

Agenda:

- Welcome to CS 238 —
Algorithmic Techniques in Computational Biology
- Official course information
- Course description
- Announcements
- Basic concepts in molecular biology

Official course information

- Grading weights:
 - 50% assignments (3-4)
 - 50% participation, presentation, and course report.
- Text and reference books:
 - D. Gusfield, *Algorithms for Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge Press, 1997.
 - T. Jiang, Y. Xu and M. Zhang (co-eds), *Current Topics in Computational Biology*, MIT Press, 2002. (co-published by Tsinghua University Press in China)
 - D. Krane and M. Raymer, *Fundamental Concepts of Bioinformatics*, Benjamin Cummings, 2003.
 - P. Pevzner, *Computational Molecular Biology: An Algorithmic Approach*, 2000, the MIT Press.
 - M. Waterman, *Introduction to Computational Biology: Maps, Sequences and Genomes*, Chapman and Hall, 1995.
 - These notes (typeset by Guohui Lin and Tao Jiang).
- Instructor: Tao Jiang
 - Surge Building 330, x82991, jiang@cs.ucr.edu
 - Office hours: Tuesday & Thursday 3-4pm

Course overview

- Topics covered:
 - Biological background introduction
 - My research topics
 - Sequence homology search and comparison
 - Sequence structural comparison
 - string matching algorithms and suffix tree
 - Genome rearrangement
 - Protein structure and function prediction
 - Phylogenetic reconstruction
 - * *DNA sequencing, sequence assembly*
 - * *Physical/restriction mapping*
 - * *Prediction of regulatory elements*
- Announcements:
 - Assignment #1 available soon.
 - Suggestions for topics of class presentation will be provided in a few weeks.
 - Finalize your presentation topic before mid quarter.

The Secrets of Life (also in Krane and Raymer, 2003)
(A mathematician's introduction to molecular biology)

- Basic concepts: DNA, RNA, enzymes
- DNA (deoxyribonucleic acid)
 - discovered by **James Watson** and **Francis Crick**, 40 years ago
 - DNA discovery \Rightarrow modern genetic engineering
 - at breathtaking speed
- astonishing scientific & practical implications
 1. DNA sequence \rightarrow the evolutionary record of life
 2. genes for human insulin
 \rightarrow bacteria \rightarrow protein (inexpensive large amount & pure)
 3. farm animal & crops
 \rightarrow healthier & more desirable products
 4. diagnosis for viral disease (e.g. AIDS)
sensitive & reliable
- molecular biology: the character of the field is changing
experimental science \rightarrow theoretical science
databases of DNA and protein sequences
- the mathematical sciences — mathematics, statistics, and computational science are playing an important role

The Composition of a Living Organism

- Protein, Nucleic Acid, Lipid, Carbohydrate
- The human makes 100,000 proteins
 - enzymes — digestion of food
 - structural molecular — hair, skin
 - transporter of substances — hemoglobin
 - transporter of information — receptor in surface of cells
- **Protein do the work of the cell!**
a linear chain of amino acids (20 types)

5

Classical Genetics

Geneticists study mutant organisms that differ in one component.

Gregor Mendel 's experiments on pure breeding peas — 1865

Trait (phenotype): round ○ or wrinkled ⊗

F_0	pure breeding ○ or ⊗	
cross to generate F_1	○ (100%)	
cross F_1 with ⊗	○ (50%)	⊗ (50%)
cross F_1 with F_1	○ (75%)	⊗ (25%)

genotype:

F_0	○ AA	⊗ aa
F_1	○ Aa (100%)	
F_2	○ AA (25%)	○ Aa (50%) ⊗ aa (25%)

Mendel Hypothesis (gene)

- each organism inherits two copies of a gene
- genes occur in alternative forms — **alleles**, with different biological functions
- pure breeding plants carried two copies of identical alleles (AA, aa) and are called **homozygotes**.
the F_1 plants have Aa and are **heterozygotes**.
- roundness (A) dominates over wrinkledness (a)
- inheritance is random.

6

Biochemistry

- Role of biochemistry:

biochemistry (protein)

molecular biology

genetics (gene)

→

→

→

function
- What is biochemistry?
Goal: Purifying and characterizing the chemical components responsible for carrying out a particular function.
- What is a biochemist doing?
Devises an array for measuring an "activity" and then tries successive fractionation procedures to isolate a pure fraction having the activity.

7

Molecular Biology

Scientists realized that Genes encode enzymes / proteins.

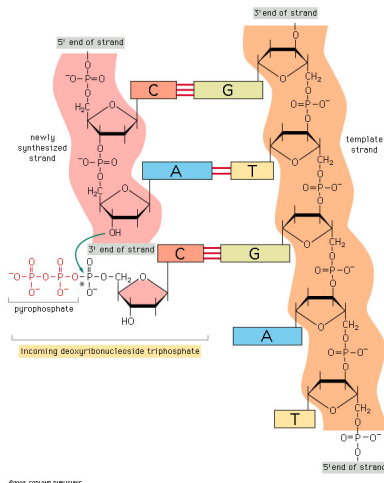
What are genes made of? — DNA

Watson & Crick, 1953

- Double helix
- Nucleotides: Adenine, Guanine, Cytosine, Thymine
- Base pairs: A–T, C–G
- Replication of DNA: the two strands unwind and each serves as a template
- DNA polymerase: carries out replication

8

DNA (Deoxyribonucleic Acid)

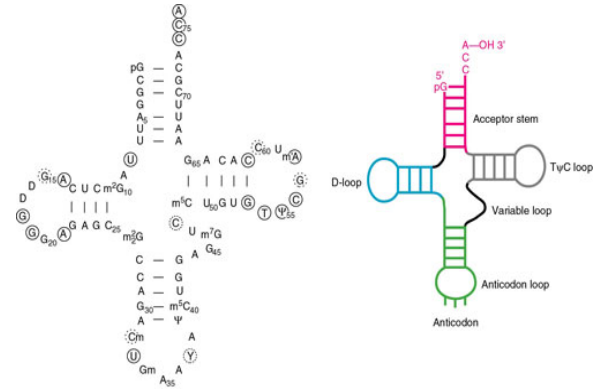


The DNA sequence is ...CACTA...

It may encode the message **dark hair**.

9

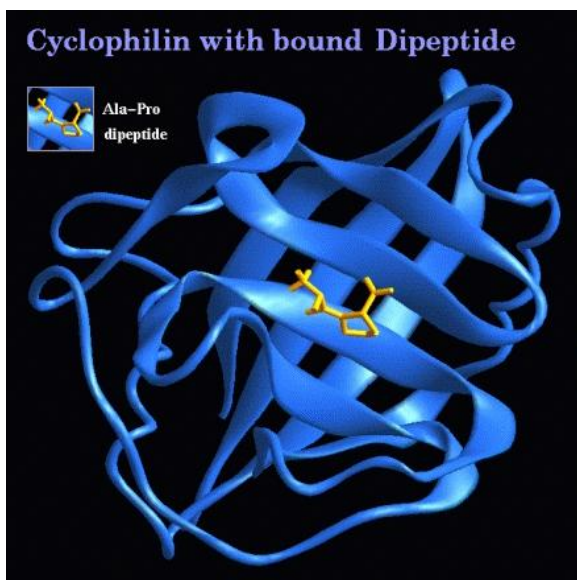
RNA (Ribonucleic Acid)



The RNA sequence is GCGGAUUGCUC...

10

Protein



The protein sequence is KVLAAALIAGSVFFLLLPG...

The elements are called *amino acids*.

11

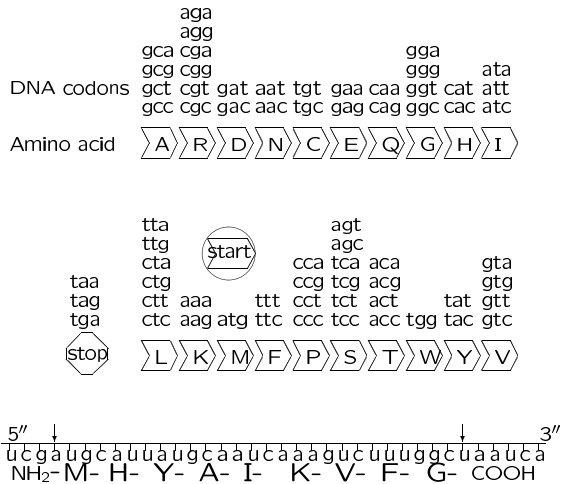
Synthesizing a Protein (in prokaryote)

- A gene is expressed with the transcription of the DNA sequence into a messenger RNA (mRNA). This is carried out by RNA polymerase. An RNA is made of A, U, C, G, and is single stranded.
- Each triple of A/U/C/G encodes an amino acid, called a codon.
- The mRNA is then translated into a protein by the ribosome.



12

Pattern Matching in Nature



13

Areas of Computational Biology

1. DNA sequencing — determine DNA sequences
 - shotgun sequencing
 - sequencing by hybridization
 - sequencing by mass spectrometry (for protein fragments)
2. Physical mapping — low resolution information about genome
 - STS content mapping
 - restriction mapping
 - radiation hybridization mapping
 - optical mapping
 - fluorescent in situ hybridization (FISH)
3. Sequence analysis — extract functional/evolutionary information from DNA/protein sequences
 - sequence comparison
 - gene recognition
 - regulatory signal analysis
 - sequence databases

14

Areas of Computational Biology – continued

4. Comparative genomics, phylogenetic analysis — recovering history of evolution
5. Protein folding, structural biology — determine secondary and tertiary structures of protein sequences
 - crystallography
 - NMR data
 - energy minimization
 - protein threading
6. Genetic linkage — relative locations of genes
 - SNP analysis
 - haplotyping
7. Gene expression arrays and DNA microarrays — functional genomics
 - How do genes relate to functions?
 - How do genes relate to diseases?
 - When are genes expressed and co-expressed?

