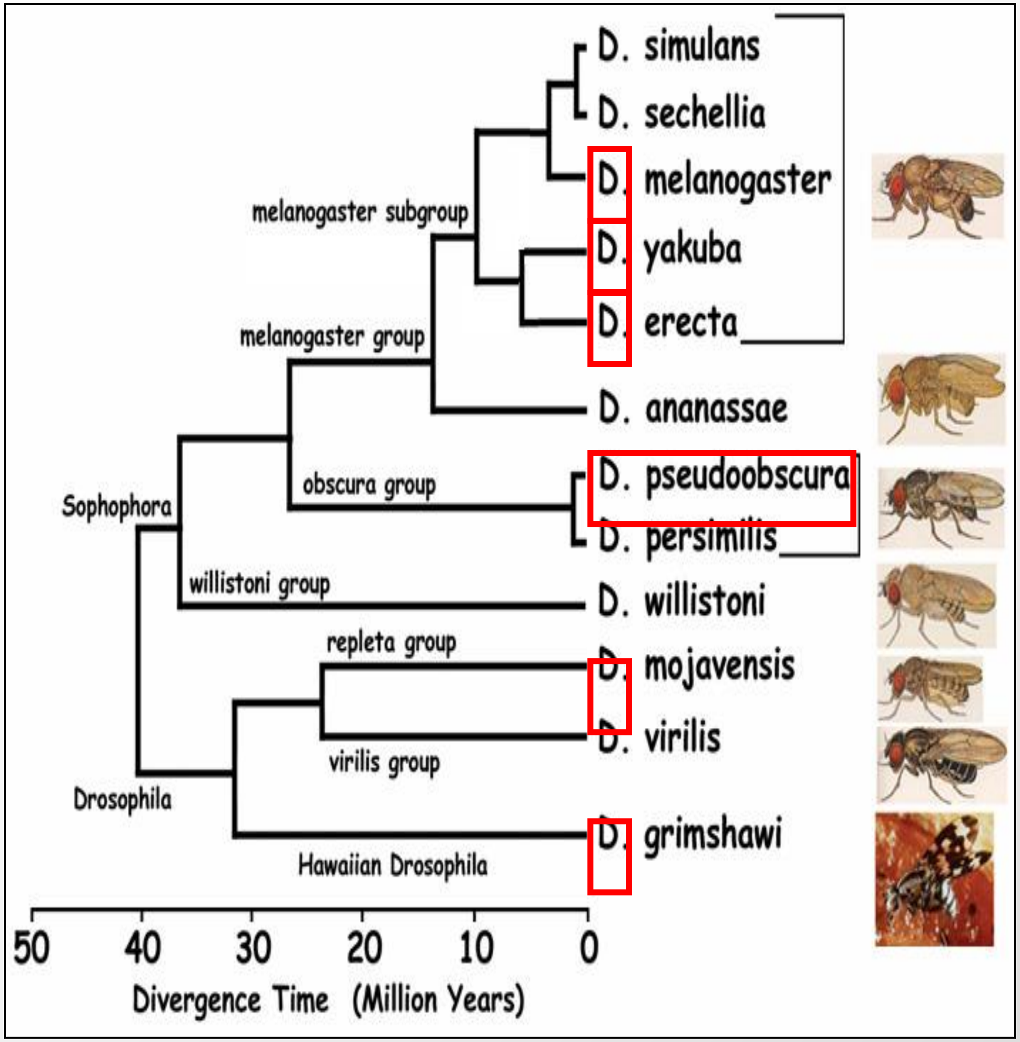
Sequence Alignment and Dynamic Programming

Module 1: Aligning and modeling genomes

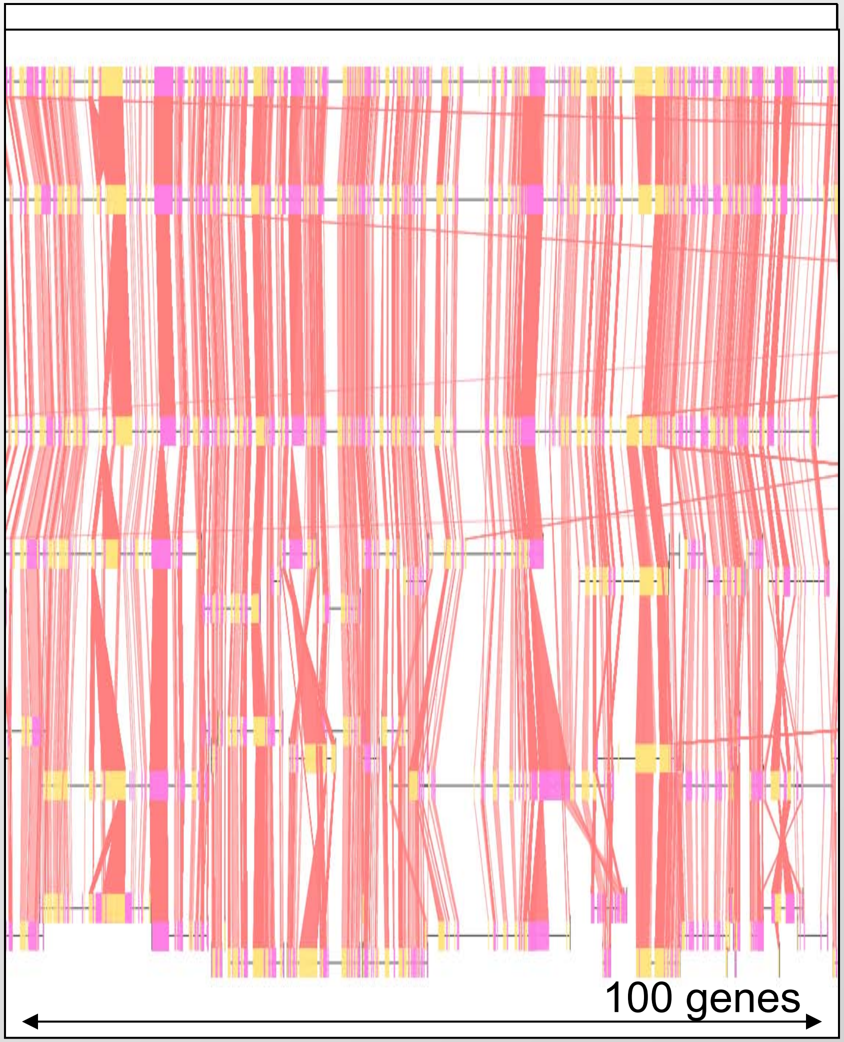
Week	Date	Topic		Lec	Topic	Read*
1	Thu, Sep 10	**Introduction**		L1	Intro: Biology, Algorithms, Machine Learning, Course Overview	1
	Fri, Sep 11			R1	Recitation 1: Biology and Probability Review	
2	Tue, Sep 15	Module I: Aligning and Modeling Genomes	Foundations	L2	Alignment I: Dynamic Programming, Global and local alignment	2
	Thu, Sep 17			L3	Alignment II: Database search, Rapid string matching, BLAST, BLOSUM	3
	Fri, Sep 18			R2	Recitation 2: Deriving Parameters of Alignment, Multiple Alignment	
3	Tue, Sep 22		Frontiers	L4	Hidden Markov Models Part 1: Evaluation/Parsing, Viterbi, Forward algorithms	7
	Thu, Sep 24			L5	Hidden Markov Models Part 2: Posterior Decoding, Learning, Baum-Welch	8
	Fri, Sep 25				No classes - student holiday	
	Fri, Sep 25		Project Intro: about the projects, self introductions, mentor intro, example projects, teamwork 32D-507			

- Module 1: Computational foundations
  - Dynamic programming: exploring exponential spaces in poly-time
  - Introduce Hidden Markov Models (HMMs): Central tool in CS
  - HMM algorithms: Decoding, evaluation, parsing, likelihood, scoring
- This week: Sequence alignment / comparative genomics
  - Local/global alignment: infer nucleotide-level evolutionary events
  - Database search: scan for regions that may have common ancestry
- Next week: Modeling genomes / exon / CpG island finding
  - Modeling class of elements, recognizing members of a class
  - Application to gene finding, conservation islands, CpG islands

Genome-wide alignments reveal orthologous segments

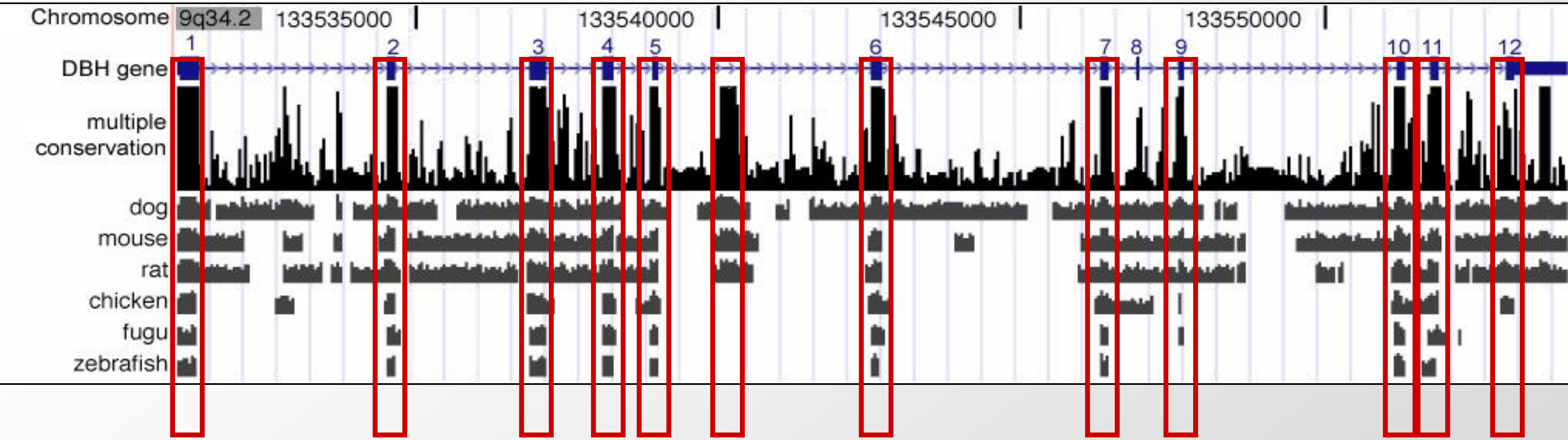


Courtesy of Don Gilbert. Used with permission.



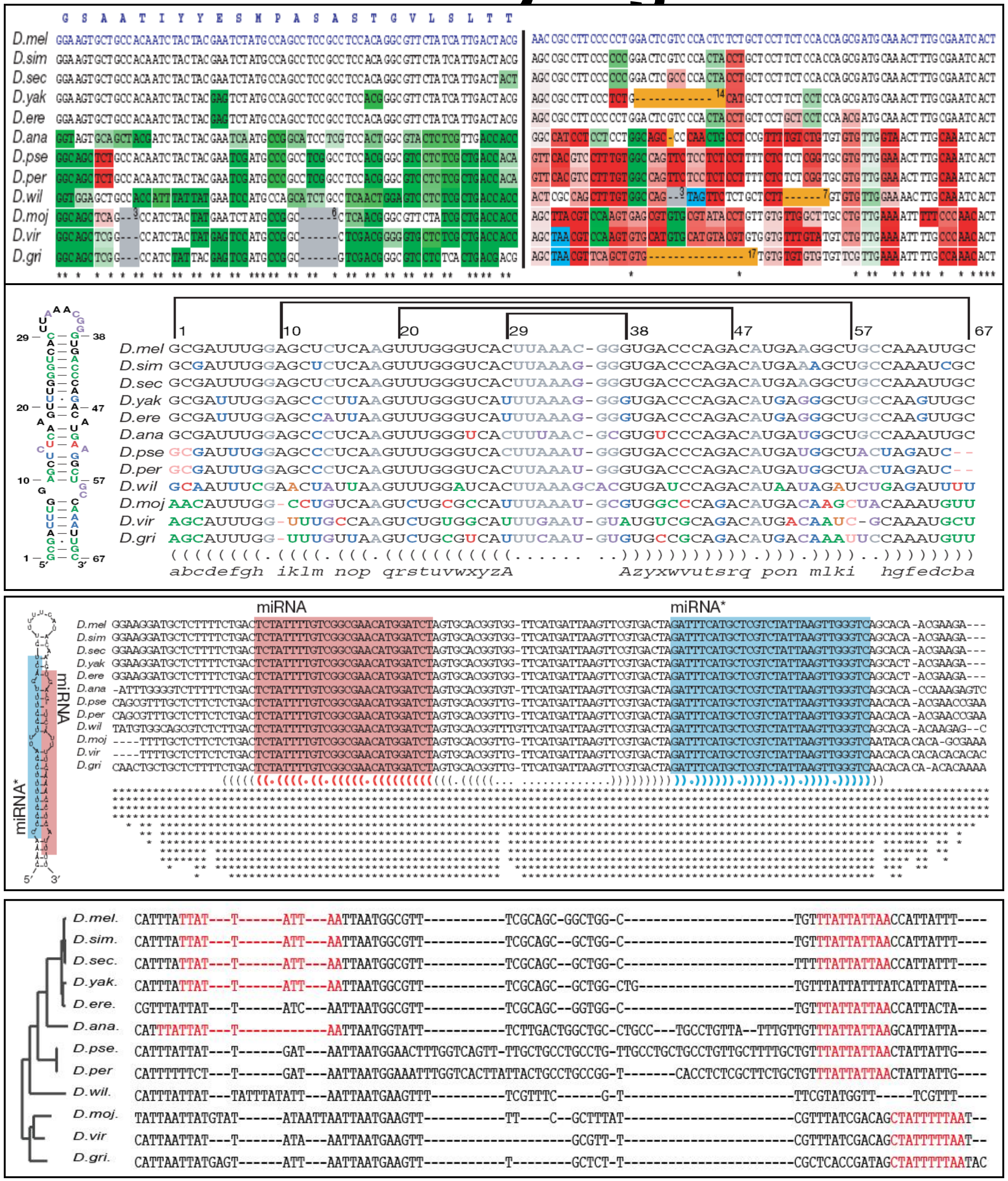
- Genome-wide alignments span entire genome
- Comparative identification of functional elements

Comparative genomics reveals conserved regions



- **Comparative genomics can reveal functional elements**
  - For example: exons are deeply conserved to mouse, chicken, fish
  - Many other elements are also strongly conserved: exons / regulatory?
- **Develop methods for estimating the level of constraint**
  - Count the number of edit operations, number of substitutions and gaps
  - Estimate the number of mutations (including estimate of back-mutations)
  - Incorporate information about neighborhood: conservation ‘windows’
  - Estimate the probability of a constrained ‘hidden state’: HMMs next week
  - Use phylogeny to estimate tree mutation rate, or ‘rejected substitutions’
  - Allow different portions of the tree to have different rates: phylogenetics

Evolutionary signatures for diverse functions



Courtesy of Macmillan Publishers Limited. Used with permission.
Source: Stark, Alexander et al. "Discovery of functional elements in 12 Drosophila genomes using evolutionary signatures." Nature 450, no. 7167 (2007): 219-232.

Protein-coding genes

- Codon Substitution Frequencies
- Reading Frame Conservation

RNA structures

- Compensatory changes
- Silent G-U substitutions

microRNAs

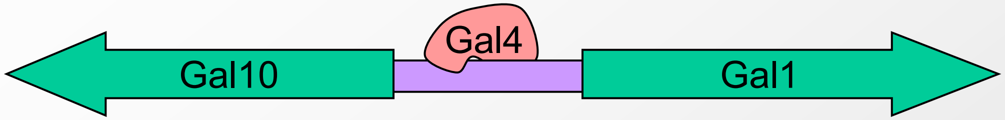
- Shape of conservation profile
- Structural features: loops, pairs
- Relationship with 3'UTR motifs

Regulatory motifs

- Mutations preserve consensus
- Increased Branch Length Score
- Genome-wide conservation

Stark et al, Nature 2007

Alignment: Evolution preserves functional elements!



```
Scer   TTATATTGAATTTTCAAAAATTCTTACTTTTTTTTTGGATGGACGCAAAGAAGTTTAATAATCATATTACATGGCATTACCACCATATACA
Spar   CTATGTTGATCTTTTCAGAATTTTT-CACTATATTAAGATGGGTGCAAAGAAGTGTGATTATTATATTACATCGCTTTCCTATCATACACA
Smik   GTATATTGAATTTTTCAGTTTTTTTTCACTATCTTCAAGGTTATGTAAAAAA-TGTCAAGATAATATTACATTTCGTTACTATCATACACA
Sbay   TTTTTTTGATTTCTTTAGTTTCTTTCTTTAACTTCAAAATTATAAAAGAAAGTGTAGTCACATCATGCTATCT-GTCACTATCACATATA
        * * ****  * *  *   ** ** *  *   **           **  ** * *    *    **   **    *  * * ** * * *

                          TBP
Scer   TATCCATATCTAATCTTACTTATATGTTGT-GGAAAT-GTAAAGAGCCCCATTATCTTAGCCTAAAAAAACC--TTCTCTTTGGAACTTTCAGTAATACG
Spar   TATCCATATCTAGTCTTACTTATATGTTGT-GAGAGT-GTTGATAACCCCAGTATCTTAACCCAAGAAAGCC--TT-TCTATGAAACTTGAACTG-TACG
Smik   TACCGATGTCTAGTCTTACTTATATGTTAC-GGGAATTGTTGGTAATCCCAGTCTCCCAGATCAAAAAAGGT--CTTTCTATGGAGCTTTG-CTA-TATG
Sbay   TAGATATTTCTGATCTTTCTTATATATTATAGAGAGATGCCAATAAACGTGCTACCTCGAACAAAAGAAGGGGATTTTCTGTAGGGCTTTCCCTATTTTG
       **   ** ***  **** *******  **   *  *   *     *  *    *  *       **  **      * *** *    ***    *  *  *

                                        GAL4               GAL4              GAL4
Scer   CTTAACTGCTCATTGC-----TATATTGAAGTACGGATTAGAAGCCGCCGAGCGGGCGACAGCCCTCCGACGGAAGACTCTCCTCCGTGCGTCCTCGTCT
Spar   CTAAACTGCTCATTGC-----AATATTGAAGTACGGATCAGAAGCCGCCGAGCGGACGACAGCCCTCCGACGGAATATTCCCCTCCGTGCGTCGCCGTCT
Smik   TTTAGCTGTTCAAG--------ATATTGAAATACGGATGAGAAGCCGCCGAACGGACGACAATTCCCCGACGGAACATTCTCCTCCGCGCGGCGTCCTCT
Sbay   TCTTATTGTCCATTACTTCGCAATGTTGAAATACGGATCAGAAGCTGCCGACCGGATGACAGTACTCCGGCGGAAAACTGTCCTCCGTGCGAAGTCGTCT
             **  **          ** ***** ******* ****** ***** ***  ****   * *** ***** * *  ****** ***    * ***

                                        GAL4
Scer   TCACCGG-TCGCGTTCCTGAAACGCAGATGTGCCTCGCGCCGCACTGCTCCGAACAATAAAGATTCTACAA-----TACTAGCTTTT--ATGGTTATGAA
Spar   TCGTCGGGTTGTGTCCCTTAA-CATCGATGTACCTCGCGCCGCCCTGCTCCGAACAATAAGGATTCTACAAGAAA-TACTTGTTTTTTTATGGTTATGAC
Smik   ACGTTGG-TCGCGTCCCTGAA-CATAGGTACGGCTCGCACCACCGTGGTCCGAACTATAATACTGGCATAAAGAGGTACTAATTTCT--ACGGTGATGCC
Sbay   GTG-CGGATCACGTCCCTGAT-TACTGAAGCGTCTCGCCCCGCCATACCCCGAACAATGCAAATGCAAGAACAAA-TGCCTGTAGTG--GCAGTTATGGT
            ** *   ** *** *      *      ***** ** *  *   ****** **     *   * **     * *             ** ***

                                    MIG1
Scer   GAGGA-AAAATTGGCAGTAA----CCTGGCCCCACAAACCTT-CAAATTAACGAATCAAATTAACAACCATA-GGATGATAATGCGA------TTAG--T
Spar   AGGAACAAAATAAGCAGCCC----ACTGACCCCATATACCTTTCAAACTATTGAATCAAATTGGCCAGCATA-TGGTAATAGTACAG------TTAG--G
Smik   CAACGCAAAATAAACAGTCC----CCCGGCCCCACATACCTT-CAAATCGATGCGTAAAACTGGCTAGCATA-GAATTTTGGTAGCAA-AATATTAG--G
Sbay   GAACGTGAAATGACAATTCCTTGCCCCT-CCCCAATATACTTTGTTCCGTGTACAGCACACTGGATAGAACAATGATGGGGTTGCGGTCAAGCCTACTCG
              ****    *         *   *****     ***              * * *    *  * *    *     *           **

                MIG1                                                     TBP
Scer   TTTTTAGCCTTATTTCTGGGGTAATTAATCAGCGAAGCG--ATGATTTTT-GATCTATTAACAGATATATAAATGGAAAAGCTGCATAACCAC-----TT
Spar   GTTTT--TCTTATTCCTGAGACAATTCATCCGCAAAAAATAATGGTTTTT-GGTCTATTAGCAAACATATAAATGCAAAAGTTGCATAGCCAC-----TT
Smik   TTCTCA--CCTTTCTCTGTGATAATTCATCACCGAAATG--ATGGTTTA--GGACTATTAGCAAACATATAAATGCAAAAGTCGCAGAGATCA-----AT
Sbay   TTTTCCGTTTTACTTCTGTAGTGGCTCAT--GCAGAAAGTAATGGTTTTCTGTTCCTTTTGCAAACATATAAATATGAAAGTAAGATCGCCTCAATTGTA
        * *      *    ***       * **   *  *     *** ***   *  *  **  ** * ********   ****    *

Scer   TAACTAATACTTTCAACATTTTCAGT--TTGTATTACTT-CTTATTCAAAT----GTCATAAAAGTATCAACA-AAAAATTGTTAATATACCTCTATACT
Spar   TAAATAC-ATTTGCTCCTCCAAGATT--TTTAATTTCGT-TTTGTTTATT----GTCATGGAAATATTAACA-ACAAGTAGTTAATATACATCTATACT
```

We can 'read' evolution to reveal functional elements

Conservation island

Yeast (Kellis et al, Nature 2003), Mammals (Xie, Nature 2005), Fly (Stark et al, Nature 07)

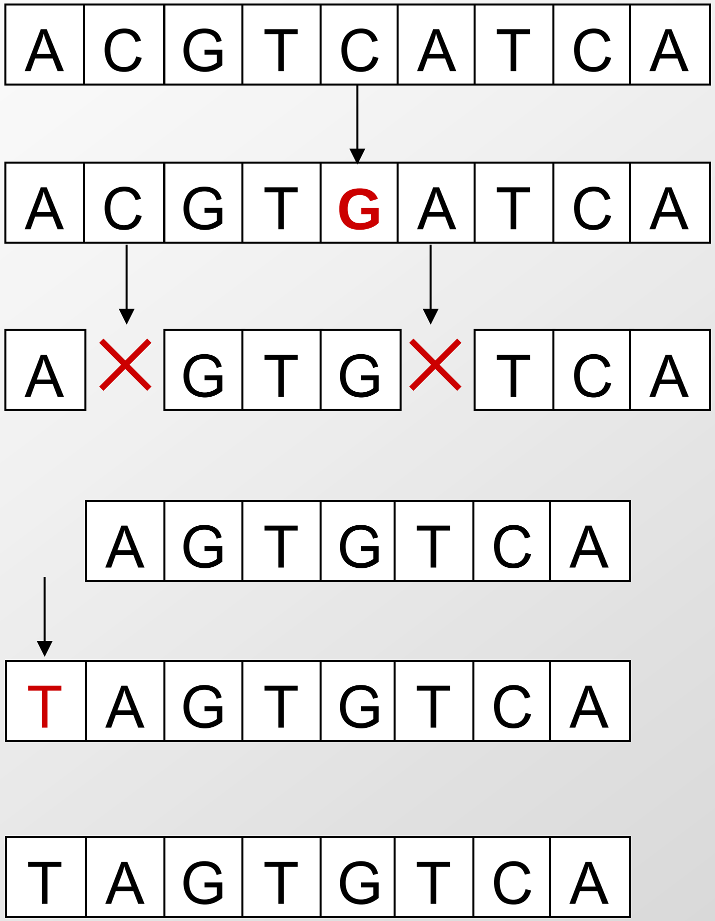
Today’s goal:

How do we actually align two genes?

Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it's hard: Exponential number of alignments
2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back
3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry⇔prefix alignment score; path⇔aligmnt
4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP(heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition

Genomes change over time



begin

A C G T C A T C A

mutation

A C G T G A T C A

deletion

A ✕ G T G ✕ T C A

A G T G T C A

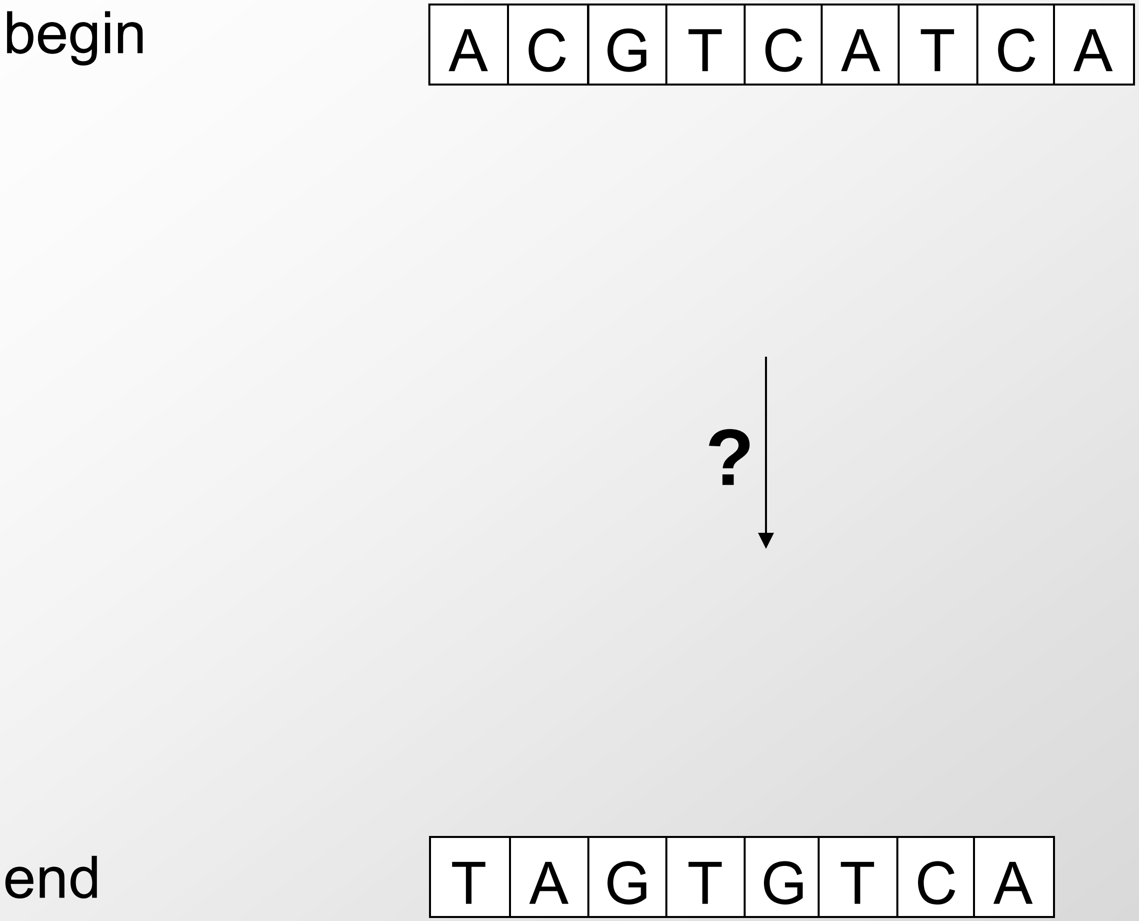
insertion

T A G T G T C A

end

T A G T G T C A

Goal of alignment: Infer edit operations



begin

A	C	G	T	C	A	T	C	A

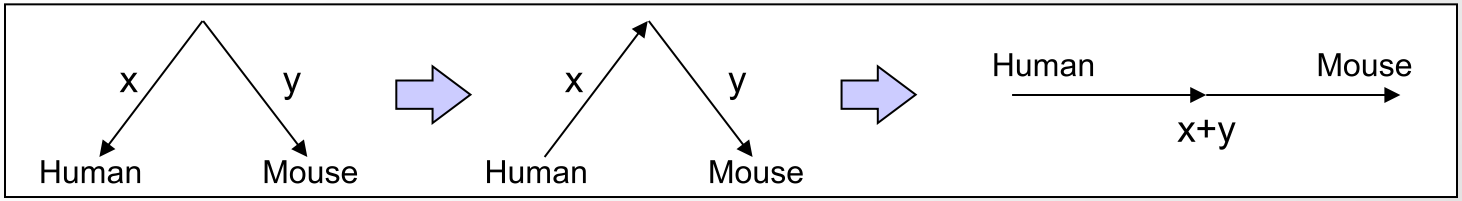
?

end

T	A	G	T	G	T	C	A

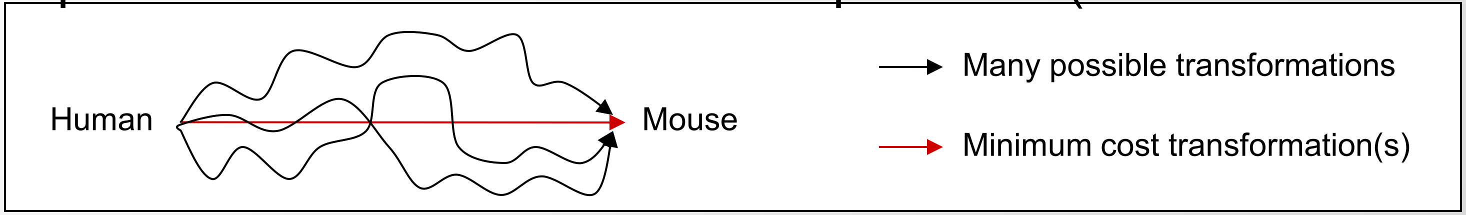
From Bio to CS: Formalizing the problem

- Define set of evolutionary operations (insertion, deletion, mutation)
  - Symmetric operations allow time reversibility (part of design choice)

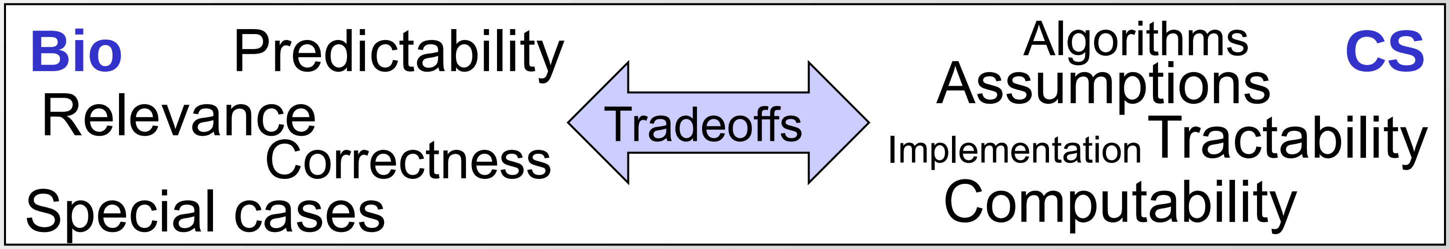


(Exception: methylated CpG dinucleotides → TpG/CpA non-symmetric)

- Define optimality criterion (min number, min cost)
  - Impossible to infer exact series of operations (Occam's razor: find min)



- Design algorithm that achieves that optimality (or approximates it)
  - Tractability of solution depends on assumptions in the formulation