15

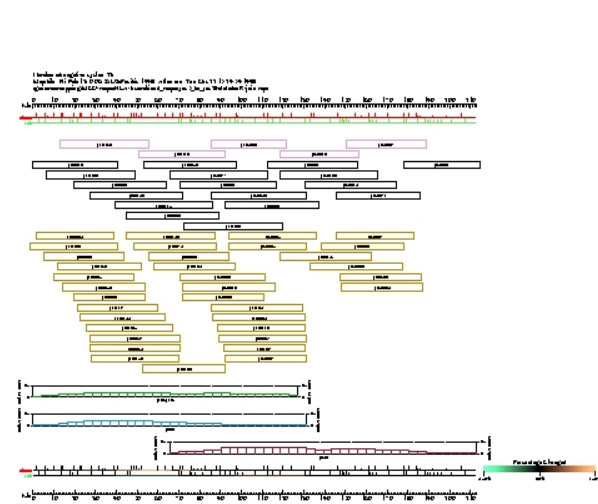
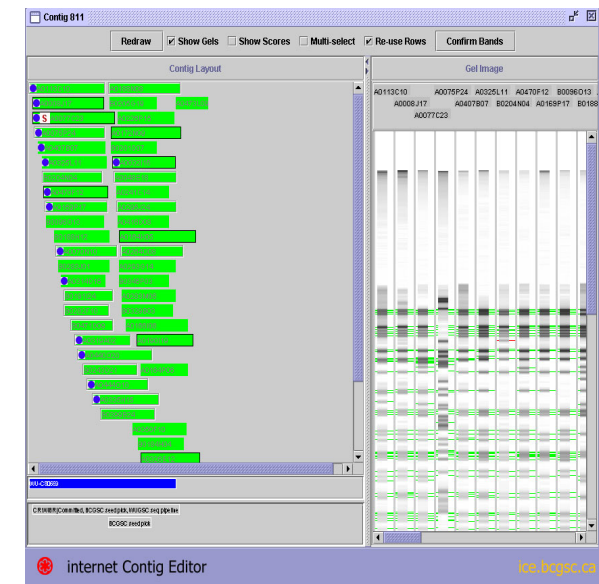
Areas of Computational Biology – continued

8. Gene networks, metabolism pathways
 - How do genes relate to / control each other?
9. Genetic drug discovery, combinatorial chemistry
 - Identify drug targets from molecular information (sequence, 3D structure, etc.)

16

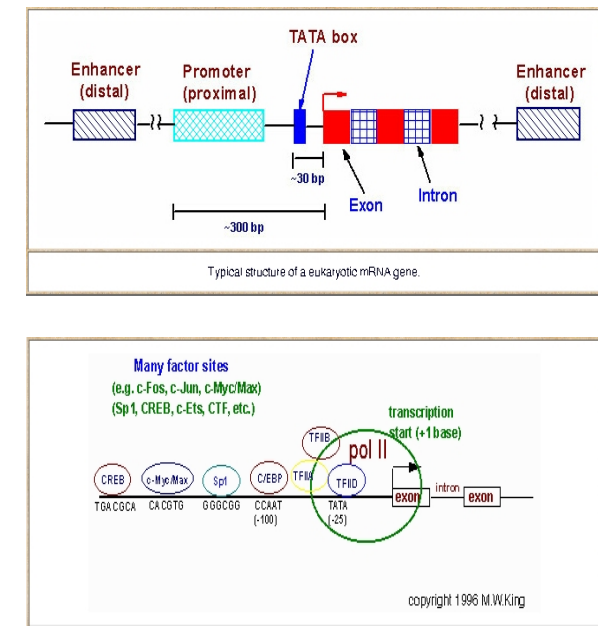
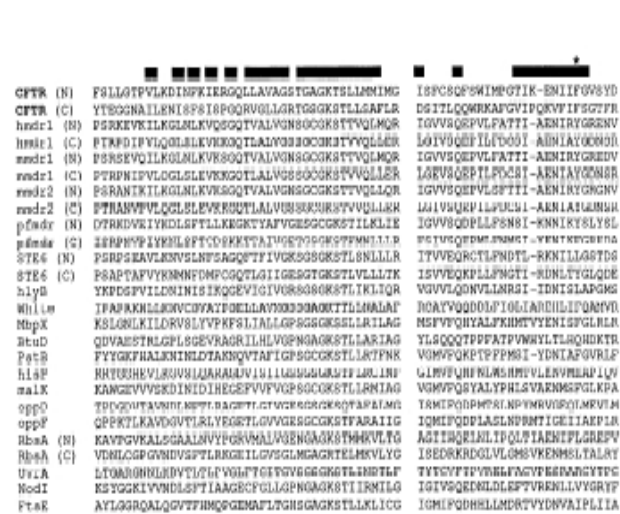
Sequence Assembly

Physical mapping - a restriction (MCD) map

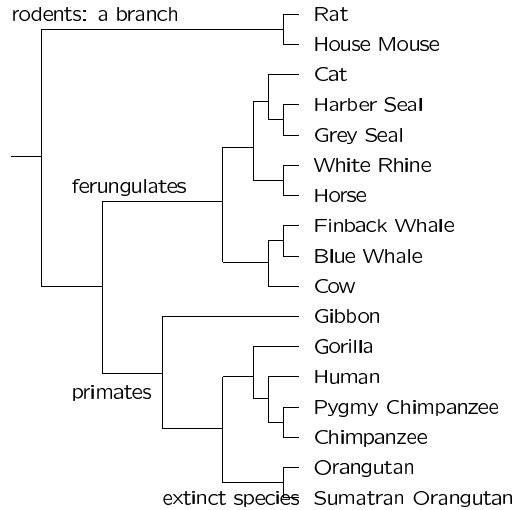


Sequence Alignment

Gene regulation



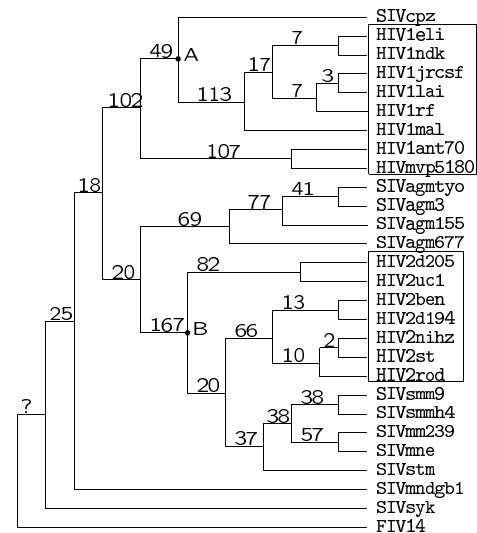
Phylogeny (evolutionary tree) reconstruction



- what data? (mitochondrial genomes)
- which method?

21

gap p24 (2390bp) and *pol* (705bp) nucleotide sequences combined.

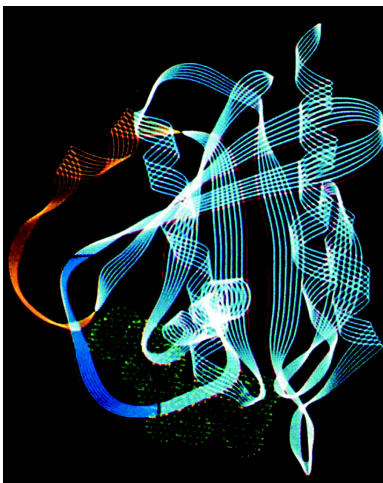


A phylogeny of immunodeficiency viruses [D. Mindell, 1994]

- organism / species: DNA / RNA / protein sequence
- distance between sequences: estimate the time of evolution on a molecular clock
- objective function: minimize the distance

22

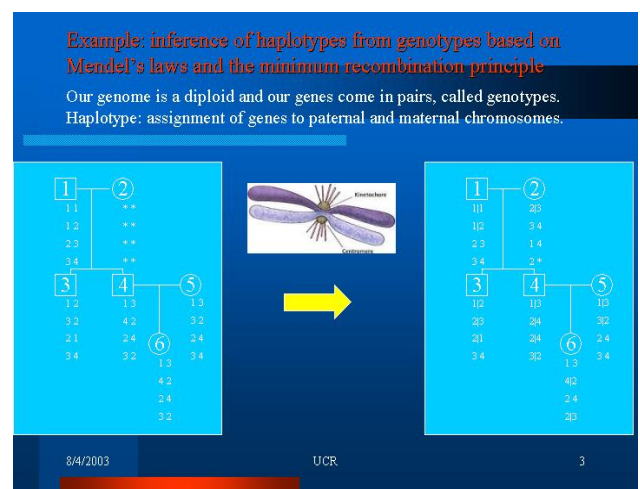
Protein Structure



This is the *ras* protein. Knowing the 3-D structure of this important molecular switch governing cell growth may enable interventions to shut off this switch in cancer cells.

23

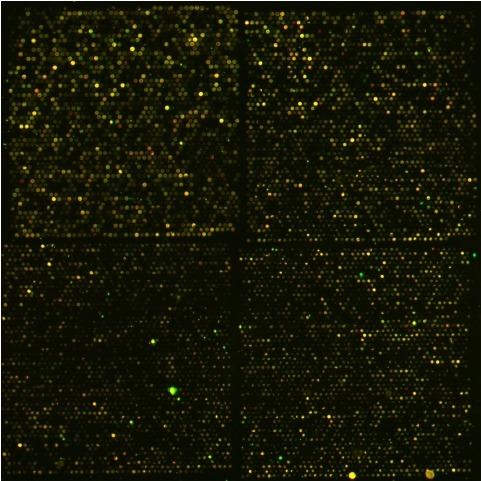
Inference of haplotypes from genotypes



Haplotype information is useful in fine gene mapping, association studies, etc.

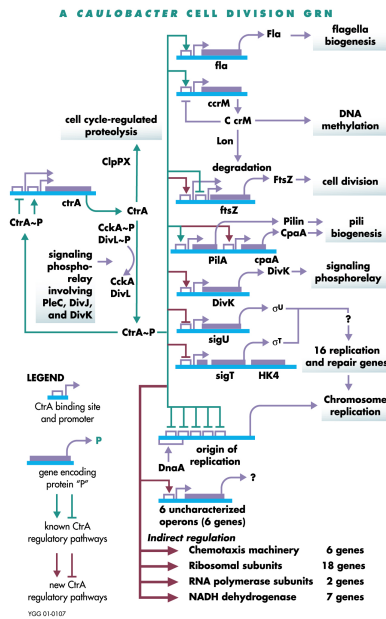
24

Gene Expression Array

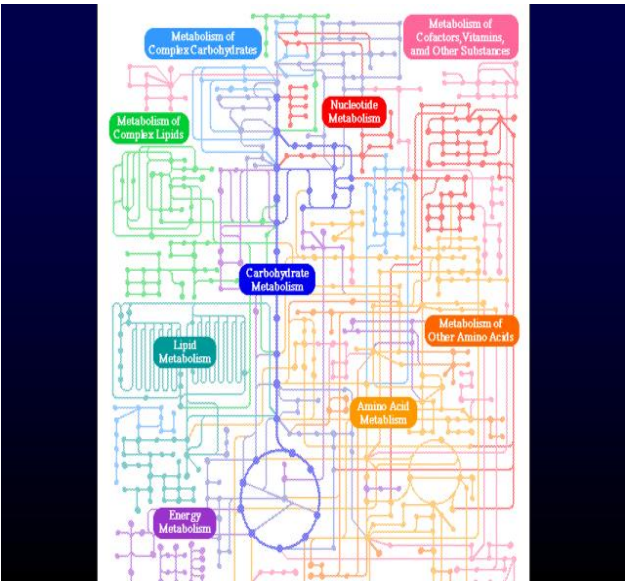


A 9000 gene sequence verified mouse microarray.