Bio Predictability

Relevance

Correctness

Special cases

Tradeoffs

Algorithms CS

Assumptions

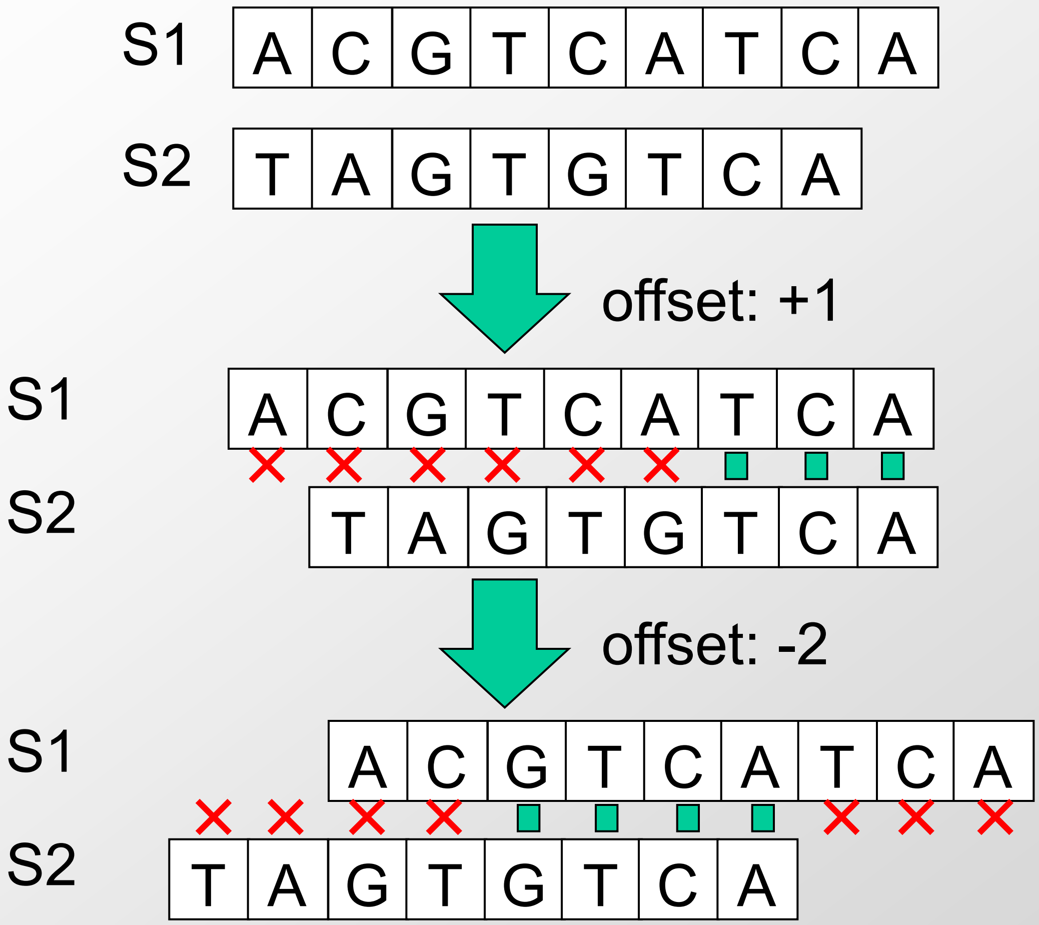
Implementation Tractability

Computability

Note: Not all decisions are conflicting (some are both relevant and tractable)
(e.g. Pevzner vs. Sankoff and directionality in chromosomal inversions)

Formulation 1: Longest common substring

- Given two possibly related strings S1 and S2
  - What is the longest common substring? (no gaps)



S1 A C G T C A T C A

S2 T A G T G T C A

offset: +1

S1 A C G T C A T C A

✗ ✗ ✗ ✗ ✗ ✗ 🟩 🟩 🟩

S2 T A G T G T C A

offset: -2

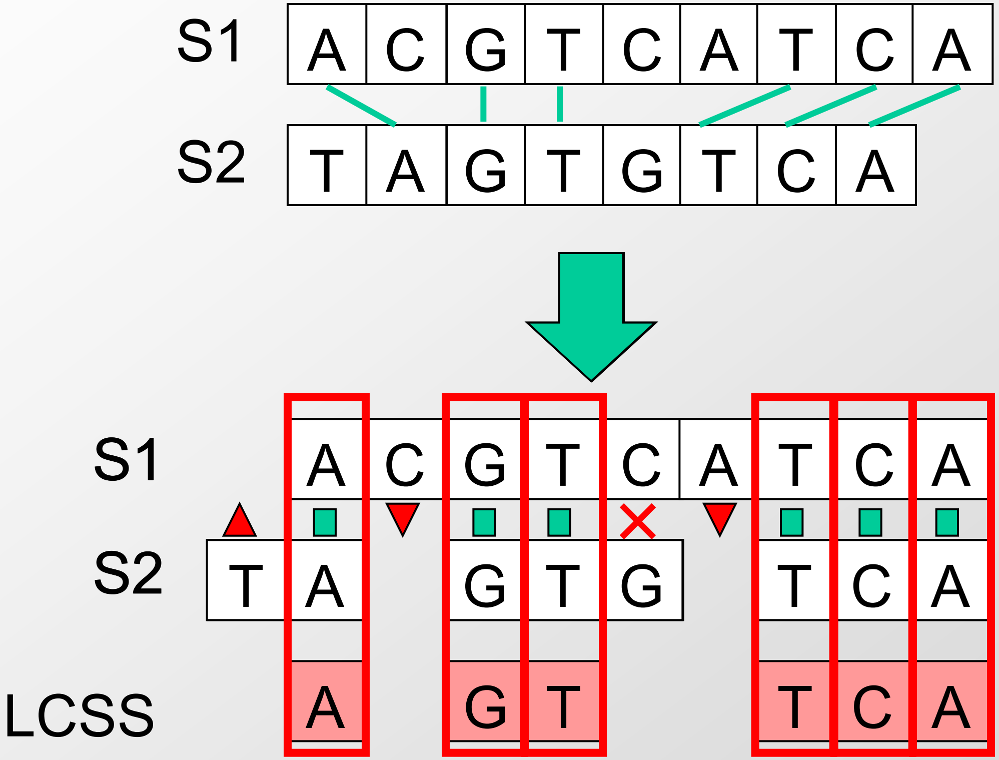
S1 A C G T C A T C A

✗ ✗ ✗ ✗ 🟩 🟩 🟩 🟩 ✗ ✗ ✗

S2 T A G T G T C A

Formulation 2: Longest common subsequence

- Given two possibly related strings S1 and S2
  - What is the longest common subsequence? (gaps allowed)



Related to:
Edit distance:

- Number of changes needed for S1→S2
- Uniform scoring function

Formulation 3: Sequence alignment

- Allow gaps (fixed penalty)
  - Insertion & deletion operations
  - Unit cost for each character inserted or deleted
- Varying penalties for edit operations
  - Transitions (Pyrimidine⇔Pyrimidine, Purine⇔Purine)
  - Transversions (Purine ⇔ Pyrimidine changes)
  - Polymerase confuses Aw/G and Cw/T more often

Scoring function:

Match(x,x) = +1

Mismatch(A,G)= -½

Mismatch(C,T)= -½

Mismatch(x,y) = -1

	A	G	T	C
A	+1	-½	-1	-1
G	-½	+1	-1	-1
T	-1	-1	+1	-½
C	-1	-1	-½	+1

purine pyrimid.

Transitions:

A⇔G, C⇔T common (lower penalty)

Transversions:

All other operations

Formulation 4: Varying gap cost models

1. Linear gap penalty
   - Same as before
2. Affine gap penalty
   - Big initial cost for starting or ending a gap
   - Small incremental cost for each additional character
3. General gap penalty
   - Any cost function
   - No longer computable using the same model
4. Frame-aware gap penalty
   - Multiples of 3 disrupt coding regions
5. Seek duplicated regions, rearrangements, …
   - Etc

How many alignments are there?

S1	A	C	▨	G	▨	T	C	▨	▨	A	▨	T	▨	▨	▨	C	A
S2	▨	▨	T	▨	A	▨	▨	G	T	▨	G	▨	T	C	A	▨	▨

- Longest ‘non-boring’ alignment: n+m entries
  - Otherwise a gap will be aligned to a gap→condense
- Alignment is equivalent to gap placement
  - (n+m choose n) ways to choose S1 placement
  - At each position yes/no answer of placing character
  - Exponential number of possible placements
- Exponential number of sequence alignment
  - Enumerating and scoring each of them not an option
  - Need faster solution for finding best alignment

Need **polynomial** algorithm to find best alignment
amongst an **exponential** number of possible alignments!

→ DP

Goal: Sequence Alignment / Dynamic Programming

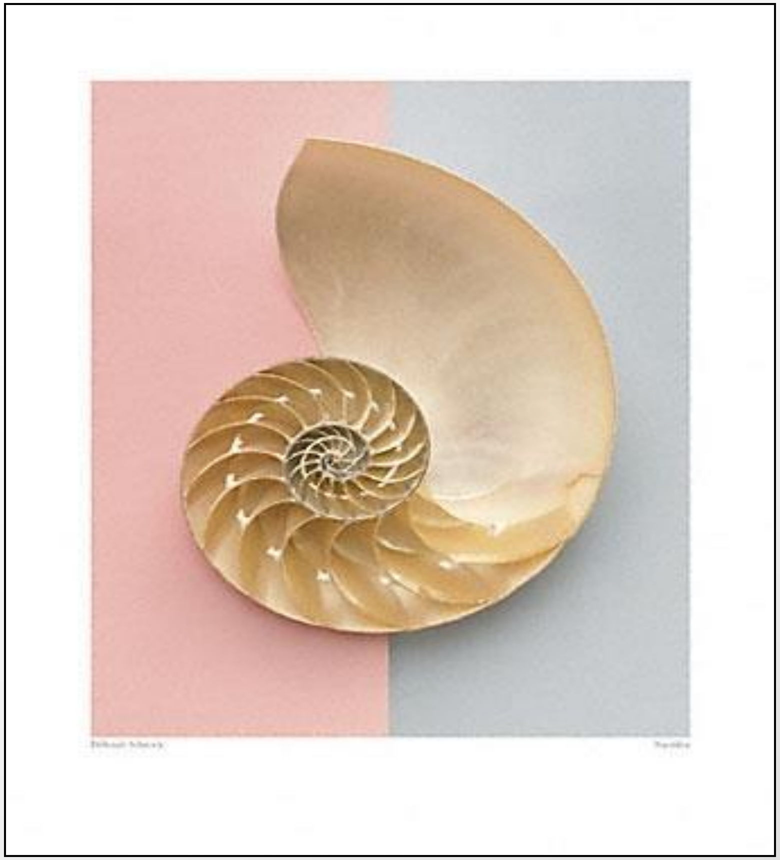
1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it's hard: Exponential number of alignments
2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back
3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry⇔prefix alignment score; path⇔aligmnt
4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP(heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition

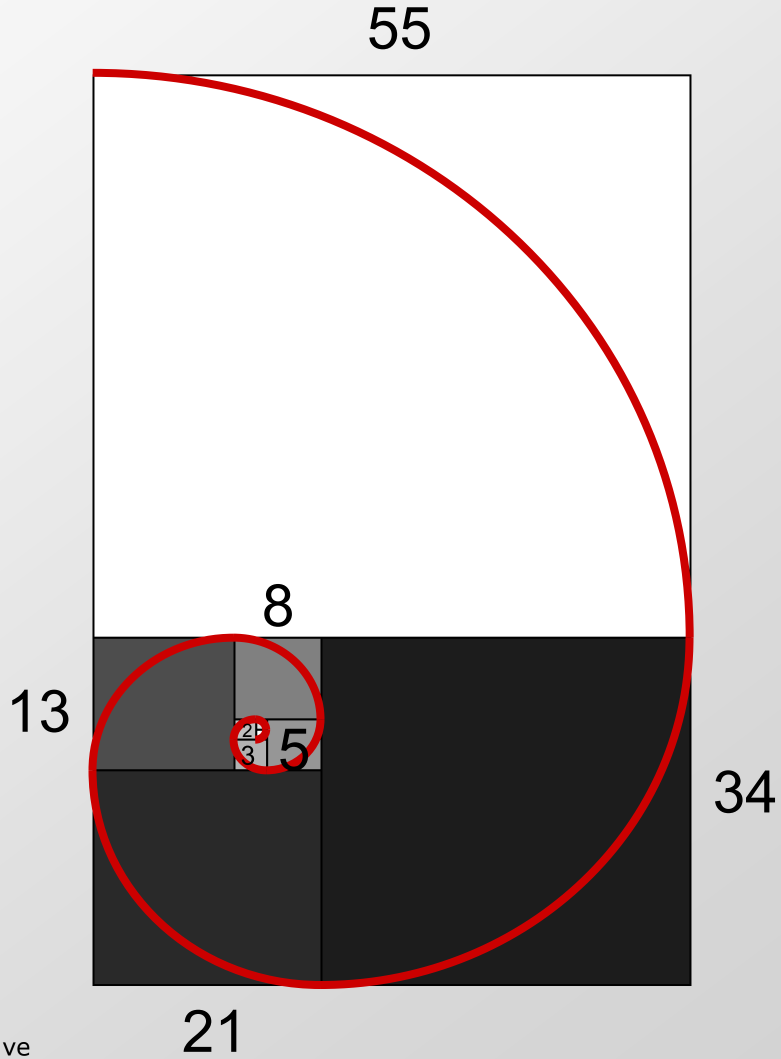
A simple introduction to the principles of Dynamic Programming