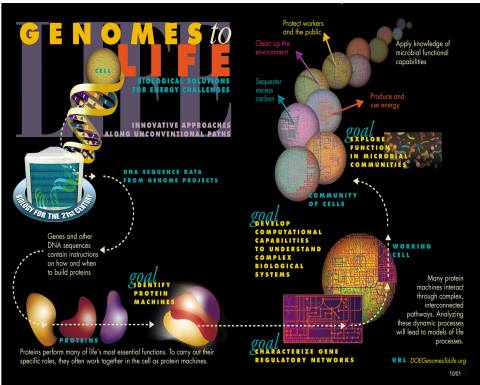
Gene network



Gene network

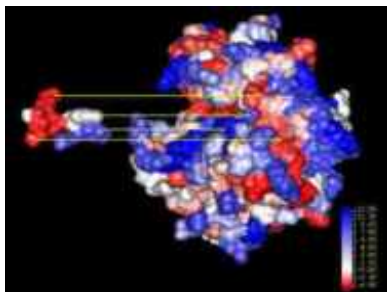


Functional genomics



Courtesy U.S. Department of Energy Genomes to Life program: DOEGenomesToLife.org

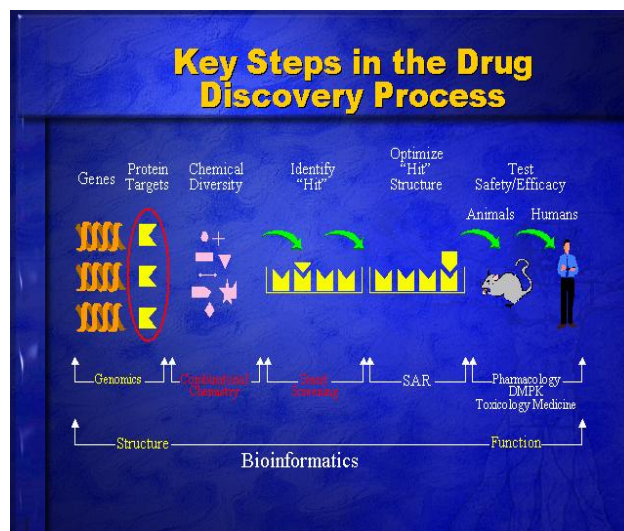
Drug Discovery



The cellular targets (or receptors) of many drugs used for medical treatment are proteins. By binding to the receptor, drugs either enhance or inhibit its activity.

29

Drug Discovery



30

Research Projects in Jiang's Lab

- Efficient haplotyping on a pedigree
 - polynomial-time algorithm for 0-recombinant data
 - missing alleles imputation using LD information
- Dynamics of gene evolution on plant genomes
 - comparison of 12 chloroplast genomes
 - analysis of Myb families on Arabidopsis and rice genomes
- Search for TFBS in the human genome
 - an "optimized" markov chain algorithm
 - prediction + *in vitro* validation doubled known HNF4 α binding sequences
- Classification of microbial communities by oligonucleotide fingerprinting on DNA arrays
 - design of probe sets
 - a discrete approach to clustering
- Sequence annotation and comparison
 - comparison of RNA structures
 - prediction of operons in *Synechococcus* Sp.
- Analysis of barley EST database
 - design of popular and unique oligo probes
- automatic NMR spectral peak assignment

31

Lecture 4: Pairwise Sequence Alignment

Agenda:

- Dynamic programming concept, review
- Edit distance between two sequences
- Global alignment
- Optimal alignment in linear space

Reading:

- Ref[JXZ, 2002]: Pages 45 – 69
- Ref[Gusfield, 1997]: Pages 209 – 258
- Ref[CLRS, 2001]: Pages 331 – 356

32

Why sequence comparison:

- Genes diverging from a common ancestor are *homologous*.
- Homologous genes perform the same or similar functions.
- Homologous gene sequences are *similar*.
 - On average, human and mouse homologous gene sequences are 85% similar.
- Similarity:
 - How to define it
 - How to compute it
 - twin definition — distance.

33

Matrix-chain multiplication (from CRLS):

- Input: matrices A_1, A_2, \dots, A_n with dimensions $d_0 \times d_1, d_1 \times d_2, \dots, d_{n-1} \times d_n$, respectively.
- Output: an order in which matrices should be multiplied such that the product $A_1 \times A_2 \times \dots \times A_n$ is computed using the minimum number of scalar multiplications.
- Example: $n = 4$ and $(d_0, d_1, \dots, d_n) = (5, 2, 6, 4, 3)$
- Possible orders with different number of scalar multiplications:

$((A_1 \times A_2) \times A_3) \times A_4$	$5 \times 2 \times 6 + 5 \times 6 \times 4 + 5 \times 4 \times 3 = 240$
$(A_1 \times (A_2 \times A_3)) \times A_4$	$5 \times 2 \times 4 + 2 \times 6 \times 4 + 5 \times 4 \times 3 = 148$
$(A_1 \times A_2) \times (A_3 \times A_4)$	$5 \times 2 \times 6 + 5 \times 6 \times 3 + 6 \times 4 \times 3 = 222$
$A_1 \times ((A_2 \times A_3) \times A_4)$	$5 \times 2 \times 3 + 2 \times 6 \times 4 + 2 \times 4 \times 3 = 102$
$A_1 \times (A_2 \times (A_3 \times A_4))$	$5 \times 2 \times 3 + 2 \times 6 \times 3 + 6 \times 4 \times 3 = 138$
- Number of orders in $\Omega((4 - \epsilon)^n)$!!!
 - Cannot afford exhaustive enumeration.
- Try recursion?
 - $M(i, j)$ — the minimum number of scalar multiplications needed to compute product $A_i \times A_{i+1} \times \dots \times A_j$ ($i \leq j$)
 -
$$M(i, j) = \begin{cases} 0, & \text{if } i = j \\ \min_{i \leq k < j} \{M(i, k) + M(k+1, j) + d_{i-1}d_kd_j\}, & \text{if } i < j \end{cases}$$
 - $M(1, n) \in \Omega(3^n)$

34

Matrix-chain multiplication (from CS 218):

- Observations:
 - A lot of re-computation in the recursion approach! Avoid them!!!
 - Carefully define the order of computation for $M(i, j)$, store them somewhere.
 - When needed, get the value, rather than re-computation.
- Build a table of dimension $n \times n$ to store $M(i, j)$
- Order of computation: increasing $(j - i)$
- This is **Dynamic Programming**
- Essence of dynamic programming
 - Optimal substructures
 - Overlapping subproblems
 - Number of subproblems/substructures is **polynomial** !!!
 - Given substructures, assembling an optimal structure can be done in **polynomial** time !!!

35

Defining distance — String edit:

Problem: Given two strings S_1 and S_2 , edit S_1 into S_2 with the minimum number of *edit operations*:

- Replace a letter a with b
- Insert a letter a
- Delete a letter b

Example:

S_1 = contextfree
 S_2 = test-file
 context free
 test-file

edit cost: 111001010110 = 7

The minimum number of operations is called the *edit distance* between S_1 and S_2 .

Theorem: String edit can be done in $O(n_1n_2)$ time.

Examine the last edit operation
 what about the first edit operation

36

Edit distance by Dynamic Programming:

- Satisfying DP requirements?
 - Computing edit distance between $S_1[1..n_1]$ and $S_2[1..n_2]$ reduces to
 - computing edit distance between $S_1[1..n_1-1]$ and $S_2[1..n_2]$
 - computing edit distance between $S_1[1..n_1]$ and $S_2[1..n_2-1]$, and
 - computing edit distance between $S_1[1..n_1-1]$ and $S_2[1..n_2-1]$
- Define a generalized form of the problem with a few parameters:

$D[i, j]$ — edit distance between $a_1a_2\dots a_i$ and $b_1b_2\dots b_j$, $0 \leq i \leq n_1$ and $0 \leq j \leq n_2$.
- Derive a recurrence relation:

Consider the last (rightmost) edit operation for $a_1a_2\dots a_i \rightarrow b_1b_2\dots b_j$:

 - insertion
 - deletion
 - match or replace

$$D[i, j] = \min \begin{cases} D[i-1, j] + 1 \\ D[i, j-1] + 1 \\ D[i-1, j-1] + \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{otherwise} \end{cases} \end{cases}$$

37

Edit distance by Dynamic Programming (cont'd):

- Need to order the entries $D[i, j]$ to be computed.
- Need to initialize some entries to start the computation:

$D[i, 0] = i, D[0, j] = j$
- Use a table to store the entry values.
 - $S_1 = a_1a_2\dots a_{n_1}$
 - $S_2 = b_1b_2\dots b_{n_2}$
 - DP table: $D[i, j]$ — records the edit distance between $a_1a_2\dots a_i$ and $b_1b_2\dots b_j$, for any pair $0 \leq i \leq n_1$ and $0 \leq j \leq n_2$.
- The recurrence relation for D :

$$D[i, j] = \min \begin{cases} D[i-1, j] + 1 \\ D[i, j-1] + 1 \\ D[i-1, j-1] + \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{otherwise} \end{cases} \end{cases}$$

- An example,

		f	r	e	e
	0	1	2	3	4
f	1				
i	2				
l	3				
e	4				

38

Edit distance by Dynamic Programming (cont'd):

- Need to order the entries $D[i, j]$ to be computed.
- Need to initialize some entries to start the computation:

$D[i, 0] = i, D[0, j] = j$
- Use a table to store the entry values.
 - $S_1 = a_1a_2\dots a_{n_1}$
 - $S_2 = b_1b_2\dots b_{n_2}$
 - DP table: $D[i, j]$ — records the edit distance between $a_1a_2\dots a_i$ and $b_1b_2\dots b_j$, for any pair $0 \leq i \leq n_1$ and $0 \leq j \leq n_2$.
- The recurrence relation for D :

$$D[i, j] = \min \begin{cases} D[i-1, j] + 1 \\ D[i, j-1] + 1 \\ D[i-1, j-1] + \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{otherwise} \end{cases} \end{cases}$$

- An example,

		f	r	e	e
	0	1	2	3	4
f	1	0			
i	2				
l	3				
e	4				

39

Edit distance by Dynamic Programming (cont'd):

- Need to order the entries $D[i, j]$ to be computed.
- Need to initialize some entries to start the computation:

$D[i, 0] = i, D[0, j] = j$
- Use a table to store the entry values.
 - $S_1 = a_1a_2\dots a_{n_1}$
 - $S_2 = b_1b_2\dots b_{n_2}$
 - DP table: $D[i, j]$ — records the edit distance between $a_1a_2\dots a_i$ and $b_1b_2\dots b_j$, for any pair $0 \leq i \leq n_1$ and $0 \leq j \leq n_2$.
- The recurrence relation for D :

$$D[i, j] = \min \begin{cases} D[i-1, j] + 1 \\ D[i, j-1] + 1 \\ D[i-1, j-1] + \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{otherwise} \end{cases} \end{cases}$$