Turning exponentials into polynomials

Computing Fibonacci Numbers

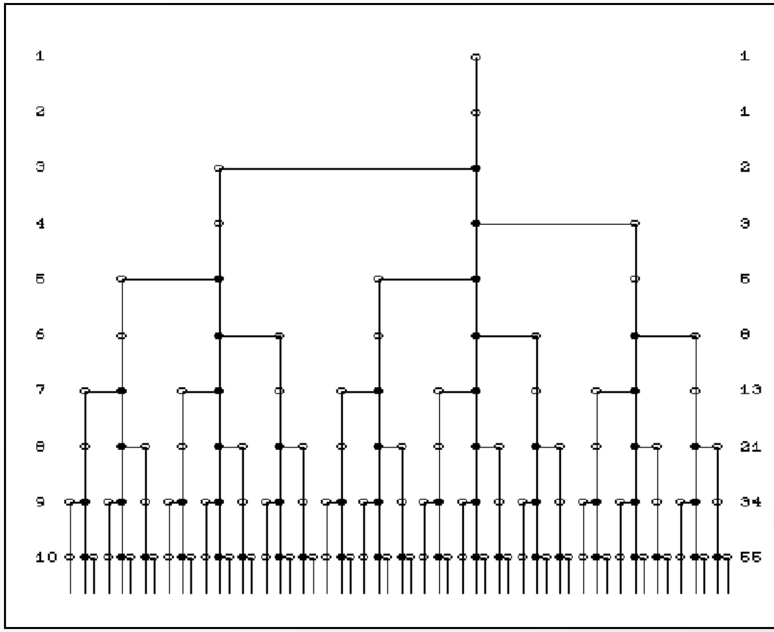
- Fibonacci numbers



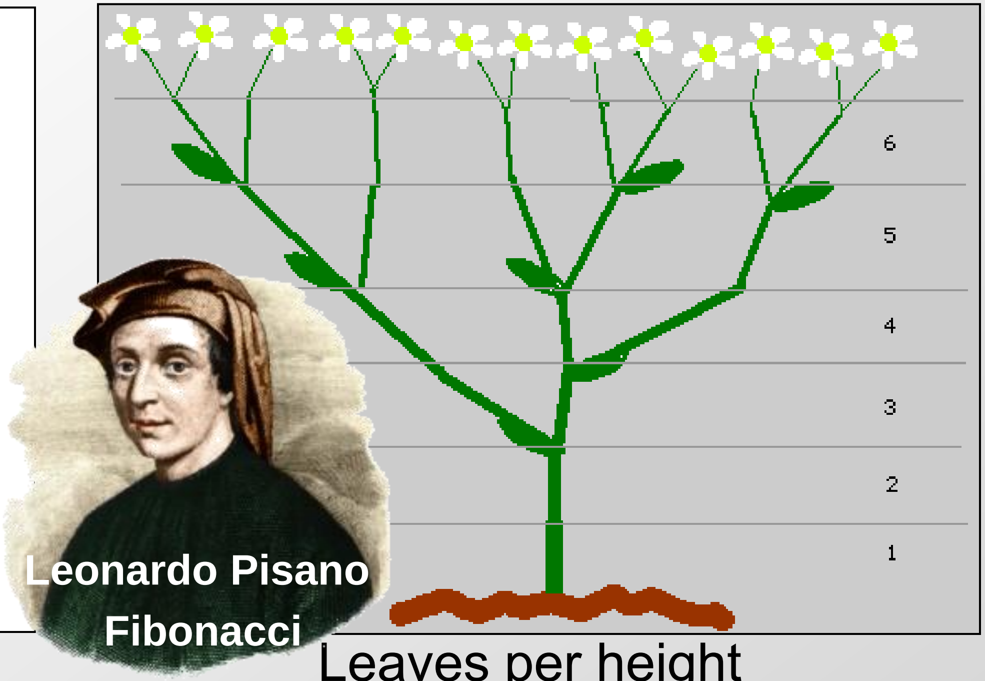


F6=F5+F4=(F4+F3)+(F3+F2))=….=(3+2)+(2+1)=5+3=8

Fibonacci numbers are ubiquitous in nature



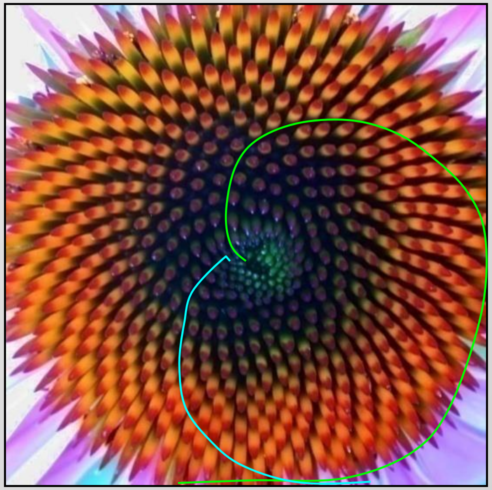
Rabbits per generation



Leaves per height

Romanesque spirals

Nautilus size



Coneflower spirals

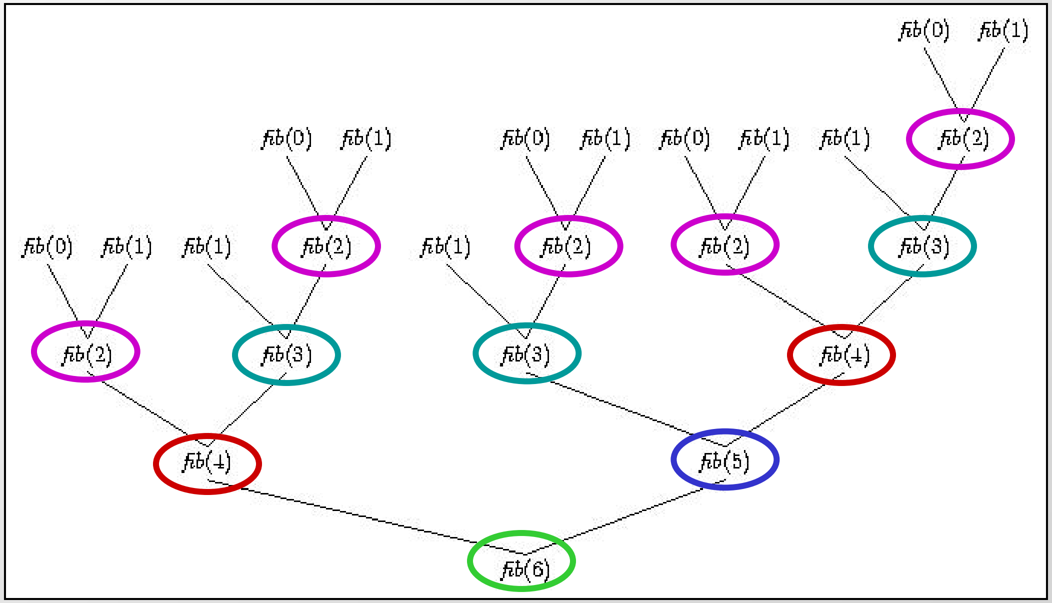
Leaf ordering

Computing Fibonacci numbers: Top down

- Fibonacci numbers are defined recursively:
  - Python code

```
def fibonacci(n):
    if n==1 or n==2: return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

- Goal: Compute n^th Fibonacci number.
  - F(0)=1, F(1)=1, F(n)=F(n-1)+F(n-2)
  - 1,1,2,3,5,8,13,21,34,55,89,144,233,377,…
- Analysis:
  - T(n) = T(n-1) + T(n-2) = (…) = $O(2^n)$



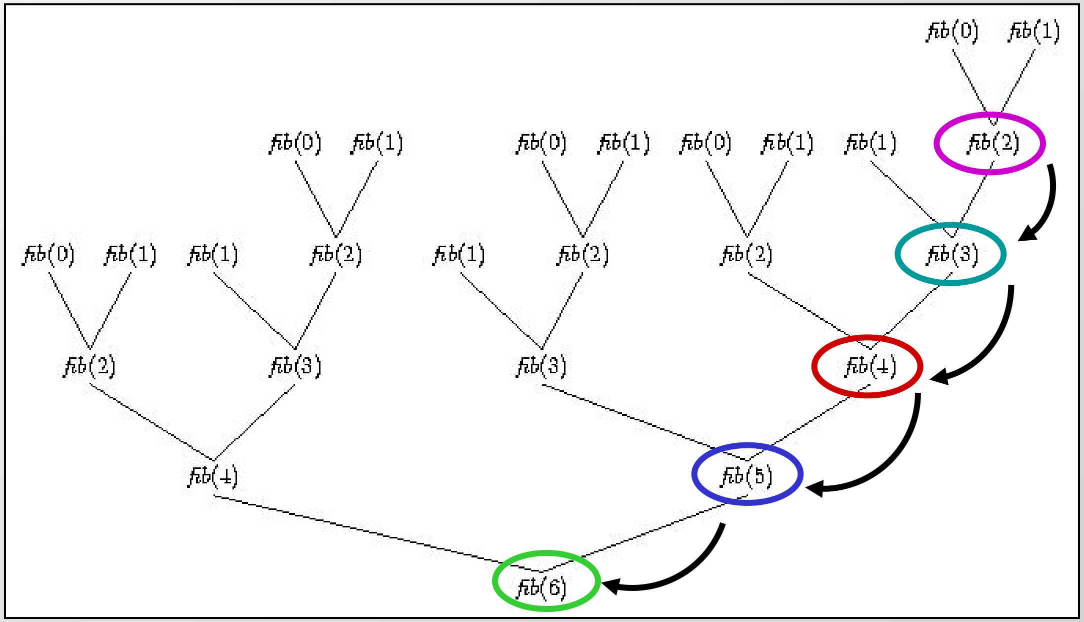
Computing Fibonacci numbers: Bottom up

- Bottom up approach
  - Python code

fib_table	
F[1]	1
F[2]	1
F[3]	2
F[4]	3
F[5]	5
F[6]	8
F[7]	13
F[8]	21
F[9]	34
F[10]	55
F[11]	89
F[12]	**?**