- An example,

		f	r	e	e
	0	1	2	3	4
f	1	0	1	2	3
i	2				
l	3				
e	4				

40

Edit distance by Dynamic Programming (cont'd):

- Need to order the entries $D[i, j]$ to be computed.
- Need to initialize some entries to start the computation:
 $D[i, 0] = i, D[0, j] = j$
- Use a table to store the entry values.
 - $S_1 = a_1 a_2 \dots a_{n_1}$
 - $S_2 = b_1 b_2 \dots b_{n_2}$
 - DP table: $D[i, j]$ — records the edit distance between $a_1 a_2 \dots a_i$ and $b_1 b_2 \dots b_j$, for any pair $0 \leq i \leq n_1$ and $0 \leq j \leq n_2$.
- The recurrence relation for D :

$$D[i, j] = \min \begin{cases} D[i-1, j] + 1 \\ D[i, j-1] + 1 \\ D[i-1, j-1] + \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{otherwise} \end{cases} \end{cases}$$

- An example,

		f	r	e	e
f	0	1	2	3	4
i	1	0	1	2	3
l	2	1	1	2	3
e	3	2	2	2	3
e	4	3	3	2	2

41

Edit distance by Dynamic Programming (cont'd):

- Need to order the entries $D[i, j]$ to be computed.
- Need to initialize some entries to start the computation:
 $D[i, 0] = i, D[0, j] = j$
- Use a table to store the entry values.
 - $S_1 = a_1 a_2 \dots a_{n_1}$
 - $S_2 = b_1 b_2 \dots b_{n_2}$
 - DP table: $D[i, j]$ — records the edit distance between $a_1 a_2 \dots a_i$ and $b_1 b_2 \dots b_j$, for any pair $0 \leq i \leq n_1$ and $0 \leq j \leq n_2$.
- The recurrence relation for D :

$$D[i, j] = \min \begin{cases} D[i-1, j] + 1 \\ D[i, j-1] + 1 \\ D[i-1, j-1] + \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{otherwise} \end{cases} \end{cases}$$

- An example,

		f	r	e	e
f	0	1	2	3	4
i	1	0	1	2	3
l	2	1	1	2	3
e	3	2	2	2	3
e	4	3	3	2	2

42

Edit distance by Dynamic Programming (cont'd):

- Need to order the entries $D[i, j]$ to be computed.
- Need to initialize some entries to start the computation:
 $D[i, 0] = i, D[0, j] = j$
- Use a table to store the entry values.
 - $S_1 = a_1 a_2 \dots a_{n_1}$
 - $S_2 = b_1 b_2 \dots b_{n_2}$
 - DP table: $D[i, j]$ — records the edit distance between $a_1 a_2 \dots a_i$ and $b_1 b_2 \dots b_j$, for any pair $0 \leq i \leq n_1$ and $0 \leq j \leq n_2$.
- The recurrence relation for D :

$$D[i, j] = \min \begin{cases} D[i-1, j] + 1 \\ D[i, j-1] + 1 \\ D[i-1, j-1] + \begin{cases} 0 & \text{if } a_i = b_j \\ 1 & \text{otherwise} \end{cases} \end{cases}$$

- An example,

		f	r	e	e
f	0	1	2	3	4
i	1	0	1	2	3
l	2	1	1	2	3
e	3	2	2	2	3
e	4	3	3	2	2

Complexity $O(n_1 n_2)$ time and space.

43

Pseudocode for Algorithm Edit($a_1 a_2 \dots a_{n_1}, b_1 b_2 \dots b_{n_2}$)

```

int D[0..n1, 0..n2]
int P[1..n1, 1..n2]

for i ← 0 to n1 do
  D[i, 0] ← i
for j ← 0 to n2 do
  D[0, j] ← j
for i ← 1 to n1 do
  for j ← 1 to n2 do
    if ai = bj then
      d ← 0
    else
      d ← 1
    D[i, j] ← D[i-1, j] + 1
    P[i, j] ← (0, -1)
    if D[i-1, j] < D[i, j-1] then
      D[i, j] ← D[i-1, j] + 1
      P[i, j] ← (-1, 0)
    if D[i-1, j-1] + d < D[i, j] then
      D[i, j] ← D[i-1, j-1] + d
      P[i, j] ← (-1, -1)

```

Notes:

- Edit distance only: $O(n)$ space
- Backtracing the *path* from entry $[n_1, n_2]$ to entry $[0, 0]$ in array P to find an optimal series of edit operations.
- Next: reducing time & space complexity

44

Speeding up string edit by preprocessing:

Assumptions:

- Sequences $S_1 = a_1a_2 \dots a_{n_1}$ and $S_2 = b_1b_2 \dots b_{n_2}$ are from a fixed alphabet Σ with $|\Sigma| = k$.
- $n_1 \leq n_2$.
- Let $\ell = \frac{1}{2} \log_k n_2 - 1$.

Preprocessing: The Idea

- Compute an $(\ell + 1) \times (\ell + 1)$ matrix $D_{i,j}$ for each pair $a_i a_{i+1} \dots a_{i+\ell}$ and $b_j b_{j+1} \dots b_{j+\ell}$.
- $D_{i,j}$ can be computed in $O((\ell + 1)^2) = O(\log_k^2 n_2)$ time.
- How many distinct $D_{i,j}$ matrices?
Since $|\Sigma| = k$, there are $(k^{\ell+1})^2 = n_2$ of them.

Conclusion:

- Setting $\ell = \frac{1}{2} \log_{3k} n_2 - 1$.
- Preprocessing each pair on two vectors over $\{-1, 0, 1\}$:
 $O((k^{2(\ell+1)} \log_k^2 n_2) \times 3^{2(\ell+1)}) = O(n_2 \log^2 n_2)$.
- For each of the $(\frac{n_1 n_2}{\ell^2})$ entries, filling in $2(\ell + 1)$ values from the corresponding $D_{i,j}$ matrix:
 $O(\frac{n_1 n_2}{\log n_2})$.
- Preprocessing is minor and thus in total $O(\frac{n_1 n_2}{\log n_2})$ time.
- Two keys:
 - How many $D_{i,j}$ matrices are there?
 - Why preprocessing on those vectors only?

45

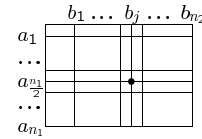
Linear space algorithm for string edit:

Ideas: Edit distance takes $O(n)$ space.

(Recall: optimal substructure?)

$$\text{Edit}(a_1 a_2 \dots a_{n_1}, b_1 b_2 \dots b_{n_2}) =$$

$$\text{Edit}(a_1 a_2 \dots a_{\frac{n_1}{2}}, b_1 b_2 \dots b_j) + \text{Edit}(a_{\frac{n_1}{2}+1} a_{\frac{n_1}{2}+2} \dots a_{n_1}, b_{j+1} b_{j+2} \dots b_{n_2}),$$

for some j .

Recursively solve for $(a_1 a_2 \dots a_{\frac{n_1}{2}}, b_1 b_2 \dots b_j)$ and $(a_{\frac{n_1}{2}+1} \dots a_{n_1-1} a_{n_1}, b_{j+1} \dots b_{n_2})$.

Determining j :

- Compute $\text{Edit}(a_1 a_2 \dots a_{\frac{n_1}{2}}, b_1 b_2 \dots b_j)$ for all j ;
- Compute $\text{Edit}(a_{n_1-1} a_{n_1} \dots a_{\frac{n_1}{2}+1}, b_{n_2} b_{n_2-1} \dots b_{j+1})$ for all j ;
- Find the j that minimizes
 $\text{Edit}(a_1 a_2 \dots a_{\frac{n_1}{2}}, b_1 b_2 \dots b_j) + \text{Edit}(a_{n_1-1} a_{n_1} \dots a_{\frac{n_1}{2}+1}, b_{n_2} b_{n_2-1} \dots b_{j+1})$

46

Linear space algorithm for string edit (cont'd):

Conclusion:

- Divide-and-conquer;
- Compute $\text{Edit}(a_1 a_2 \dots a_{\frac{n_1}{2}}, b_1 b_2 \dots b_{n_2})$;
Compute $\text{Edit}(a_{n_1-1} a_{n_1} \dots a_{\frac{n_1}{2}+1}, b_{n_2} b_{n_2-1} \dots b_1)$;
- Linear space for storing all necessary entries;
- Space complexity:
 $S(n_2) = \max\{2n_2, S(j) + S(n_2 - j)\} \in O(n_2)$.
- Time complexity:
 $T_0(n_1, n_2) = n_1 n_2$ — normal running time

$$\begin{aligned} T(n_1, n_2) &= 2 \times T_0(\frac{n_1}{2}, n_2) + T(\frac{n_1}{2}, j) + T(\frac{n_1}{2}, n_2 - j) \\ &= n_1 n_2 + 2 \times (T(\frac{n_1}{4}, j) + T(\frac{n_1}{4}, n_2 - j)) + T(\frac{n_1}{4}, \dots) \\ &= n_1 n_2 + \frac{n_1}{2} n_2 + \frac{n_1}{4} n_2 + \dots \\ &\leq 2n_1 n_2 \\ &\in O(n_1 n_2). \end{aligned}$$
- Alignment construction?
concatenating.

47

General string edit:

- Strings S_1 and S_2 over a fixed alphabet Σ
- Replacing letter a with b has a cost $\sigma(a, b)$, where $a, b \in \Sigma \cup \{-\}$
- The minimum cost of series of operations transforming S_1 into S_2 is the *edit distance* between S_1 and S_2 .
- A series of operations is equivalent to an alignment of the sequences:
 - inserting spaces $(-)$ into or at the ends of sequences
 - putting one resultant sequence on top of the other, such that every letter or space in one sequence is opposite to a unique letter or a unique space in the other sequence
 - each edit operation corresponds to a column of the alignment
 - alignment cost is the sum of costs of columns
- A series of edit operations of the minimum cost corresponds to an optimal (pairwise) sequence (global) alignment.
- Global alignment* — similarity over the *whole* sequences.

48

Lecture 5: Pairwise Sequence Alignment & Sequence Homology Search

Agenda:

- Global & local pairwise alignments
- Pairwise alignments with affine gap penalty
- The BLAST
- Pattern Hunter
- Database search demonstration

Reading:

- Ref[JXZ, 2002]: Pages 45 – 69
- Ref[Gusfield, 1997]: Pages 209 – 258
- S. F. Altschul *et al.* *Basic local alignment search tool.* Journal of Molecular Biology. (1990) 215, 403 – 410.
- S. F. Altschul *et al.* *Gapped BLAST and PSI-BLAST: a new generation of protein database search programs.* Nucleic Acids Research. (1997) 25, 3389 – 3402.
- B. Ma *et al.* *Pattern hunter: faster and more sensitive homology search.* Bioinformatics. (2002) 18, 440 – 445.
- Webpage: <http://www.ncbi.nlm.nih.gov/BLAST/>

49

Lecture 5: Pairwise Sequence Alignment

Given two sequences, form a correspondence between the letters by inserting spaces if necessary.

```
ATGCTCGCCATACA      ATGCTCGCCATA CA
ACCTCGTCATGCCA      AC CTCGTATGCCA
```

cost: 011000010001100 = 5

Assume a *similarity score/distance cost* function s .

s1 - A C G T	s2 - A C G T
- 0 1 1 1 1	- 0 2.3 2.3 2.3 2.3
A 1 0 1 1 1	A 2.3 0 1.7 1 1.7
C 1 1 0 1 1	C 2.3 1.7 0 1.7 1
G 1 1 1 0 1	G 2.3 1 1.7 0 1.7
T 1 1 1 1 0	T 2.3 1.7 1 1.7 0

The score/cost reflects the probability of mutations, e.g. $s(a, b) = -\log \text{prob}(a, b)$.

The score/cost of the alignment is the total score/cost of all columns.

The goal is to *maximize* the score or to *minimize* the cost.

The same DP algorithm for String Edit works for (global) sequence alignment in $O(mn)$ time and space.

50

Lecture 5: Pairwise Sequence Alignment

Score schemes for sequence alignment:

- The score scheme tells how to calculate the score of the alignment.
- The score scheme can measure either distance or similarity.
- Various types of score schemes have been investigated:
 - columns independent to each other:
 - * letter-independent (edit distance, BLAST)
 - * letter-dependent (PAM250, BLOSUM62, PsiBLAST)
 - gap penalties: what is a *gap*? why gap?

```
ATCGGTGGTCTCGCCAT      CA
ACCC      CTCGTATGCCA
```

cost: 0100 g(6) 00001000 g(7) 00

- * general: $g(i)$ denotes the cost of a gap of length i
 $O(n_1 n_2 (n_1 + n_2))$ [Waterman *et al.*, 1976]
- * concave:
 $O(n_1 n_2 (\log n_1 + \log n_2))$ [Miller & Myers, 1988; Eppstein *et al.*, 1989]
- * affine: $g(i) = g_{\text{open}} + i \cdot g_{\text{ext}}$
 $O(n_1 n_2)$ [Gotoh, 1982]

51

Lecture 5: Pairwise Sequence Alignment

The PAM250 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	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	-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	-4	-2	7	-5	-3	0	10						
V	0	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4				
B	0	-1	2	3	-4	1	2	0	1	-2	-3	1	-2	-5	-1	0	0	-5	-3	-2	2		
Z	0	0	1	3	-5	3	3	-1	2	-2	-3	0	-2	-5	0	0	-1	-6	-4	-2	2	3	
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

B: Aspartic D or Asparagine N;

Z: Glutamic E or Glutamine Q; and X: any undetermined amino acid.

52

Longest common subsequence (LCS):

Definitions:

- A sequence/string $S = a_1a_2 \dots a_n$
- A subsequence is any $a_{i_1}a_{i_2} \dots a_{i_k}$ where $1 \leq i_1 < i_2 < \dots < i_k \leq n$.
- $a_1a_2 \dots a_n$ is called a supersequence of $a_{i_1}a_{i_2} \dots a_{i_k}$.

Warning: Difference between subsequence and substring

Problem: Given two sequences $S_1 = a_1a_2 \dots a_{n_1}$ and $S_2 = b_1b_2 \dots b_{n_2}$, find a longest subsequence of both S_1 and S_2 .

Example:

$$\begin{array}{l} S_1 = 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \\ S_2 = 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \\ \text{LCS} = 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \end{array}$$

LCS as Sequence Alignment: Similarity score scheme: match gets 1, mismatch and indel get 0.

The maximum score alignment is equivalent to an LCS.

State of the arts:

- Can be computed in $O(n_1n_2)$ time (in linear space).
- Open: the expected length of an LCS of random sequences of length n ?

53

Affine gap penalty ($g(i) = q + i \cdot r$)

Tables:

$S[i, j]$ is the maximum score of all alignments of $a_1a_2 \dots a_i$ and $b_1b_2 \dots b_j$.

$D[i, j]$ is the maximum score of all alignments of $a_1a_2 \dots a_i$ and $b_1b_2 \dots b_j$ that end with a deletion gap (in B).

$I[i, j]$ is the maximum score of all alignments of $a_1a_2 \dots a_i$ and $b_1b_2 \dots b_j$ that end with an insertion gap (in A).

Recurrences:

$$S[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0 \\ D[i, 0] & \text{if } i > 0 \text{ and } j = 0 \\ I[0, j] & \text{if } i = 0 \text{ and } j > 0 \\ \max \begin{cases} S[i-1, j-1] + \sigma(a_i, b_j) \\ D[i, j] \\ I[i, j] \end{cases} & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

$$D[i, j] = \begin{cases} S[0, j] - q & \text{if } i = 0 \text{ and } j \geq 0 \\ D[i-1, 0] - r & \text{if } i > 0 \text{ and } j = 0 \\ \max \begin{cases} D[i-1, j] - r \\ S[i-1, j] - q - r \end{cases} & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

$$I[i, j] = \begin{cases} S[i, 0] - q & \text{if } i \geq 0 \text{ and } j = 0 \\ I[0, j-1] - r & \text{if } i = 0 \text{ and } j > 0 \\ \max \begin{cases} I[i, j-1] - r \\ S[i, j-1] - q - r \end{cases} & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

Linear-Space Algorithm:

An exercise.

54

Local sequence alignment:

Find one substring of S_1 and one substring of S_2 to have the maximum (sequence) similarity.

For example:

ATAAGGTTGCTAGCGC
CCOCTTCTATTGATA

ATAAGGTTGCTAGCGC TTGCTGA ATA
CCOCTTCTATTGATA TTCCT A ATA

- Local sequence alignment can be done in $O(n_1n_2)$ time using Dynamic Programming as well.

Key observation: allowing re-start

[Smith & Waterman, 1981]

- k -best *non-intersecting* local alignments can be found in $O(n_1n_2 + kn_2)$ time.
[Waterman & Eggert, 1987]

- The space can be reduced to $O(n_2)$ as well.

[Huang & Miller, 1991; Chao & Miller, 1994]

55

The maximum likelihood approach:

- An alignment describes a scenario of mutations.

A T G C T C G C C A T A C A
A C C T C G T C A T G C C A

- Scores are the negative log of probabilities.
- Summing up scores is equivalent to multiplying probabilities of mutations.
- The cost of an alignment is the probability of a particular path of mutations.

- The *maximum likelihood* approach is to consider **all** possible paths of mutations.

– Let $L[i, j] = \Pr(a_1a_2 \dots a_i \rightarrow b_1b_2 \dots b_j)$.

$$L[i, j] = \sum \begin{cases} \Pr_{\text{insert}} \cdot L[i-1, j] \\ \Pr_{\text{delete}} \cdot L[i, j-1] \\ \Pr(a_i \rightarrow b_j) \cdot L[i-1, j-1] \end{cases}$$

– We need find parameters \Pr_{insert} , \Pr_{delete} , $\Pr(a_i \rightarrow b_j)$ to maximize the $L[n_1, n_2] = \Pr(S_1 \rightarrow S_2)$.

– This is a non-linear programming problem and heuristic methods like Simplex and E(nergy)M(inimization) are often used.

56

Sequences annotations and comparison of annotated sequences

1. Functional information

TBS SerSerAlaSer SerLeuAsn
 ***** ##### #####
 cgccctataaaaccagc...agctctgctagcacctgagcctcaat

2. Structural information

alpha beta coil
 ***** ##### %%%/%%/%%/%%/%%
 VLSPADKTNVKAAGWKVGAHAGEYGAEALERMFSLFPTTKTYFPHF...

3. Physical contact/bond



4. And many more.

57

Alignment of DNA Sequences with both Coding and Non-Coding Regions

1. A *Genomic sequence* is a DNA sequence consisting of both coding and noncoding regions.

TBS SerSerAlaSer SerLeuAsn
 ***** ##### #####
 cgccctataaaaccagc...agctctgctagcacctgagcctcaat

How do we compare genomic sequences?

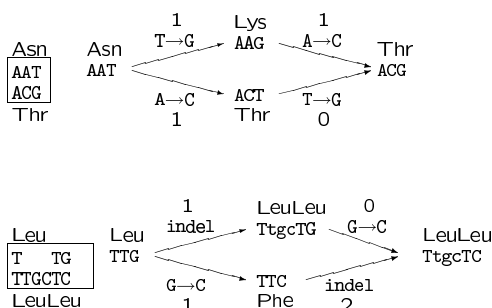
2. The sequences are usually broken up into segments, which are compared separately.
3. For coding regions, classical methods align either DNA sequences based on nucleotide evolution or protein sequences based on amino acid evolution.
 Since amino acids evolve slower than nucleotides, alignments of protein sequences are usually more reliable.
 But nucleotide changes are also important!
4. Can we align coding DNA to also reflect the evolution of its protein?

58

Combined DNA and protein sequence alignment:

- Amino acids are coded by triplets of nucleotides (codon); thus actual mutations happen at the DNA level.
- Alignments of protein sequences are reliable (amino acid evolution is slower than nucleotide evolution).
- Nucleotide changes are also important.

How do we align (coding) DNA sequences to reflect changes at both DNA **and** protein levels?



Alignment cost depends on order of events!

59

The notion of codon alignment [Hein, 1994]:

Align DNA to reflect nucleotide changes; but also consider the cost of *induced* amino acid changes.

Alignment Model:

- $c_d(a, b)$: cost of substituting base b for base a .
- $c_p(e_1e_2e_3, f_1f_2f_3)$: cost of substituting the amino acid coded by $f_1f_2f_3$ for the amino acid coded by $e_1e_2e_3$.
- $g_d(i)$: cost of inserting (or deleting) a block of i nucleotides.
 We consider only indels of lengths divisible by 3, i.e. no frame shifts.
- $g_p(i)$: cost of inserting (or deleting) a block of i amino acids.
- Let $g(i) = g_d(3i) + g_p(i)$. (affine gap: $g(i) = g_{open} + i \cdot g_{ext}$)

An alignment can be partitioned into codon alignments at codon boundaries of both sequences.

The codon alignments are independent of each other.

Operation Costs:

Mutation

For $e_1e_2e_3 \rightarrow f_1e_2e_3$,
 $cost_{mut} = c_d(e_1, f_1) + c_p(e_1e_2e_3, f_1e_2e_3)$.

Insertion

For $e_1e_2e_3 \rightarrow e_1e_2e_3f_1...f_{3i}$,
 $cost_{ins} = g(i)$.