```
def fibonacci(n):
        fib_table[1] = 1
        fib_table[2] = 1
        for i in range(3,n+1):
          fib_table[i] = fib_table[i-1]+fib_table[i-2]
        return fib_table[n]
```

  - Analysis: T(n) = O(n)



Lessons from iterative Fibonacci algorithm

fib_table	
F[1]	1
F[2]	1
F[3]	2
F[4]	3
F[5]	5
F[6]	8
F[7]	13
F[8]	21
F[9]	34
F[10]	55
F[11]	89
F[12]	**?**

- What did the iterative solution do?
  - Reveal identical sub-problems
  - Order computation to enable result reuse
  - Systematically filled-in table of results
  - Expressed larger problems from their subparts
- Ordering of computations matters
  - Naïve top-down approach very slow
    - results of smaller problems not available
    - repeated work
  - Systematic bottom-up approach successful
    - Systematically solve each sub-problem
    - Fill-in table of sub-problem results in order.
    - Look up solutions instead of recomputing

Dynamic Programming in Theory

- Hallmarks of Dynamic Programming
  - **Optimal substructure:** Optimal solution to problem (instance) contains optimal solutions to sub-problems
  - **Overlapping subproblems:** Limited number of distinct subproblems, repeated many many times
- Typically for optimization problems (unlike Fib example)
  - Optimal choice made locally: max(subsolution score)
  - Score is typically added through the search space
  - Traceback common, find optimal path from indiv. choices
- Middle of the road in range of difficulty
  - Easier: greedy choice possible at each step
  - DynProg: requires a traceback to find that optimal path
  - Harder: no opt. substr., e.g. subproblem dependencies

Hallmarks of optimization problems

Greedy algorithms | **Dynamic Programming**

1. *Optimal substructure*

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

2. *Overlapping subproblems*

A recursive solution contains a "small" number of distinct subproblems repeated many times.

3. *Greedy choice property*

Locally optimal choices lead to globally optimal solution

Greedy Choice is not possible

Globally optimal solution requires trace back through many choices

Dynamic Programming in Practice

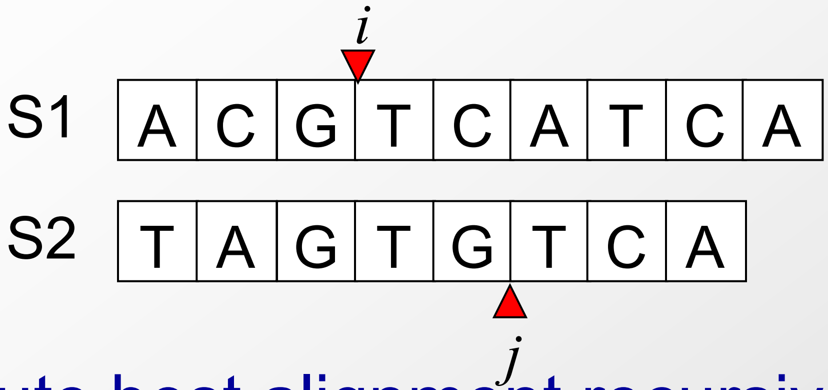
- Setting up dynamic programming
  1. Find ‘matrix’ parameterization (# dimensions, variables)
  2. Make sure sub-problem space is finite! (not exponential)
     - If not all subproblems are used, better off using memoization
     - If reuse not extensive, perhaps DynProg is not right solution!
  3. Traversal order: sub-results ready when you need them
     - Computation order matters! (bottom-up, but not always obvious)
  4. **Recursion formula: larger problems = F(subparts)**
  5. Remember choices: typically F() includes min() or max()
     - Need representation for storing pointers, is this polynomial !
- Then start computing
  1. Systematically fill in table of results, find optimal score
  2. Trace-back from optimal score, find optimal solution

Goal: Sequence Alignment / Dynamic Programming

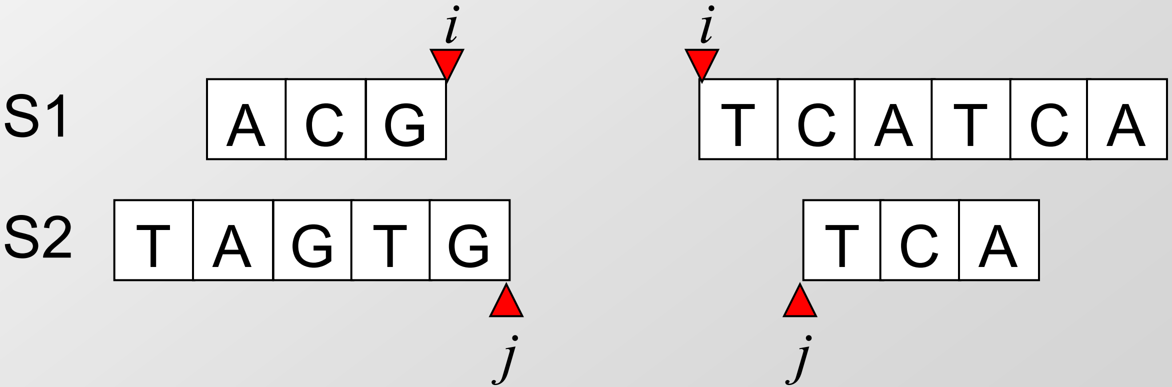
1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it's hard: Exponential number of alignments
2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back
3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry⇔prefix alignment score; path⇔alignmnt
4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP(heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition

(3) How do we apply dynamic programming to sequence alignment ?

Key insight #1: Score is additive, smaller to larger

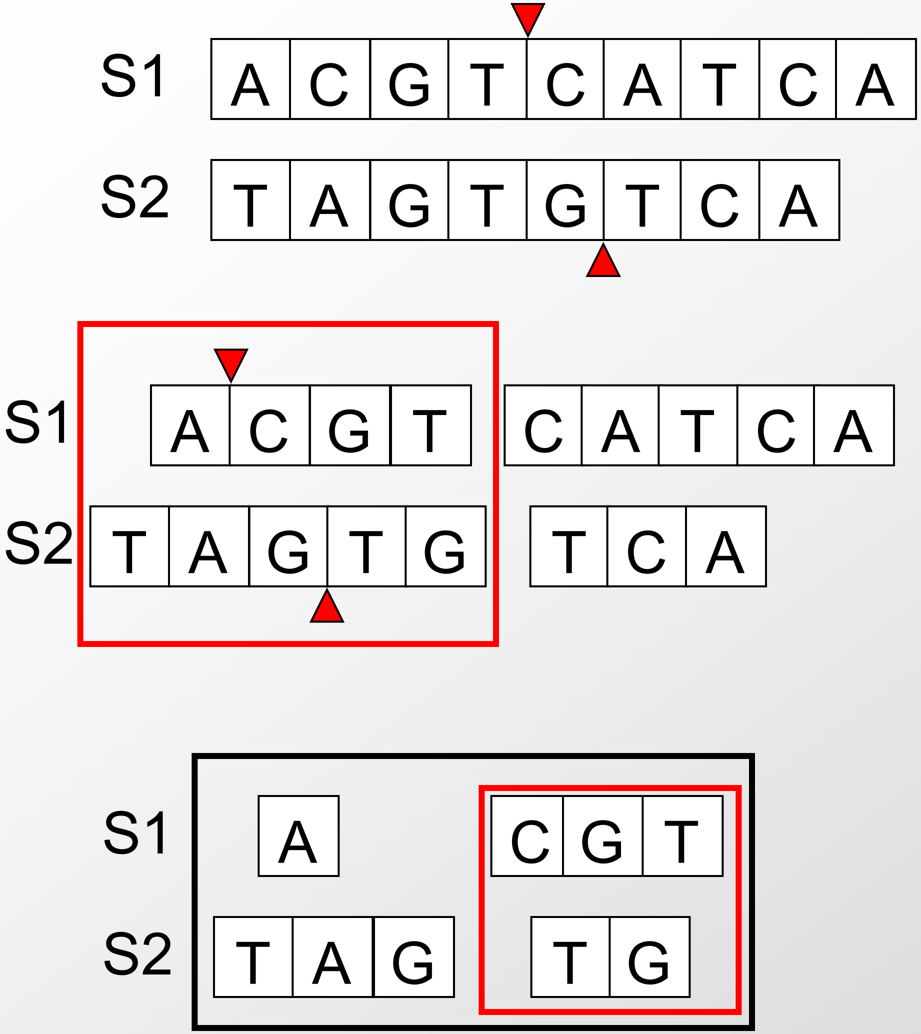


- Compute best alignment recursively
  - For a given aligned pair ***(i, j)***, the best alignment is:
    - Best alignment of S1[1..i]   and S2[1..j]
    - + Best alignment of S1[i..n] and S2[j..m]
  - Proof: cut-and-paste argument (see 6.046)



This allows a single recursion (top-left to bottom-right) instead of two recursions (middle-to-outside top-down)

Key insight #2: compute scores recursively



S1 A C G T C A T C A

S2 T A G T G T C A

S1 A C G T C A T C A

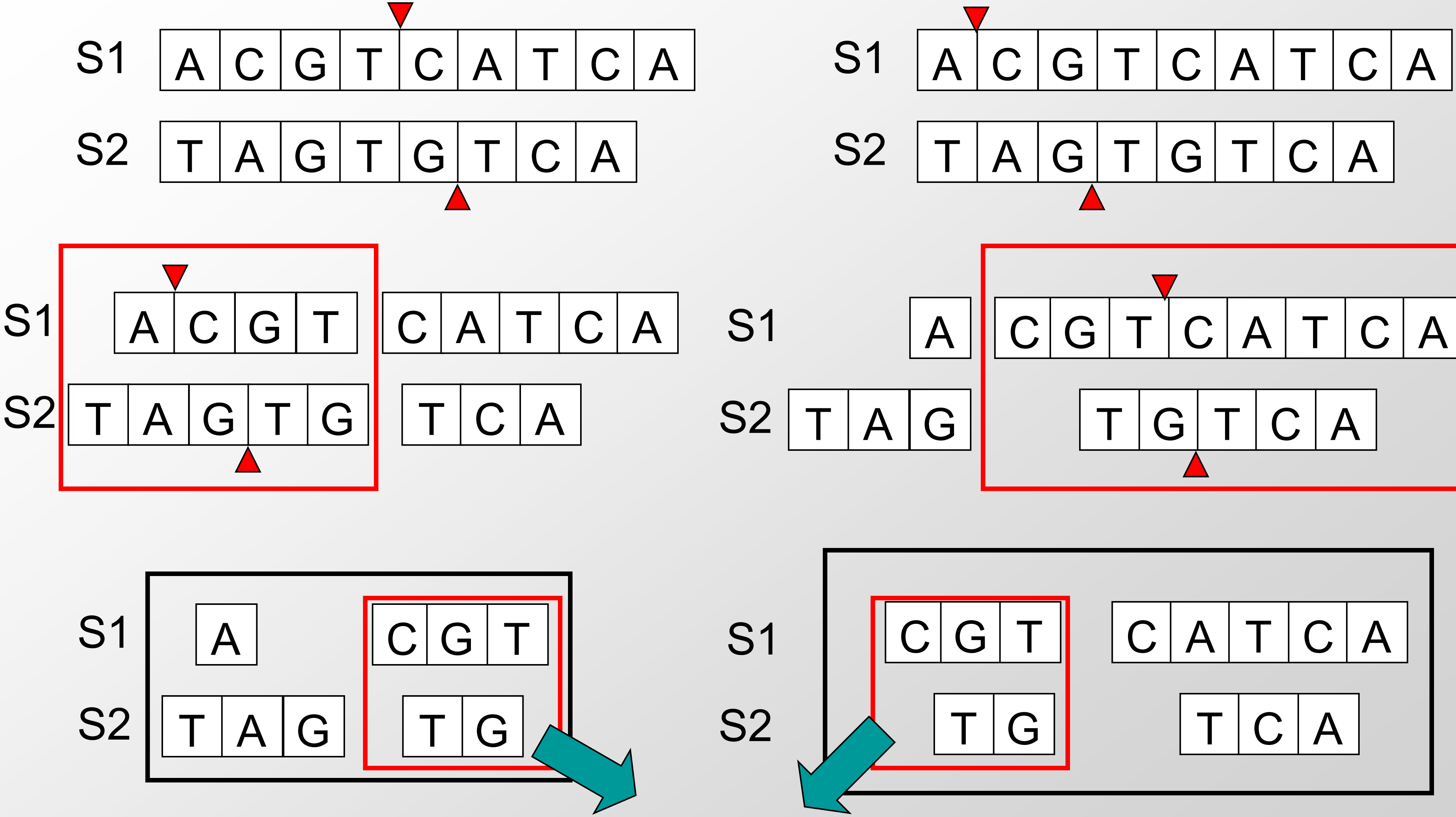
S2 T A G T G T C A

S1 A C G T

S2 T A G T G

➔ Compute alignment of CGT vs. TG exactly once

Key insight #3: sub-problems are repeated → reuse!



S1 A C G T C A T C A

S2 T A G T G T C A

S1 A C G T C A T C A

S2 T A G T G T C A

S1 A C G T C A T C A

S2 T A G T G T C A

S1 A C G T C A T C A

S2 T A G T G T C A

S1 A C G T

S2 T A G T G

S1 C A T C A

S2 T C A

S1 A C G T C A T C A

S2 T A G T G T C A

S1 A

S2 T A G

S1 C G T

S2 T G

S1 C G T

S2 T G

S1 C A T C A

S2 T C A

➔ Identical sub-problems! We can reuse our work!

Solution #1 – Memoization

- Create a big dictionary, indexed by aligned seqs
  - When you encounter a new pair of sequences
  - If it is in the dictionary:
    - Look up the solution
  - If it is not in the dictionary
    - Compute the solution
    - Insert the solution in the dictionary
- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!

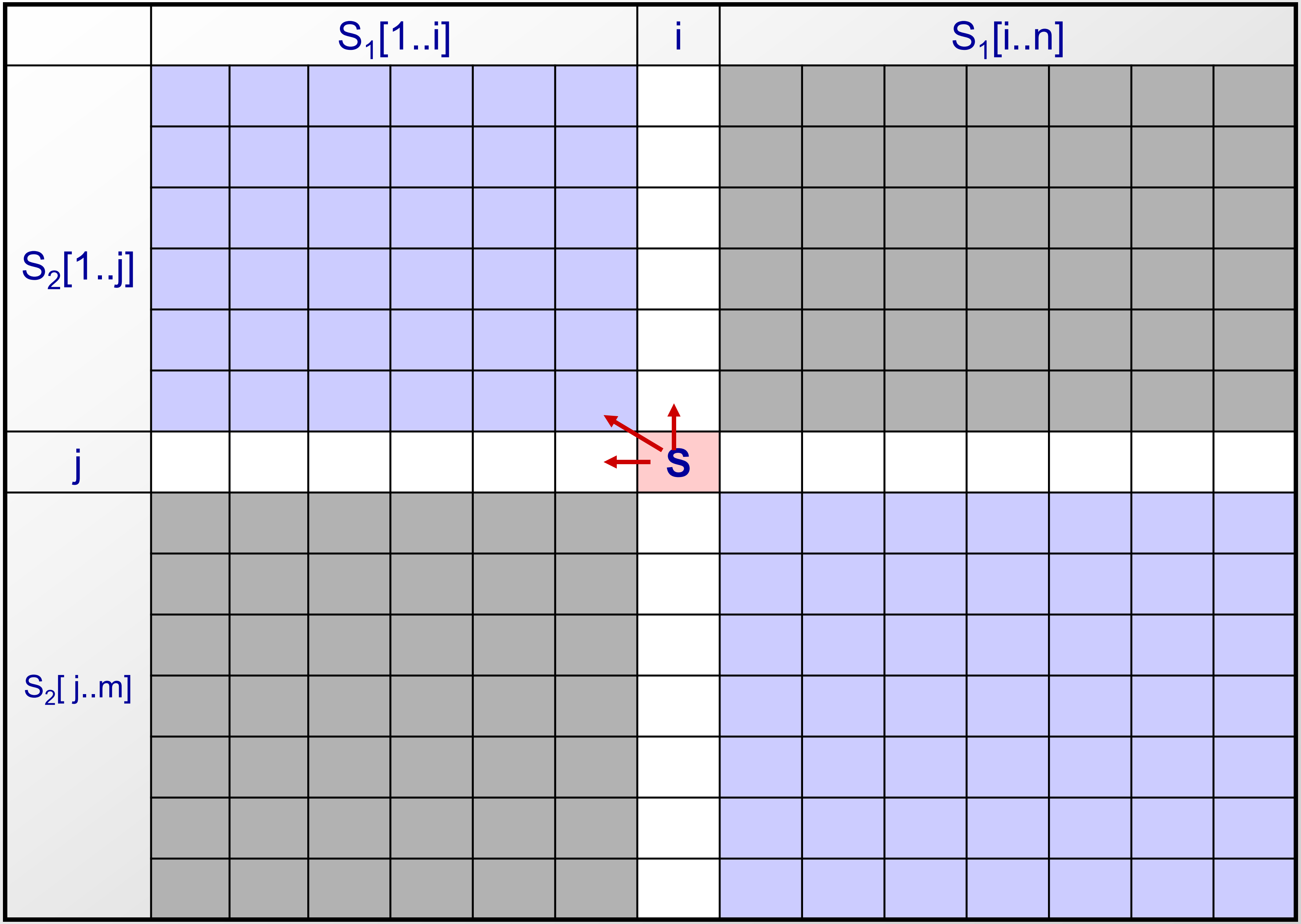
Top down approach

Solution #2 – Dynamic programming

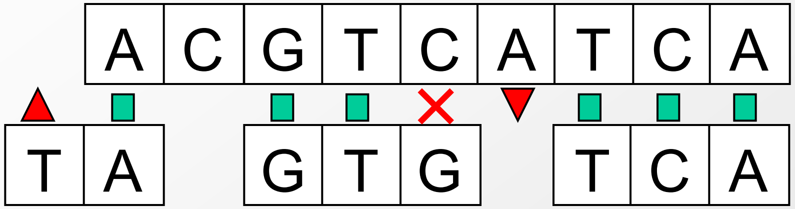
- Create a big table, indexed by (i,j)
  - Fill it in from the beginning all the way till the end
  - You know that you'll need every subpart
  - Guaranteed to explore entire search space
- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!
- Very simple computationally!

Bottom up approach

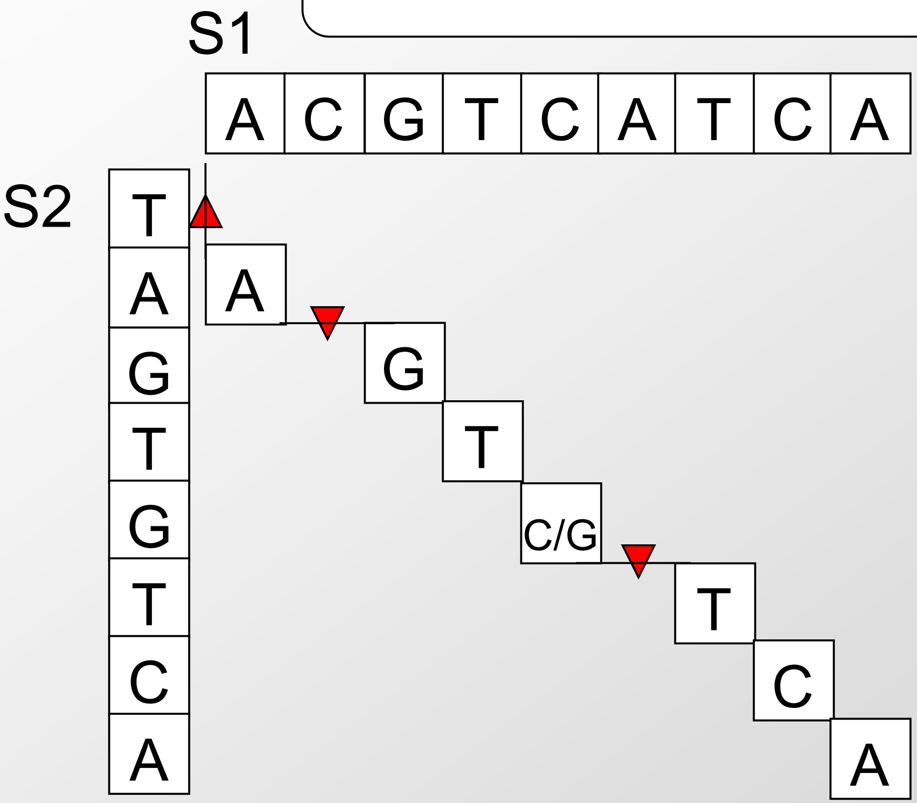
Key insight #4: Optimal prefix almt score ⇔ Matrix entry



Key insight #5: Optimal alignment ⇔ Matrix path



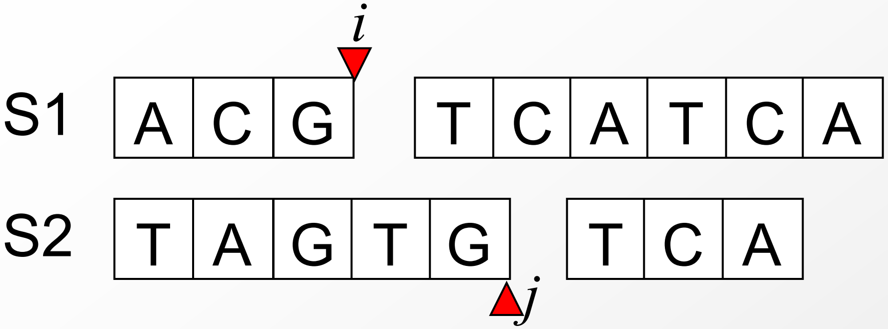
Best alignment ⇔ Best path through the matrix



Goal:

Find best path through the matrix

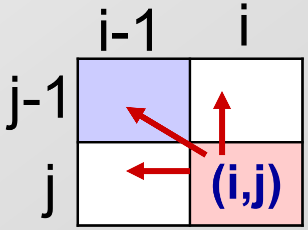
DP approach: iteratively grow best alignment soltn



- Compute all alignment scores from the bottom up
  - Define M[i,j] prefix alignment score of S_1[1..i] and S_2[1..j]
  - Fill up table recursively from smaller to bigger alignments
- Express alignment of S_1[1..i+1] and S_2[1..j+1] ➔ M[i+1,j+1]
  - One of three possibilities: (1) extend alignment from M[i,j] (2) extend from M[i-1,j], or (3) extend from M[i,j-1]
  - Only a local computation, takes O(1) time!
- Proof of correctness (cut-and-paste argument from 6.006)
  - Best alignment of S_1[1..i+1] and S_2[1..j+1] must be composed of best alignments of smaller prefix
  - Proof: otherwise could replace sub and get better overall

Computing alignments recursively: M[i,j]=F(smaller)

- Local update rules, only look at neighboring cells:
  - Compute next alignment based on previous alignment
  - Just like Fibonacci numbers: F[i] = F[i-1] + F[i-2]
  - Table lookup avoids repeated computation
- Computing the score of a cell from smaller neighbors
  - $$M(i,j) = \max\left\{ \begin{array}{l} M(i-1, j) - \text{gap} \\ M(i-1, j-1) + \text{score} \\ M(i, j-1) - \text{gap} \end{array} \right\}$$

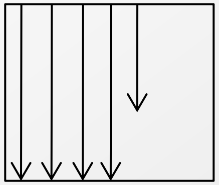


  - Only three possibilities for extending by one nucleotide: a gap in one species, a gap in the other, a (mis)match
- Compute scores for prefixes of increasing length
  - Start with prefixes of length 1, extend by one each time, until all prefixes have been computed
  - When you reach bottom right, alignment score of $S_1[1..m]$ and $S_2[1..n]$ is alignment of full S_1 and full S_2
  - (Can then trace back to construct optimal path to it)

Dynamic Programming for sequence alignment

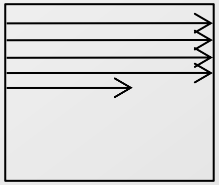
- Setting up dynamic programming

1. Find ‘matrix’ parameterization
   - Prefix parameterization. Score(S_1[1..i],S_2[1..j]) ➔ M(i,j)
   - (i,j) only prefixes vs. (i,j,k,l) all substrings ➔ simpler 2-d matrix
2. Make sure sub-problem space is finite! (not exponential)
   - It’s just n^2, quadratic (which is polynomial, not exponential)
3. Traversal order: sub-results ready when you need them



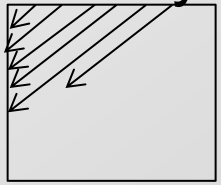
Cols

L→R



Rows

top→bot



Diags

topR→botL

4. Recursion formula: larger problems = Func(subparts)
   - Need formula for computing M[i,j] as function of previous results
   - Single increment at a time, only look at M[i-1,j], M[i,j-1], M[i-1,j-1] corresponding to 3 options: gap in S_1, gap in S_2, char in both
   - Score in each case depends on gap/match/mismatch penalties
5. Remember choice: F() typically includes min() or max()
   - Remember which of three cells (top,left,diag) led to maximum

Step 1: Setting up the scoring matrix M[i,j]

	-	A	G	T
-	0			
A				
A				
G				
C				

Initialization:

- Top left: 0

Update Rule:

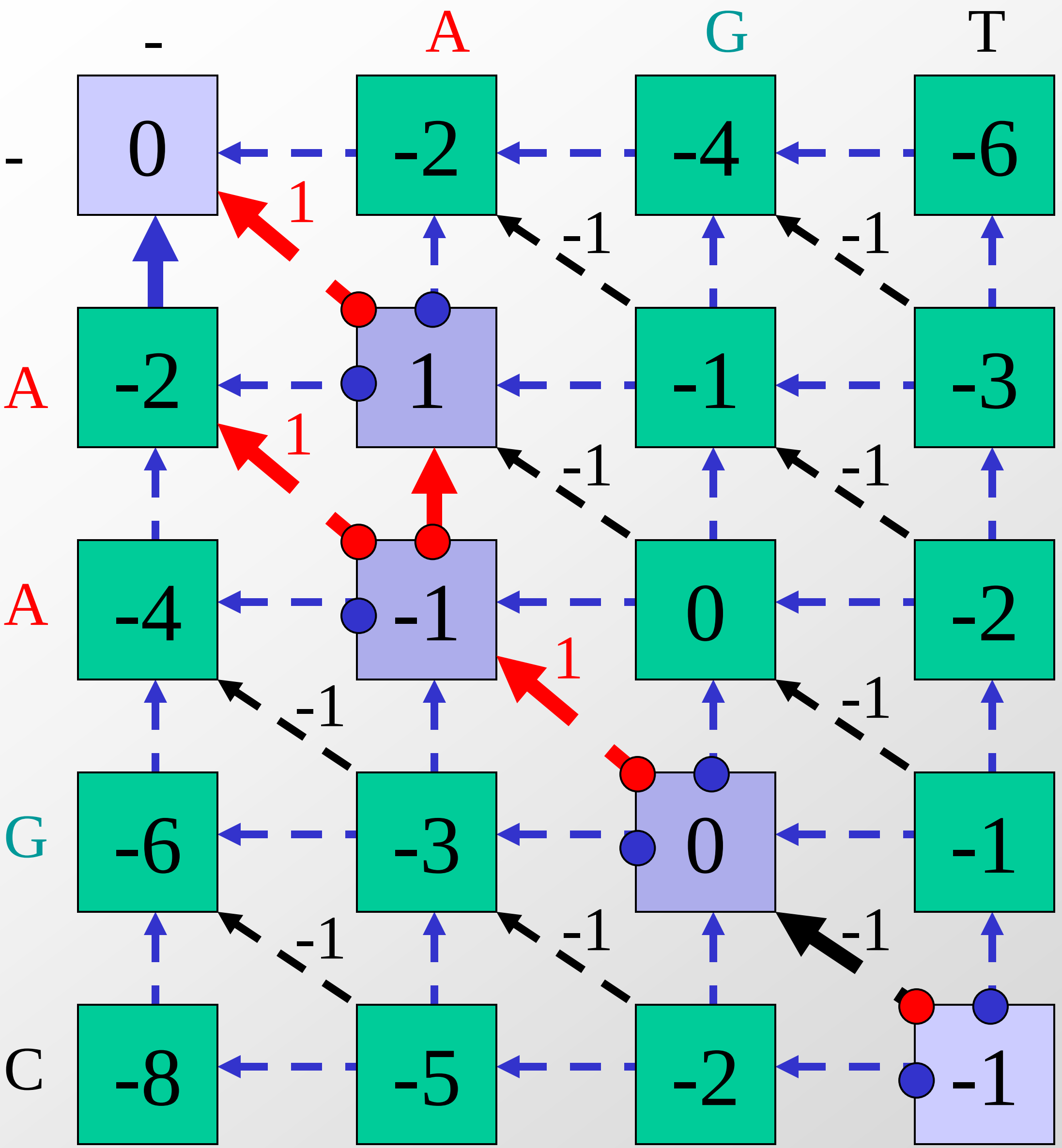
M(*i*,*j*)=max{

}

Termination:

- Bottom right

Step 2: Filling in the optimal scores from top left



	-	A	G	T
-	0	-2	-4	-6
A	-2	1	-1	-3
A	-4	-1	0	-2
G	-6	-3	0	-1
C	-8	-5	-2	-1

Initialization:

- Top left: 0

Update Rule:

M(*i*,*j*)=max{

- M(*i*-1 , *j*) - 2 gap
- M(*i* , *j*-1) - 2 gap
- M(*i*-1 , *j*-1) -1 mismatch
- M(*i*-1 , *j*-1)+1 match

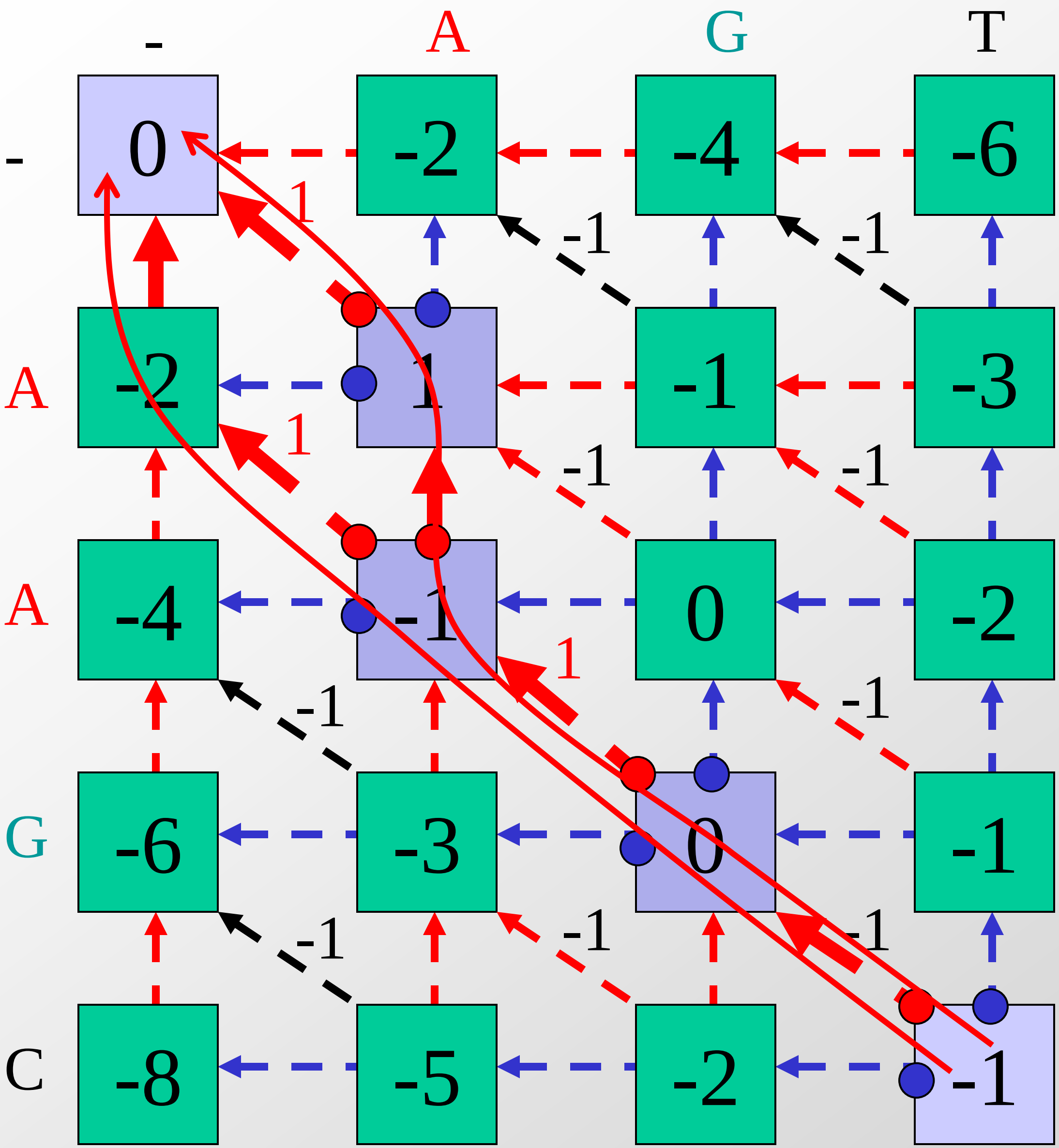
}

Termination:

- Bottom right

● Path segment that lead to the optimal choice

Step 3: Trace back pointers to construct alignment



	-	A	G	T
-	0	-2	-4	-6
A	-2	1	-1	-3
A	-4	-1	0	-2
G	-6	-3	0	-1
C	-8	-5	-2	-1

Initialization:

- Top left: 0

Update Rule:

M(i,j)=max{

- M(i-1 , j) - 2 gap
- M(i , j-1) - 2 gap
- M(i-1 , j-1) -1 mismatch
- M(i-1 , j-1)+1 match

}

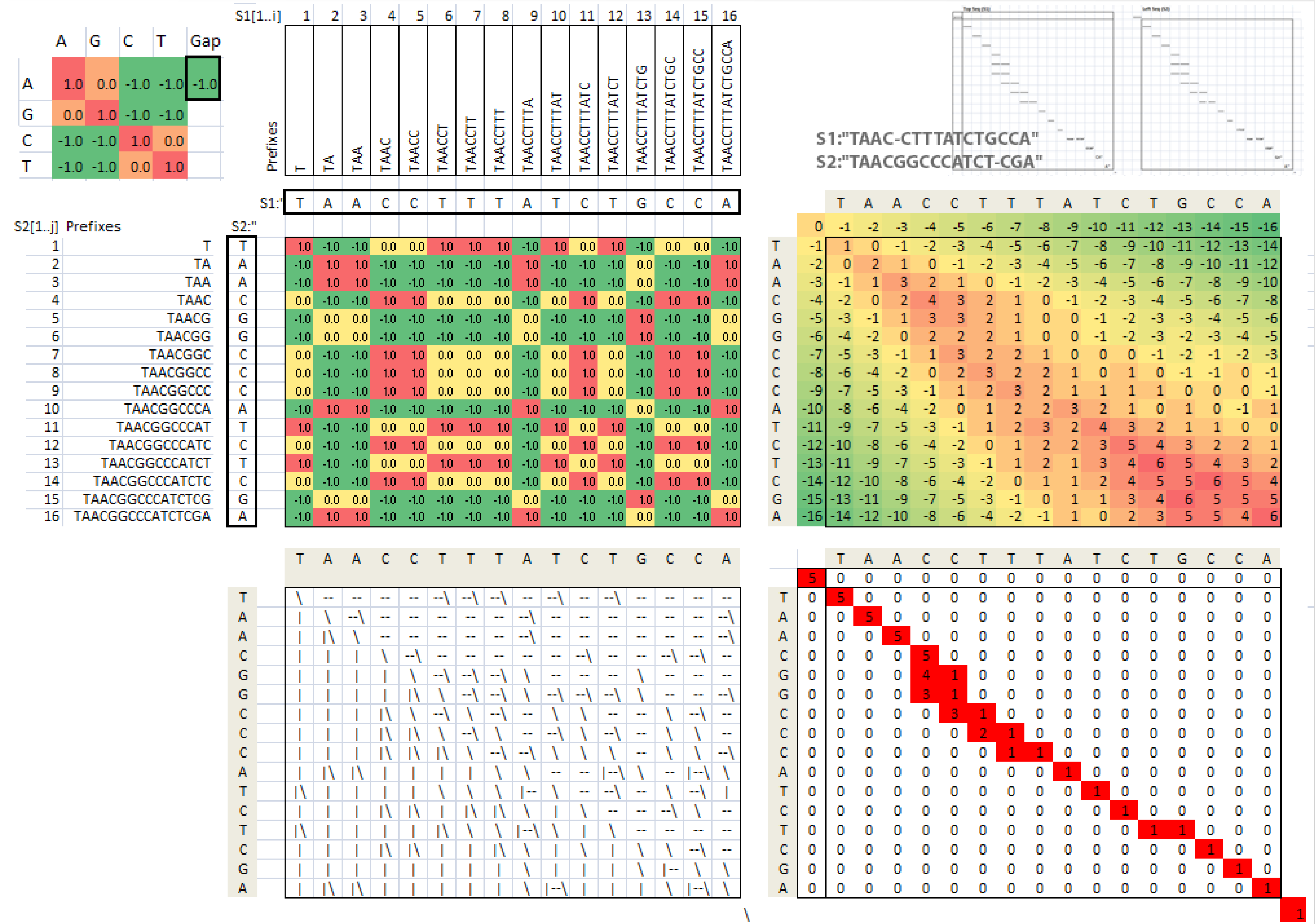
Termination:

- Bottom right

- - - - Path segments that lead to locally optimal choices

— — Path segments that lead to the globally optimal solution

Genome alignment in an excel spreadsheet



	A	G	C	T	Gap
A	1.0	0.0	-1.0	-1.0	-1.0
G	0.0	1.0	-1.0	-1.0	
C	-1.0	-1.0	1.0	0.0	
T	-1.0	-1.0	0.0	1.0	

		S1[1..i]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		Prefixes	T	TA	TAA	TAAC	TAACC	TAACCT	TAACCTT	TAACCTTT	TAACCTTTA	TAACCTTTAT	TAACCTTTATC	TAACCTTTATCT	TAACCTTTATCTG	TAACCTTTATCTGC	TAACCTTTATCTGCC	TAACCTTTATCTGCCA
		S1:"	T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
S2[1..j]	Prefixes	S2:"																
1	T	T	1.0	-1.0	-1.0	0.0	0.0	1.0	1.0	1.0	-1.0	1.0	0.0	1.0	-1.0	0.0	0.0	-1.0
2	TA	A	-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0
3	TAA	A	-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0
4	TAAC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
5	TAACG	G	-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	0.0
6	TAACGG	G	-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	0.0
7	TAACGGC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
8	TAACGGCC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
9	TAACGGCCC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
10	TAACGGCCCA	A	-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0
11	TAACGGCCCAT	T	1.0	-1.0	-1.0	0.0	0.0	1.0	1.0	1.0	-1.0	1.0	0.0	1.0	-1.0	0.0	0.0	-1.0
12	TAACGGCCCATC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
13	TAACGGCCCATCT	T	1.0	-1.0	-1.0	0.0	0.0	1.0	1.0	1.0	-1.0	1.0	0.0	1.0	-1.0	0.0	0.0	-1.0
14	TAACGGCCCATCTC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
15	TAACGGCCCATCTCG	G	-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	0.0
16	TAACGGCCCATCTCGA	A	-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0

S1:"TAAC-CTTTATCTGCCA"
S2:"TAACGGCCCATCT-CGA"

		T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16
T	-1	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14
A	-2	0	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
A	-3	-1	1	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
C	-4	-2	0	2	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
G	-5	-3	-1	1	3	3	2	1	0	0	-1	-2	-3	-3	-4	-5	-6
G	-6	-4	-2	0	2	2	2	1	0	0	-1	-2	-3	-2	-3	-4	-5
C	-7	-5	-3	-1	1	3	2	2	1	0	0	0	-1	-2	-1	-2	-3
C	-8	-6	-4	-2	0	2	3	2	2	1	0	1	0	-1	-1	0	-1
C	-9	-7	-5	-3	-1	1	2	3	2	1	1	1	1	0	0	0	-1
A	-10	-8	-6	-4	-2	0	1	2	2	3	2	1	0	1	0	-1	1
T	-11	-9	-7	-5	-3	-1	1	2	3	2	4	3	2	1	1	0	0
C	-12	-10	-8	-6	-4	-2	0	1	2	2	3	5	4	3	2	2	1
T	-13	-11	-9	-7	-5	-3	-1	1	2	1	3	4	6	5	4	3	2
C	-14	-12	-10	-8	-6	-4	-2	0	1	1	2	4	5	5	6	5	4
G	-15	-13	-11	-9	-7	-5	-3	-1	0	1	1	3	4	6	5	5	5
A	-16	-14	-12	-10	-8	-6	-4	-2	-1	1	0	2	3	5	5	4	6

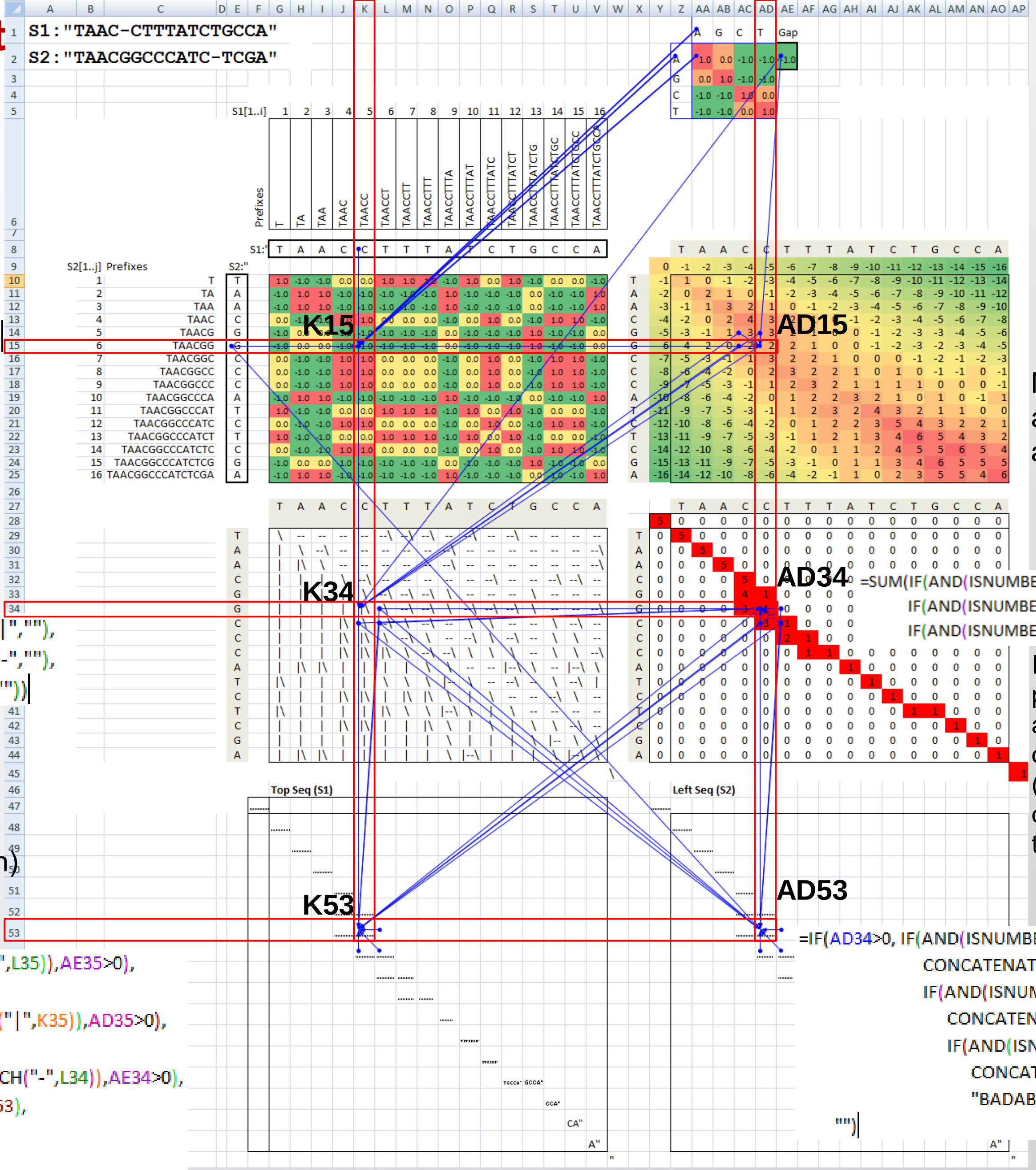
<table>
<tr><th></th><th>T</th><th>A</th><th>A</th><th>C</th><th>C</th><th>T</th><th>T</th><th>T</th><th>A</th><th>T</th><th>C</th><th>T</th><th>G</th><th>C</th><th>C</th><th>A</th></tr>
<tr><td>T</td><td>\</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--\</td><td>--\</td><td>--\</td><td>--</td><td>--\</td><td>--</td><td>--\</td><td>--</td><td>--</td><td>--</td><td>--</td></tr>
<tr><td>A</td><td>|</td><td>\</td><td>--\</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--\</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--\</td></tr>
<tr><td>A</td><td>|</td><td>|\</td><td>\</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--\</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--\</td></tr>
<tr><td>C</td><td>|</td><td>|</td><td>|</td><td>\</td><td>--\</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--</td><td>--\</td><td>--</td><td>--</td><td>--\</td><td>--\</td><td>--</td></tr>
<tr><td>G</td><td>|</td><td>|</td><td>|</td><td>|</td><td>\</td><td>--\</td><td>--\</td><td>--\</td><td>\</td><td>--</td><td>--</td><td>--</td><td>\</td><td>--</td><td>--</td><td>--</td></tr>
<tr><td>G</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|\</td><td>\</td><td>--\</td><td>--\</td><td>\</td><td>--\</td><td>--\</td><td>--\</td><td>\</td><td>--</td><td>--</td><td>--\</td></tr>
<tr><td>C</td><td>|</td><td>|</td><td>|</td><td>|\</td><td>\</td><td>--\</td><td>\</td><td>--\</td><td>--</td><td>\</td><td>\</td><td>--</td><td>--</td><td>\</td><td>--\</td><td>--</td></tr>
<tr><td>C</td><td>|</td><td>|</td><td>|</td><td>|\</td><td>|\</td><td>\</td><td>--\</td><td>\</td><td>--</td><td>--\</td><td>\</td><td>--\</td><td>--</td><td>\</td><td>\</td><td>--</td></tr>
<tr><td>C</td><td>|</td><td>|</td><td>|</td><td>|\</td><td>|\</td><td>|\</td><td>\</td><td>--\</td><td>--\</td><td>\</td><td>\</td><td>\</td><td>--</td><td>\</td><td>\</td><td>--\</td></tr>
<tr><td>A</td><td>|</td><td>|\</td><td>|\</td><td>|</td><td>|</td><td>|</td><td>|</td><td>\</td><td>\</td><td>--</td><td>--</td><td>|--\</td><td>\</td><td>--</td><td>|--\</td><td>\</td></tr>
<tr><td>T</td><td>|\</td><td>|</td><td>|</td><td>|</td><td>|</td><td>\</td><td>\</td><td>\</td><td>|--</td><td>\</td><td>--</td><td>--\</td><td>--</td><td>\</td><td>--\</td><td>|</td></tr>
<tr><td>C</td><td>|</td><td>|</td><td>|</td><td>|\</td><td>|\</td><td>|</td><td>|\</td><td>|\</td><td>\</td><td>|</td><td>\</td><td>--</td><td>--</td><td>--\</td><td>\</td><td>--</td></tr>
<tr><td>T</td><td>|\</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|\</td><td>\</td><td>\</td><td>|--\</td><td>\</td><td>|</td><td>\</td><td>--</td><td>--</td><td>--</td><td>--</td></tr>
<tr><td>C</td><td>|</td><td>|</td><td>|</td><td>|\</td><td>|\</td><td>|</td><td>|</td><td>|\</td><td>\</td><td>|</td><td>\</td><td>|</td><td>\</td><td>\</td><td>--\</td><td>--</td></tr>
<tr><td>G</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>\</td><td>|</td><td>|</td><td>|</td><td>\</td><td>|--</td><td>\</td><td>\</td></tr>
<tr><td>A</td><td>|</td><td>|\</td><td>|\</td><td>|</td><td>|</td><td>|</td><td>|</td><td>|</td><td>\</td><td>|--\</td><td>|</td><td>|</td><td>|</td><td>\</td><td>|--\</td><td>\</td></tr>
</table>

\

		T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

1

Genome alignment in an excel spreadsheet



S1:"TAAC-CTTTATCTGCCA"

S2:"TAACGGCCCATC-TCGA"

	A	G	C	T	Gap
A	1.0	0.0	-1.0	-1.0	-1.0
G	0.0	1.0	-1.0	-1.0	
C	-1.0	-1.0	1.0	0.0	
T	-1.0	-1.0	0.0	1.0	

S2[1..j]	Prefixes	S2:"	T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
1	T	T	1.0	-1.0	-1.0	0.0	0.0	1.0	1.0	1.0	-1.0	1.0	0.0	1.0	-1.0	0.0	0.0	-1.0
2	TA	A	-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0
3	TAA	A	-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0
4	TAAC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
5	TAACG	G	-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	0.0
6	TAACGG	G	-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	0.0
7	TAACGGC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
8	TAACGGCC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
9	TAACGGCCC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
10	TAACGGCCCA	A	-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0
11	TAACGGCCCAT	T	1.0	-1.0	-1.0	0.0	0.0	1.0	1.0	1.0	-1.0	1.0	0.0	1.0	-1.0	0.0	0.0	-1.0
12	TAACGGCCCATC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
13	TAACGGCCCATCT	T	1.0	-1.0	-1.0	0.0	0.0	1.0	1.0	1.0	-1.0	1.0	0.0	1.0	-1.0	0.0	0.0	-1.0
14	TAACGGCCCATCTC	C	0.0	-1.0	-1.0	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	-1.0	1.0	1.0	-1.0
15	TAACGGCCCATCTCG	G	-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	0.0
16	TAACGGCCCATCTCGA	A	-1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	-1.0	-1.0	-1.0	0.0	-1.0	-1.0	1.0

		T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16
T	-1	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14
A	-2	0	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
A	-3	-1	1	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
C	-4	-2	0	2	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
G	-5	-3	-1	1	3	3	2	1	0	0	-1	-2	-3	-3	-4	-5	-6
G	-6	-4	-2	0	2	2	2	1	0	0	-1	-2	-3	-2	-3	-4	-5
C	-7	-5	-3	-1	1	3	2	2	1	0	0	0	-1	-2	-1	-2	-3
C	-8	-6	-4	-2	0	2	3	2	2	1	0	1	0	-1	-1	0	-1
C	-9	-7	-5	-3	-1	1	2	3	2	1	1	1	1	0	0	0	-1
A	-10	-8	-6	-4	-2	0	1	2	2	3	2	1	0	1	0	-1	1
T	-11	-9	-7	-5	-3	-1	1	2	3	2	4	3	2	1	1	0	0
C	-12	-10	-8	-6	-4	-2	0	1	2	2	3	5	4	3	2	2	1
T	-13	-11	-9	-7	-5	-3	-1	1	2	1	3	4	6	5	4	3	2
C	-14	-12	-10	-8	-6	-4	-2	0	1	1	2	4	5	5	6	5	4
G	-15	-13	-11	-9	-7	-5	-3	-1	0	1	1	3	4	6	5	5	5
A	-16	-14	-12	-10	-8	-6	-4	-2	-1	1	0	2	3	5	5	4	6

		T	A	A	C	C	T	T	T	A	T	C	T	G	C	C	A
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	5	0	0	0	0	0							
G	0	0	0	0	4	1	0	0	0	0							
G	0	0	0	0	3	1	0	0	0	0							
C	0	0	0	0	0	3	1	0	0	0							
C	0	0	0	0	0	0	2	1	0	0							
C	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

K15