For $e_1e_2e_3 \rightarrow e_1f_1...f_{3i}e_2e_3$,
 $cost_{ins} = g(i) + \min \left\{ \begin{array}{l} c_p(e_1e_2e_3, e_1f_1f_2), \\ c_p(e_1e_2e_3, f_{3i}e_2e_3). \end{array} \right.$

Deletion

For $e_1e_2e_3...e_{3i+1}e_{3i+2}e_{3i+3} \rightarrow e_1e_2e_3$,
 $cost_{del} = g(i)$.

For $e_1e_2e_3...e_{3i+1}e_{3i+2}e_{3i+3} \rightarrow e_1e_{3i+2}e_{3i+3}$,
 $cost_{del} = g(i) + \min \left\{ \begin{array}{l} c_p(e_1e_2e_3, e_1e_{3i+2}e_{3i+3}), \\ c_p(e_{3i+1}e_{3i+2}e_{3i+3}, e_1e_{3i+2}e_{3i+3}). \end{array} \right.$

60

Codon alignment:

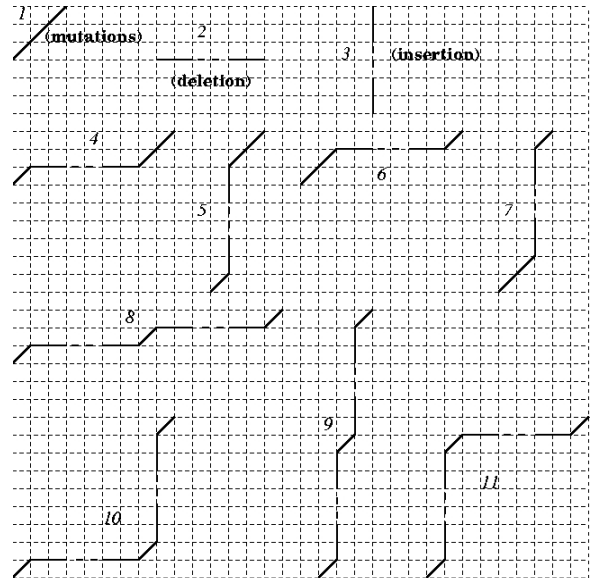
- An alignment can be partitioned into codon alignments at codon boundaries of both sequences.
- The codon alignments are independent of each other.
- There are 11 types of codon alignment.
- Each codon alignment involves at most 5 operations and thus the optimal can be computed easily.

Summary:

1. An $O(n_1^2 n_2^2)$ algorithm [Hein, 1994]
2. An $O(n_1 n_2)$ heuristic algorithm [Hein-Støvlbæk, 1994]
3. An $O(n_1 n_2)$ time algorithm assuming affine gaps [Hua-Jiang, 1987], which requires a table of size roughly $16644 n_1 n_2$.
4. An $O(n_1 n_2)$ time algorithm assuming affine gaps [Pedersen-Lyngsø-Hein, 1998], which requires a table of size roughly $15476 n_1 n_2$ (may be reduced to $800 n_1 n_2$).
5. An $O(n_1 n_2)$ time algorithm assuming affine gaps for a slightly simplified model (context-free), requiring a table of size only $292 n_1 n_2$ [Hua-Jiang-Wu, 1998].

61

The 11 types of codon alignment:



62

A practical consideration:

Genomic Database Search. Given a query sequence $Q = a_1 a_2 \dots a_n$, find all sequences in the database that are similar to Q .

A database may contain millions of sequences, totaling in billions of bases.

The quadratic time DP algorithm is **NOT** fast enough!

Ideas in BLAST [Altschul et al., 1990]

- Screen out all sequences which don't share a common substring of length w with Q .
- Often $w = 12$ for DNA and $w = 4$ for protein.
- Consider $n - w + 1$ substrings $a_i a_{i+1} \dots a_{i+w-1}$, for $i = 1, 2, \dots, n - w + 1$.
- How to find the exact occurrences of these substrings is the topic of *exact string matching* (later).
- It is an approximate searching algorithm.

Ideas in PatternHunter [Li et al., 2002]

- Looking for appearances of length m substrings with at least w matches.

63

BLAST:

- Generalized from local alignment
- *Maximal segment pair* (MSP)
 - highest scoring pair of identical length segments chosen from 2 sequences
 - Generalize to non-identical (score cutoff S)
- *Word pair*
 - a segment pair of fixed length w (score cutoff T) — 12 for DNA, 4 for protein
 - Extending word pair to a possible MSP with score $\geq S$
- MSP vs. WP — $T \downarrow$, chance of MSP containing a WP \uparrow
 - number of WP hits \uparrow
 - running time \uparrow
- 3 algorithmic steps in BLAST:
 1. compiling a list of high-scoring words (two ways: _____, _____)
 2. scanning the database for hits (two ways: _____, _____)
 3. extending hits (heuristics: time vs. accuracy tradeoff)
- Implementation details:
 - database compression
 - removing repetitive words
 - allowing gaps (in the extension step)

64

PsiBLAST:

- In the original BLAST, the extension step accounts more than 90% of execution time.
Try to reduce the number of extensions performed
- Observation: length of HSP is much larger than word-length
Therefore, multiple hits are very possible (same diagonal)

Idea: Do the extension only when

There are two hits at distance within A (pre-specified).
Some commonly used: $w = 3 \sim 4$, $T = 11 \sim 13$, $A = 37 \sim 40$

- Implementation details:
 - hit generation — use of position-specific score matrix
 - hit extension (E -value, affine gap penalty, etc.)
 - allowing gaps (only when the score is high enough)

65

PatternHunter:

- A dilemma — increasing seed size decreases sensitivity while decreasing seed size slows down computation
— large seeds lose distant homology while small seeds creates too many random hits (which slow down the computation)
- *Expect less to get more* — use of nonconsecutive w letters as seeds
The expected number of hits of a weight W , length w model within a length m region of similarity p ($0 \leq p \leq 1$), is $(m - w + 1)p^W$.
- For $W = 11$, the most sensitive model:
111010010100110111 (sensitivity 0.467122 for certain p)
- Implementation details:
 - hit generation
 - hit extension
 - allowing gaps (in the extension step)

66

Notes:

- In any of the above homology search model, we look for either **exact** or **approximate** occurrences of some word/seed/substring.
- Gaps are NOT allowed in the occurrences
- This is the *Exact/Approximate String Matching* problem.

Subject of some future lectures ...

67

Lecture 6: Multiple Sequence Alignment

Agenda:

- Straightforward extension from pairwise algorithm
- Hardness results review
- Approximation algorithm concepts, review
- Approximate MSA

Reading:

- Ref[JXZ, 2002]: Pages 71 – 110
- Ref[Gusfield, 1997]: Pages 332 – 369
- Ref[CLRS, 2001]: Pages 966 – 1054

68

Identification of the Cystic Fibrosis Gene:

Cloning and Characterization of Complementary DNA:

- Official Gene Symbol: CFTR

- Name of Gene Product:

cystic fibrosis transmembrane conductance regulator

- **LOCUS:**

An alignment achieving the minimum cost (w.r.t. σ):

7q31.2

The CFTR gene is found in region q31-q32 on the long (q) arm of human chromosome 7.

- Size:

1. Identification of highly conserved regions in homologous sequences.

E.g. the case of Cystic Fibrosis (Science, 1989).

- ## 2. Reconstruction of evolutionary trees.

A typical reconstruction algorithm begins with multiple sequence alignment.

- ### 3. Applications in computer science.

E.g. syntactical analysis of HTML tables and analysis of electronic chain mail.

- **Protein Function:**

The normal CFTR protein is a chloride channel protein found in membranes of cells that line passageways of the lungs, pancreas, colon, and genitourinary tract. CFTR is also involved in the regulation of other transport pathways. Defective versions of this protein, caused by CFTR gene mutations, cause cystic fibrosis (CF) and congenital bilateral aplasia of the vas deferens (CBAVD).

Cloning and Characterization of Complementary DNA:

A multiple alignment between CFTR genes and other membrane-bound proteins.

- mRNA sequence (NM_000492 to access NCBI)
- protein sequence (NP_000483 to access NCBI)

- Common disease:

About 70% of mutations observed in CF patients result from deletion of three base pairs in CFTR's nucleotide sequence. This deletion causes loss of the amino acid phenylalanine located at position 508 in the protein (this mutation is referred to as deltaF508 CFTR).

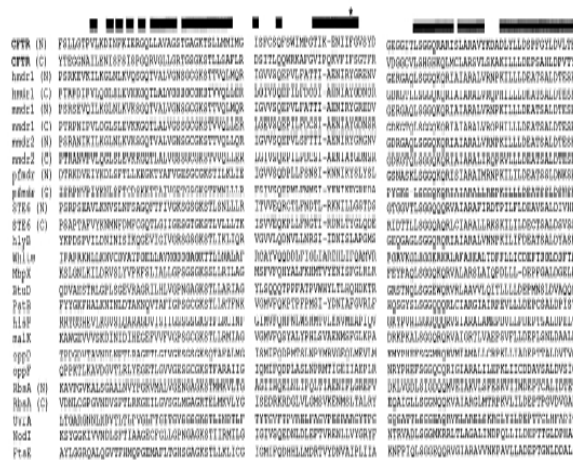
Homozygous patients have severe symptoms of cystic fibrosis (e.g. breathing difficulty).

- Identification of the cystic fibrosis gene:

Done by Riordan *et al.* from Hospital for Sick Children (Toronto)

"Identification of the Cystic Fibrosis Gene:
Cloning and Characterization of Complementary DNA."
Science, volume 245: 1066-73 (1989 September).

- Using multiple sequence alignment to detect the mutations at the DNA-level.

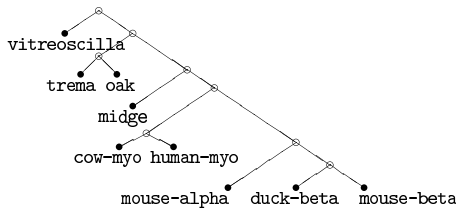


Typical reconstruction of evolutionary tree:

1. Input: a set of molecular sequences. *E.g.* some globin genes of human, mouse, etc.
2. Compute a multiple alignment of the sequences.

	121	131	141
trema	TCCT--CAAG	A--TA---TT	TG--AGATTG
mouse-alpha	AT-A--TGGA	GCTGAAGCCC	TGGAAAG---
midge	CT-T--CAAG	GCTGA---TC	CATCAAT---
cow-myo	GG-CCATGGG	CAGGAGGTCC	TCATCAG---
mouse-beta	AG-T--TGGA	GGAGAACTC	TGGGAAG---
human-myo	GG-CCATGGG	CAGGAAGTCC	TCATCAG---
vitreoscilla	TTAT--AAAA	ACTTG---TT	TGCCAAA---
duck-beta	CT-G--TGGA	GCTGAGGCC	TGGCCAG---
oak	TCTT--AAAG	A--TA---TT	TG--AGATCG

3. Analyze each column to infer pairwise distance.
4. Construct an evolutionary tree.



73

Models of MSA:

- An alignment:

```

S1:  TACCCGGGCC--CCTTTGAGCA
S2:  T-CCT-GGGCCAACTT-AAGCG
S3:  CACCC-GGGCCAGCTTTAAGCG
S4:  TACCCCGA-CCAAGTTT-AA-CT

```

S'_i denotes the original sequence S_i with spaces inserted.

- Each column j has a cost: $c(S'_1[j], S'_2[j], \dots, S'_k[j])$.
- Cost of alignment is the sum of costs of columns.
- The goal is to minimize the cost.
- Example cost functions:

- Longest Common Subsequence (LCS):

$$c(a_1, a_2, \dots, a_k) = \begin{cases} -1 & \text{if } a_1 = a_2 = \dots = a_k \\ 0 & \text{otherwise} \end{cases}$$

- Maximum Weight Trace (MWT):

$$c(a_1, a_2, \dots, a_k) = - \sum_{a \in \Sigma} \left(\sum_{i=1}^k I(a, a_i) \right)^2,$$

where

$$I(a, a_i) = \begin{cases} 1 & \text{if } a = a_i \\ 0 & \text{otherwise} \end{cases}$$

74

Models of MSA (cont'd):

- Models of cost/scoring schemes:
Given a pairwise cost scheme $s(a, b)$,

- 1. Sum-of-Pairs (SP) cost:

$$c(a_1, a_2, \dots, a_k) = \sum_{i \neq j} s(a_i, a_j)$$

- 2. Consensus cost:

$$c(a_1, a_2, \dots, a_k) = \min_a \sum_i s(a, a_i)$$

- 3. Tree cost: w.r.t. a given evolutionary tree T ,

$$c(a_1, a_2, \dots, a_k) = \min_{\text{internal node assignment}} \sum_{(a,b) \in T} s(a, b)$$

TACGGGCC	TTG	-----+	
			o-----+
T	CAGGCCC	A	-----+
CA	GGG	CCTTA	-----+
TACAA	CCTTTG		-----+
			o-----+

75

The evaluation of tree-cost of a column:

		*	
TACGGGCC	TTG	G-----+	
			b_1---+
T	CAGGCCC	A	-----+
CA	GGG	CCTTA	-----+
TACAA	CCTTTG	G-----+	
			b_2---+

tree-cost of column *:

$$\begin{aligned}
 c(G, A, A, G) &= \min_{b_1, b_2, b_3} s(G, b_1) + s(A, b_1) + s(A, b_2) + s(G, b_2) \\
 &\quad + s(b_1, b_3) + s(b_2, b_3) \\
 &= s(G, G) + s(A, G) + s(A, G) + s(G, G) \\
 &\quad + s(G, G) + s(G, G) \\
 &= 2 \\
 &\text{(assume } s(a, b) = 0 \text{ if } a = b, \text{ or } 1 \text{ otherwise)}
 \end{aligned}$$

This can be done by dynamic programming in linear time. (also called the *small parsimony problem*)

Usually assume cost $s(a, b)$ is a *distance metric*.

76

An exact algorithm — Dynamic Programming:

- Idea:
Extending the Dynamic Programming algorithm for pairwise sequence alignment to k dimensions.
- Table entry computation (recurrence):
 $d(i_1, i_2, \dots, i_k)$ denotes the cost of an optimal alignment for k prefixes $S_1[1..i_1], S_2[1..i_2], \dots, S_k[1..i_k]$, where $0 \leq i_j \leq n_j$.
$$d(i_1, i_2, \dots, i_k) = \min \{ d(i'_1, i'_2, \dots, i'_k) + s(i'_1 + 1, i'_2 + 1, \dots, i'_k + 1) \}$$
where $i'_j = i_j$ or $i_j - 1$ and $s(i'_1 + 1, i'_2 + 1, \dots, i'_k + 1)$ is the cost of the last column in the alignment containing k letters/spaces, one from each given prefix.
The minimization is taken over $2^k - 1$ entries (recall the case $k = 2$).
- Boundary entry computation:
An entry $d(i_1, i_2, \dots, i_k)$ is boundary if at least one of its indices i_j is 0.
- Running time and space requirement:
If $s(i'_1 + 1, i'_2 + 1, \dots, i'_k + 1)$ is known,
 - There are $(n_1 + 1) \times (n_2 + 1) \times \dots \times (n_k + 1)$ entries.
 - Each needs to scan $2^k - 1$ values.
 - Space: $O(n^k)$
 - Time: $O(2^k n^k)$

77

Dynamic Programming recurrences:

- SP alignment:
Boundaries:
$$d(0, i_2, \dots, i_k) = d(i_2, \dots, i_k) + \sum_{i=2}^k \sum_{j=1}^{i_j} s(S_i[j], -),$$
Column cost:
$$c(a_1, a_2, \dots, a_k) = \sum_{i \neq j} s(a_i, a_j).$$
General recurrence:
$$d(i_1, i_2, \dots, i_k) = \min_{i_j - 1 \leq i'_j \leq i_j, \sum_{j=1}^k i'_j < \sum_{j=1}^k i_j} \{ d(i'_1, i'_2, \dots, i'_k) + c(S_1[i'_1 + 1], S_2[i'_2 + 1], \dots, S_k[i'_k + 1]) \},$$
where $S_j[i_j + 1] = -$.
- Consensus alignment:
An exercise !
- Tree alignment:
An exercise !

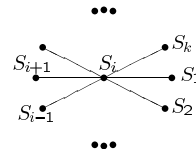
78

Approximate SP alignment:

- Sum-of-all-Pairs: $c(S_1, S_2, \dots, S_k) = \sum_{i \neq j} s(S_i, S_j)$.
- The exact algorithm running time $O(n^k)$,
which is **NOT** polynomial when k is also a variable !!!
- In general, the problem is NP-hard — no polynomial time algorithm exists unless $P = NP$
- This is **NOT** the end of story.
We still want to solve the problem to some extent
 - sacrifice the accuracy
 - approximately good (performance guaranteed) alignments in a short (polynomial) time
- Observations:
 - Fix index i , computing $c(S_i) = \sum_{j \neq i} s(S_i, S_j)$ is easy.
 - Computing $\min_i c(S_i)$ is then easy too.
 - Using the alignment \mathcal{A} produced in this way as the approximate SP alignment.

79

Analysis of the approximation algorithm:

- Running time:
 $O(k \times k \times n^2)$
- 
- Performance analysis:
 \mathcal{A} — alignment computed
 \mathcal{A}^* — an optimal alignment
 - For fixed i , in \mathcal{A}^* , $\sum_{j \neq i} s(S_i, S_j) \geq c(S_i)$.
 - Thus $c(\mathcal{A}^*) \geq \frac{1}{2} \sum_i c(S_i) \geq \frac{k}{2} \cdot \min_i c(S_i)$.
Or equivalently, $\min_i c(S_i) \leq \frac{2}{k} c(\mathcal{A}^*)$.
 - Therefore,

$$\begin{aligned} c(\mathcal{A}) &= \sum_{j \neq i} s(S_j, S_i) \\ &\leq (k-1) \sum_{j \neq i} s(S_i, S_j) \text{ (by triangle inequality)} \\ &= (k-1) c(S_i) \\ &\leq (2 - \frac{2}{k}) c(\mathcal{A}^*). \end{aligned}$$
 - $c(\mathcal{A}) \leq (2 - \frac{2}{k}) c(\mathcal{A}^*)$
 - Simulations showing that the algorithm performs much better in practice:
 - On 19 random sequences, $c(\mathcal{A}) \leq 1.02 c(\mathcal{A}^*)$.
 - On homologous sequences, $c(\mathcal{A}) \leq 1.16 c(\mathcal{A}^*)$.

80

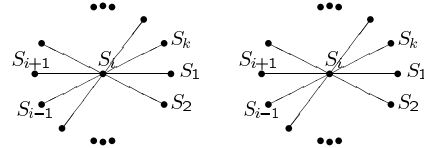
Approximation algorithm (review):

- Runs in polynomial time (in the input instance size)
- Produces a solution within certain range of the optimum
- For a minimization (maximization) problem, an α -approximation algorithm produces a solution, for any instance, with cost less than or equal to α ($\frac{1}{\alpha}$, respectively) times the cost of an optimal solution.
 α is called the performance ratio of the approximation.
- The above is a 2-approximation algorithm for the MSA with SP cost.

81

Approximate SP alignment (2):

- The above approximation algorithm tries to find the *best* sequence to be used as a *center* sequence.
It has performance ratio $2 - \frac{2}{k}$.
- The center sequence has the property that it has the minimum sum of pairwise distances to other sequences.
The graph (tree) connecting the center to all other sequences is called a **2-star**, and the algorithm the **2-star** algorithm.
- Generalizing to **3-star** (disjoint triangles glued at the center):
 - Now each “edge” from the center involves 3 sequences. Need a partition of the other sequences into pairs, which should be *optimal* regarding the chosen center sequence.



- For fixed index i , an optimal partition of the other sequences and the associated optimal alignment can be computed by *maximum weighted matching*.
Weight function is defined by:

$$w(S_j, S_\ell) = \min_A (d_A(S_j, S_\ell) + (k-2) \cdot d_A(S_i, S_j) + (k-2) \cdot d_A(S_i, S_\ell))$$

82

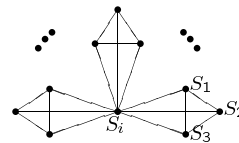
Approximate SP alignment (2) (cont'd):

- Maximum weighted matching can be computed efficiently
- An optimal center sequence S_i and its associated optimal alignment \mathcal{A} can thus be computed efficiently.
- (1992) The 3-star algorithm runs in time $O(n^3 k^3 + k^5)$ and achieves performance ratio $2 - \frac{3}{k}$.
- Analysis of 3-star algorithm:
 - Running time
An exercise
 - Performance
An exercise

83

Approximate SP Alignment (3):

- Consider ℓ -stars:



Thm. (1992) The multiple sequence alignment induced by an optimal ℓ -star costs at most $(2 - \frac{\ell}{k})c(\mathcal{A}^*)$.

Problem: Can't find an optimal ℓ -star when $\ell > 3$, because 3 dimensional matching is already NP-hard.

Solution: **balanced set** of ℓ -stars.

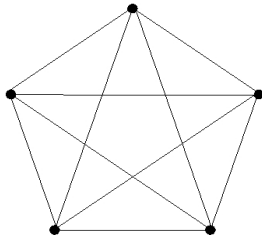
Thm. (1994) The best ℓ -star in a balanced set also induces an alignment with cost at most $(2 - \frac{\ell}{k})c(\mathcal{A}^*)$.

- Running time: $O(k \cdot n^\ell \cdot |B|) = O(k^3 \cdot n^\ell)$.