```
=INDEX($AA$2:$AD$5,
        MATCH(K$8,$Z$2:$Z$5,0),
        MATCH($E15,$AA$1:$AD$1,0))
```

Local score of matching characters $S_1[i]$ and $S_2[j]$

K34

```
=CONCATENATE(IF(AD15=AD14+$AE$2,"|",""),
             IF(AD15=AC15+$AE$2,"--",""),
             IF(AD15=AC14+K15,"\",""))
```

Is the max alignment score coming from the top ("|"), from the left ("--") or from the diagonal up ("\") (show all of them, cuz we can)

K53

```
=IF(AD34>0, IF(AND(ISNUMBER(SEARCH("\",L35)),AE35>0),
        CONCATENATE(K$8,L54),
        IF(AND(ISNUMBER(SEARCH("|",K35)),AD35>0),
          CONCATENATE("-",K54),
          IF(AND(ISNUMBER(SEARCH("-",L34)),AE34>0),
            CONCATENATE(K$8,L53),
            "BADABOOM!"))),
  "")
```

Construct the optimal alignment for sequence S_1 by adding in characters or gaps to increasingly large suffixes (and arbitrarily choose one path when multiple using nested if's)

AD15

```
=MAX(AD14+$AE$2,
     AC15+$AE$2,
     AC14+K15)
```

Max alignment score of aligning prefix $S_1[1..i]$ and prefix $S_2[1..j]$

AD34

```
=SUM(IF(AND(ISNUMBER(SEARCH("|",K35)),AD35>0),AD35,0),
     IF(AND(ISNUMBER(SEARCH("\",L35)),AE35>0),AE35,0),
     IF(AND(ISNUMBER(SEARCH("-",L34)),AE34>0),AE34,0))
```

Is the [i,j] part of an optimal path? (i.e. are chars $S_1[i]$ and $S_2[j]$ aligned to each other in an optimal path) (also count number of optimal paths/alignment through [i.j], cuz we can)

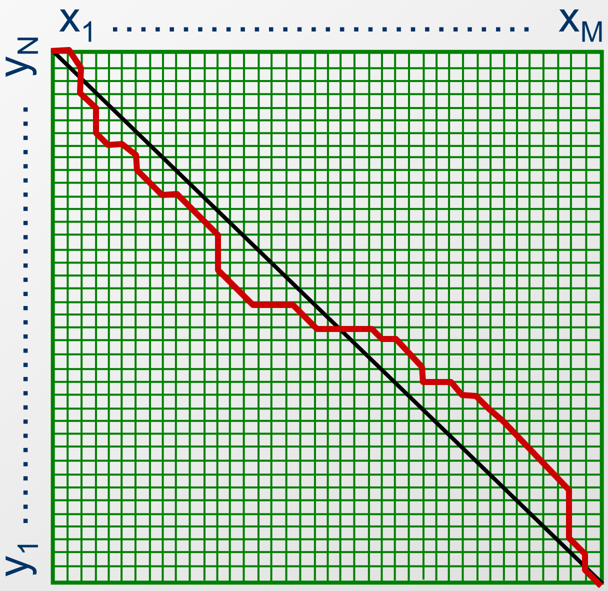
AD53

```
=IF(AD34>0, IF(AND(ISNUMBER(SEARCH("\",L35)),AE35>0),
        CONCATENATE($E15,AE54),
        IF(AND(ISNUMBER(SEARCH("|",K35)),AD35>0),
          CONCATENATE($E15,AD54),
          IF(AND(ISNUMBER(SEARCH("-",L34)),AE34>0),
            CONCATENATE("-",AE53),
            "BADABOOM!"))),
  "")
```

Construct the optimal alignment for sequence S_2 similarly to S_1

What is missing? (5) Returning the actual path!

- We know how to compute the best score
  - – Simply the number at the bottom right entry
- But we need to remember where it came from
  - – Pointer to the choice we made at each step
- Retrace path through the matrix
  - – Need to remember all the pointers



Time needed: O(m*n)

Space needed: O(m*n)

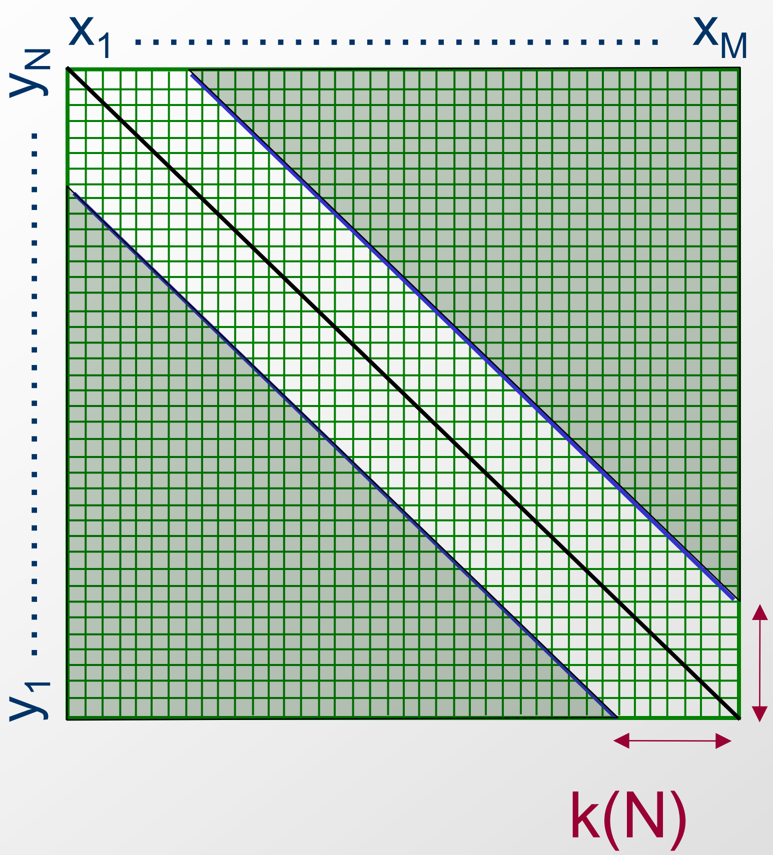
Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it’s hard: Exponential number of alignments
2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back
3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry⬄prefix alignment score; path⬄aligmnt
4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP(heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition

If time permits…

(4) Extensions to basic DP solution

Bounded Dynamic Programming



Initialization:

F(i,0), F(0,j) undefined for i, j > k

Iteration:

```
For i = 1…M
  For j = max(1, i – k)…min(N, i+k)
```

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i, j-1) - d, \text{ if } j > i - k(N) \\ F(i-1, j) - d, \text{ if } j < i + k(N) \end{cases}$$

Termination: same

Slides credit: Serafim Batzoglou

Can we do better than O(n^2)in the general case?

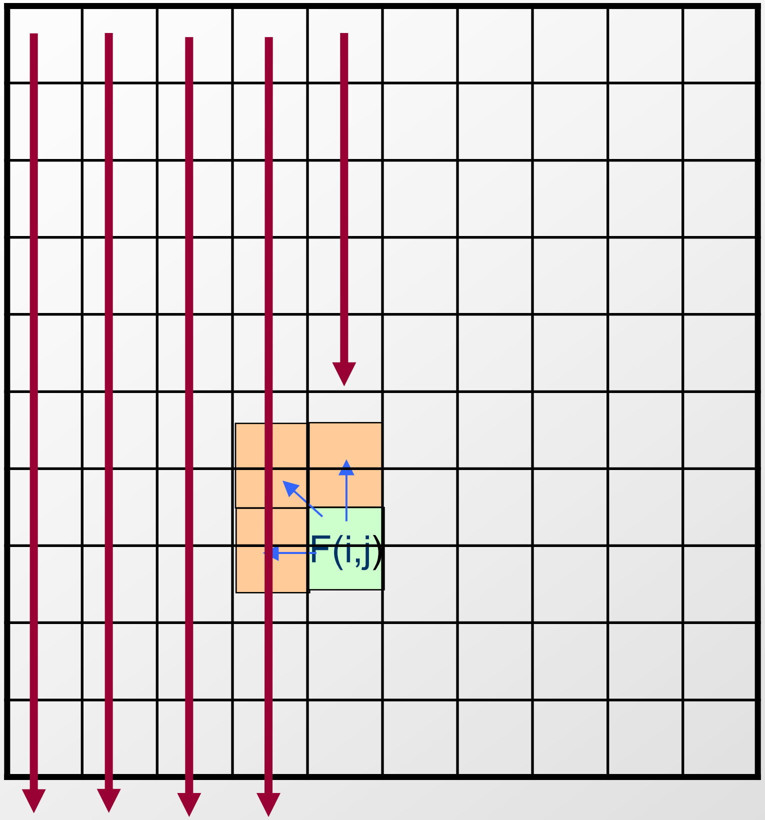
- Reduced Orthogonal Vectors to PATTERN
- Reduced PATTERN to EDIT DISTANCE
- Proved EDIT DISTANCE is a SETH-hard problem

Abstract removed due to copyright restrictions.
Source: Backurs, Arturs, and Piotr Indyk. "Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)." In Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, pp. 51-58. ACM, 2015.

- Faster edit dist. algorithm probably not a good term project

Linear space alignment

It is easy to compute F(M, N) in linear space



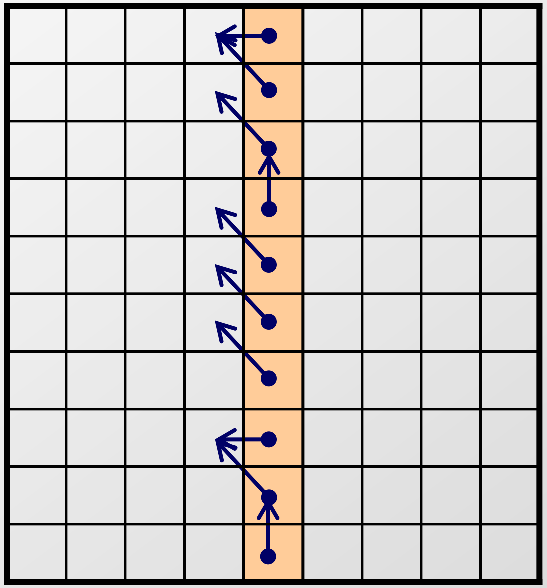
```
Allocate ( column[1] )
Allocate ( column[2] )

For   i = 1….M
    If     i > 1, then:
           Free( column[i – 2] )
           Allocate( column[ i ] )
    For  j = 1…N
           F(i, j) = …
```

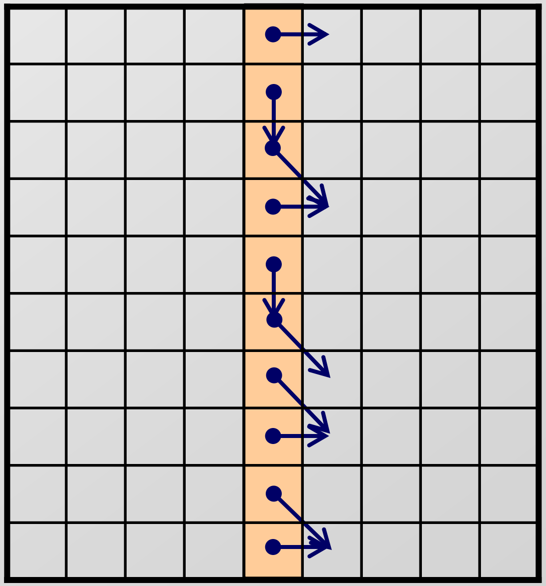
What about the pointers?

Finding the best back-pointer for current column

- Now, using 2 columns of space, we can compute for k = 1…M, F(M/2, k), F^r(M/2, N-k)

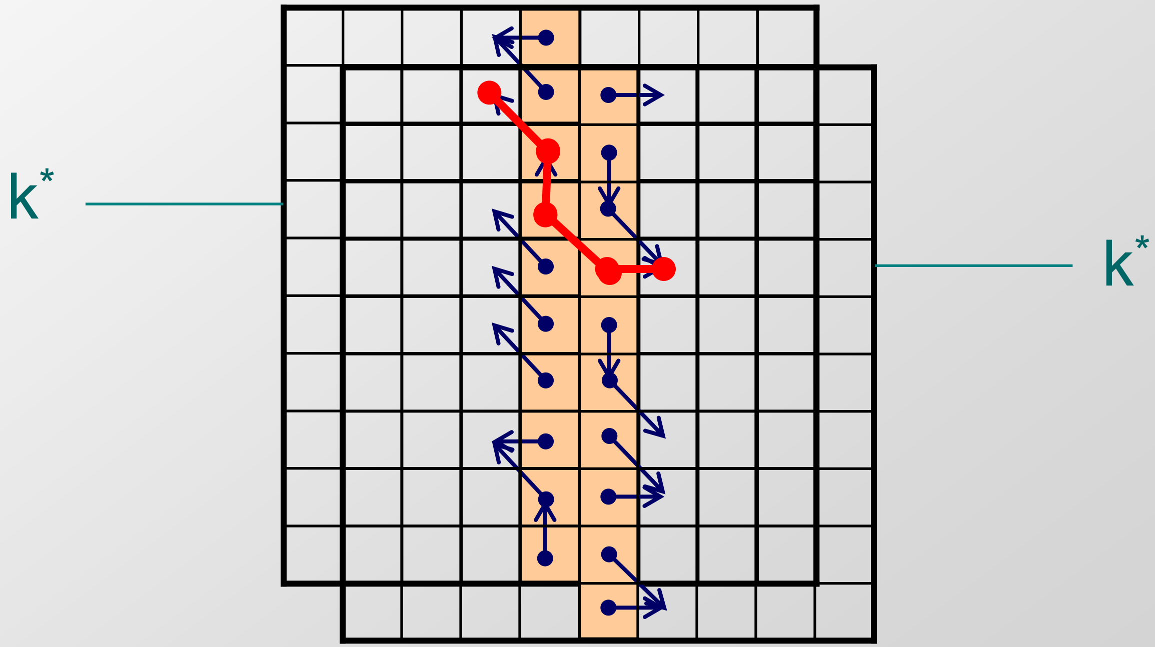
  PLUS the backpointers





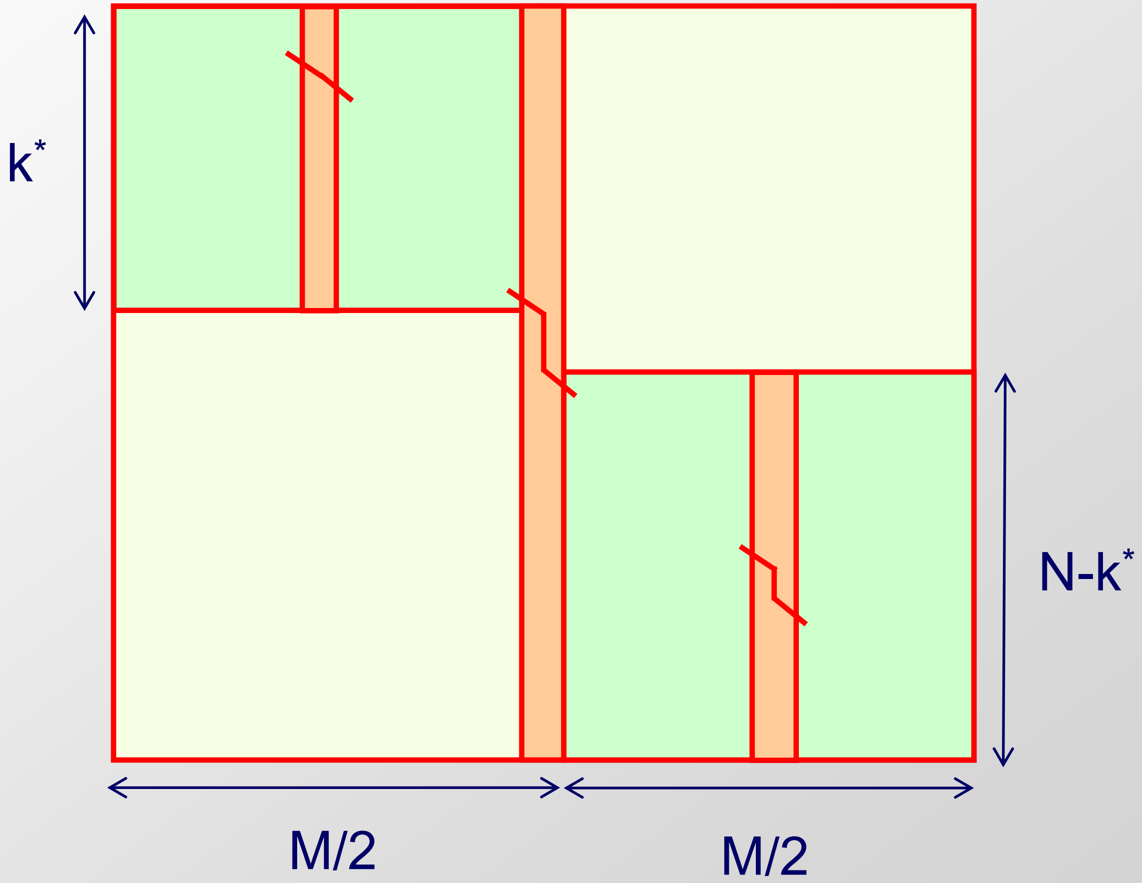
Best forward-pointer for current column

- Now, we can find k^* maximizing $F(M/2, k) + F^r(M/2, N-k)$
- Also, we can trace the path exiting column M/2 from k^*

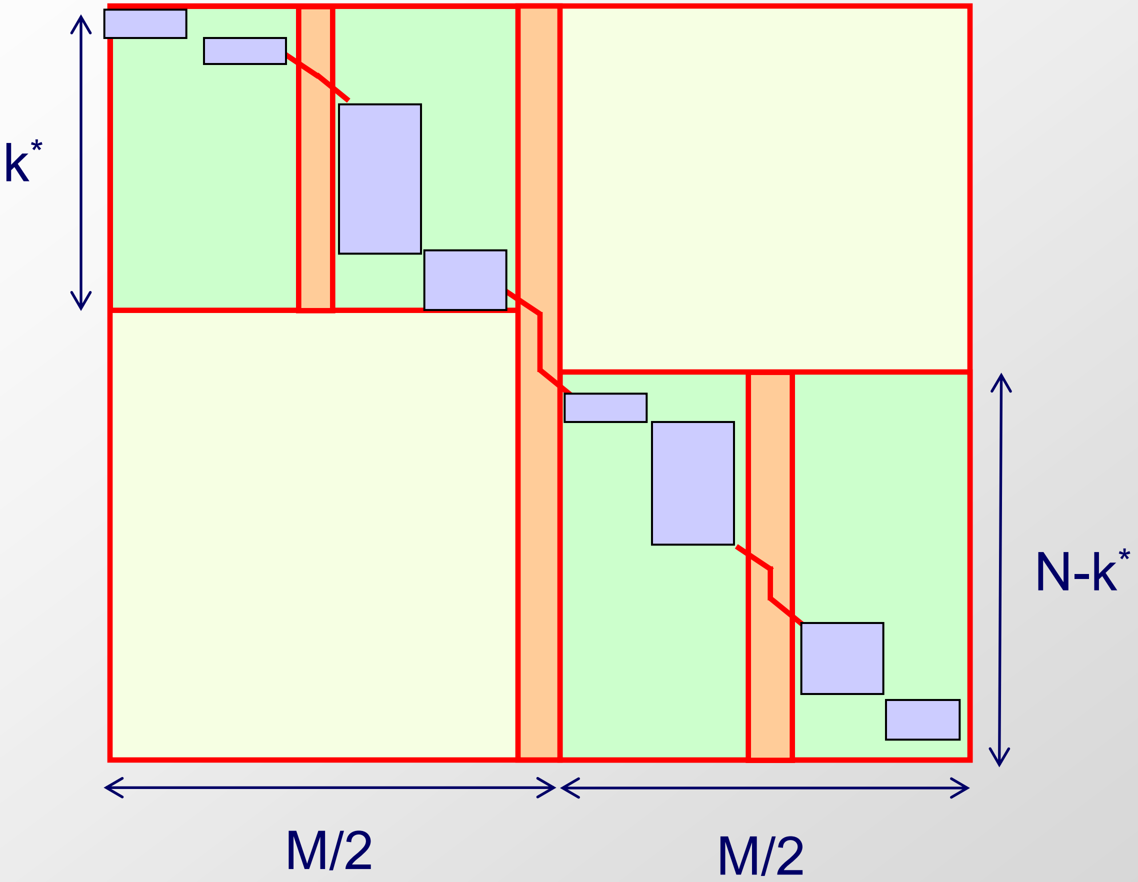


Recursively find midpoint for left & right

- Iterate this procedure to the left and right!



Total time cost of linear-space alignment



Total Time: cMN + cMN/2 + cMN/4 + ….. = 2cMN = O(MN)

Total Space: O(N) for computation,

O(N+M) to store the optimal alignment

Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it’s hard: Exponential number of alignments
2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back
3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry⇔prefix alignment score; path⇔aligmnt
4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP(heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition

Additional insights

Why the 2-dimentional parameterization worked

Summary

- Dynamic programming
  - Reuse of computation
  - Order sub-problems. Fill table of sub-problem results
  - Read table instead of repeating work (ex: Fibonacci)
- Sequence alignment
  - Edit distance and scoring functions
  - Dynamic programming matrix
  - Matrix traversal path ⇔ Optimal alignment
- Thursday: Variations on sequence alignment
  - Local/global alignment, affine gaps, algo speed-ups
  - Semi-numerical alignment, hashing, database lookup
- Recitation:
  - Dynamic programming applications
  - Probabilistic derivations of alignment scores

Goal: Sequence Alignment / Dynamic Programming

1. Introduction to sequence alignment
   - Comparative genomics and molecular evolution
   - From Bio to CS: Problem formulation
   - Why it’s hard: Exponential number of alignments
2. Introduction to principles of dynamic programming
   - Computing Fibonacci numbers: Top-down vs. bottom-up
   - Repeated sub-problems, ordering compute, table lookup
   - DP recipe: (1) Parameterization, (2) sub-problem space, (3) traversal order, (4) recursion formula, (5) trace-back
3. DP for sequence alignment
   - Additive score, building up a solution from smaller parts
   - Prefix matrix: finite subproblems, exponential paths
   - Duality: each entry⇔prefix alignment score; path⇔aligmnt
4. Advanced topics: Dynamic Programming variants
   - Linear-time bounded DP(heuristic). Linear-space DP. Gaps
   - Importance of parameterization: 2-D vs. 4-D decomposition