84

Balanced set of ℓ -stars:

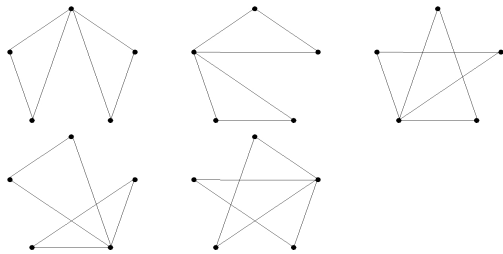
- Definition:
The edge set connecting to the ℓ -starts covers each edge of the complete graph the same number of times.
- Example:



K_5 ($k = 5$):

There are in total 30 possible 3-stars ($\ell = 3$).

- A balanced set of 3-stars — covers each edge 3 times:



85

Approximate consensus alignment:

- Consensus sequence:
 - Steiner consensus sequence S — definition without MSA
 - * S need not be from the set of given set of sequences
 - * It is generally hard to compute S
 - * Assuming the pairwise alignment cost scheme satisfies triangle inequality,
The 2-star sequence is an approximation, with performance ratio $(2 - \frac{2}{k})$.
 - Definition based on MSA:
For each column in the MSA, which letter minimizes the column cost?
Concatenating the consensus letters into a sequence — consensus sequence
- Two definitions equivalent to each other
- NP-hard
- 2-approximation algorithm
Can you improve it ?

86

Approximate tree alignment:

- Tree alignment is NP-hard, even when the cost scheme satisfies the triangle inequality.
- Generalized Tree alignment (when the tree is not given ahead of time) is MAX SNP-hard.
Which means, no PTAS !!! unless $P = NP$.

87

Agenda:

- PTAS for Tree Alignment
- Special cases of MSA
- PTAS for some special cases

Reading:

- Ref[JXZ, 2002]: Pages 71 – 110
- M. Li *et al.* *Near optimal multiple alignment within a band in polynomial time.* ACM STOC 2000. 425–434. Portland, Oregon.
- C. Tang *et al.* *Constrained multiple sequence alignment tool development and its application to RNase family alignment.* IEEE CSB 2002. 127–137. Stanford, California.

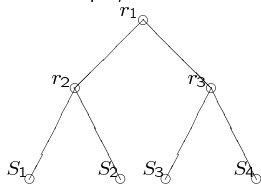
88

Approximate tree alignment:

- Tree alignment is NP-hard, even when the cost scheme satisfies the triangle inequality.
- Generalized Tree alignment (when the tree is not given ahead of time) is MAX SNP-hard.
Which means, no PTAS !!! unless $P = NP$.
- Reformulating tree alignment:

Sequence reconstruction: Given a rooted evolutionary tree T with leaves labeled by sequences, construct a sequence for each interior node to minimize the cost of the tree.

— An example,



Sequences and cost scheme:

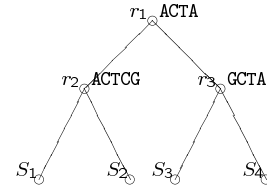
$S_1 =$	ACTG
$S_2 =$	ATCG
$S_3 =$	GCCA
$S_4 =$	GTTA

	A	C	G	T	-
A	0	1	1	1	1
C	1	0	1	1	1
G	1	1	0	1	1
T	1	1	1	0	1
-	1	1	1	1	0

89

Approximate tree alignment (cont'd):

- Cost of edge (x, y) : the optimal pairwise alignment cost of the sequence labels at x and y
String Edit, computable in $O(n^2)$ time
- Cost of (fully) labeled tree: total cost of edges
- This is a variant of *fix-topology* Steiner tree problem.
- Equivalence of the formulations:



ACTG ATCG GCCA GTTA

The induced MSA:

r_1 :	ACT-G	S_1 :	ACT-G
r_2 :	ACTCG	S_2 :	A-TCG
r_3 :	GCT-A		
r_1 :	ACT-A		
r_2 :	ACTCG		
S_3 :	GCC-A	S_3 :	GCC-A
r_3 :	GCT-A		
S_4 :	GTT-A	S_4 :	GTT-A

- Cost of optimally labeled tree = tree cost of optimal alignment

90

Tree alignment vs. Steiner tree:

- Steiner Minimal Tree (SMT):
Given a set S of terminals in some space, find the shortest network connecting S .
- For our generalized tree alignment problem, the space is DNA sequences with (weighted) edit distance.

Corollary. The generalized tree alignment problem can be approximated with ratio ~ 1.5 .

- Tree alignment is SMT with a *fixed topology* !

Corollary. SMT with a fixed topology in any metric has a PTAS.

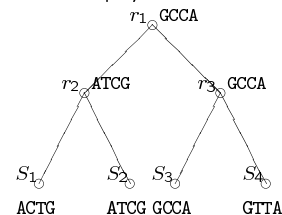
- PTAS: polynomial time approximation scheme — approximate arbitrarily close to the optimum

91

Approximate tree alignment (2):

- Lifted tree:
A fully labeled tree in which every parent sequence equals some child sequence.

- For example,



Theorem. For some lifted tree T^ℓ , $\alpha(T^\ell) \leq 2 \times \alpha(T^*)$.
 T^* is an optimal labeled tree.

Proof. By averaging over all lifted trees.

Theorem. We can compute an optimal T^ℓ in $O(k^3 + k^2n^2)$ time.

Theorem. Tree alignment has a PTAS that achieves ratio $1 + \frac{2}{1+t}$ in $O(kdn^{2^{t-1}+1})$ time, where d is the depth of the tree.

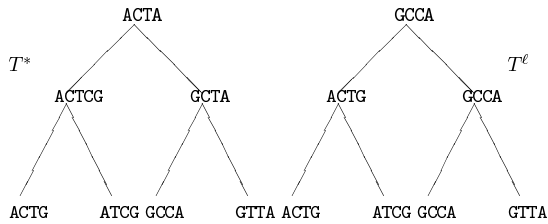
t	2	3	4
ratio	1.67	1.50	1.40
running time	$O(kdn^3)$	$O(kdn^5)$	$O(kdn^9)$

92

A sketch of analysis of 2-approximation:

- T^* — an optimal labeled tree (not necessarily lifted)
For each $v \in T^*$, $\ell(v)$ — the descendant leaf closest to v
WARNING: some constraints on choosing $\ell(v)$

- T^ℓ — obtained by lifting sequence of $\ell(v)$ to v



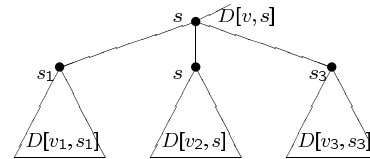
- Observation: $c(T^\ell)$ is at most the total distance between every pair of adjacent leaves in T^* .
With **suitable flipping** of the subtrees !!!

Theorem. $c(T^\ell) \leq 2 \times c(T^*)$.

93

Computing an optimal lifted tree:

- v — internal node
- T_v — subtree rooted at v
- $S(v)$ — subset of sequences corresponding to leaves in T_v
- Define for each v and $s \in S(v)$, $D[v, s]$ is the cost of an optimal lifted tree for T_v with v being labeled by s .



- Recurrence:
For each $i \neq 2$, find an $s_i \in S(v_i)$ to minimize $D[v_i, s_i] + d(s, s_i)$.

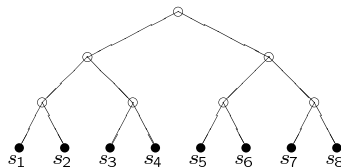
$$D[v, s] = D[v_2, s] + \sum_{i \neq 2} (D[v_i, s_i] + d(s, s_i))$$

- Running time is $O(k^2 n^2 + k^3)$
Can be improved to $O(kn \times \text{depth})$ by uniform lifting (next).

94

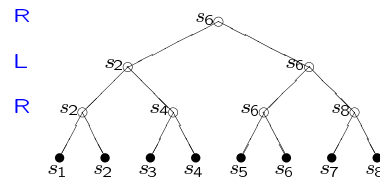
Uniform lifting tree:

- Given a phylogeny, the order of the sequences are fixed
- During the **lifting**, the lifting decisions (**L**eft or **R**ight) are identical at each level of the tree



Uniform lifting tree:

- Given a phylogeny, the order of the sequences are fixed
- During the **lifting**, the lifting decisions (**L**eft or **R**ight) are identical at each level of the tree



The uniform lifting choice vector is $V = (R, L, R)$

Theorem. For an average uniformly lifted tree T^{ul} ,

$$c(T^{ul}) \leq 2c(T^*).$$

Prove it.

Theorem. We can compute an optimal uniform T^{ul} in $O(kd + kdn^2)$ time, where d is the depth of T .

Dynamic programming.

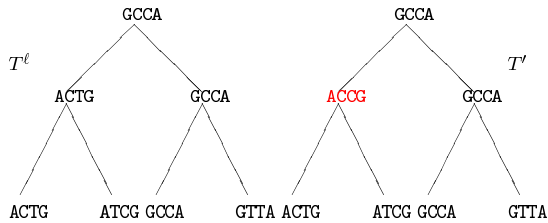
95

96

PTAS for Tree alignment:

Theorem. Tree alignment has a PTAS that achieves ratio $1 + \frac{2}{1+t}$ in $O(kdn^{2^{t-1}+1})$ time.

- Idea: from spanning to Steiner
Keeping the lifted sequences at some nodes while reconstructing *Steiner* sequences at the others
- For example,



97

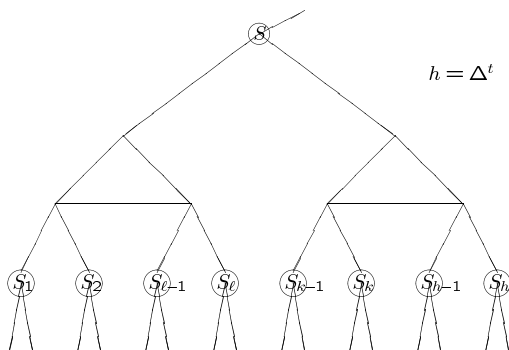
PTAS for Tree alignment (cont'd):

- Key techniques:
 - Careful partitioning strategy
 - Local optimization
 - Dynamic programming
 - Uniform lifting
- Outline:
 - Partition the tree into overlapping *components*, each with r boundary nodes
 - Uniform lifting to boundary nodes
Local optimization on each component
Dynamic programming
 - Consider a set of partitions (by shifting) so that every node has equal chance of being a boundary node
The partitions yield a good approximation on average
 - The the best result of all partitions

98

Extending the 2-approximation to a PTAS:

- General idea: *lifting + local optimization*
Assuming Δ -ary trees
- depth- t component: a subtree with depth t
- Local optimization: optimize a depth- t component with fixed labels at leaves and root



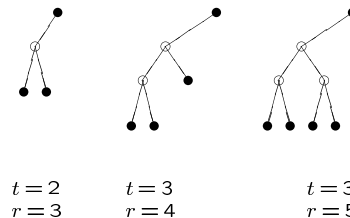
- This requires optimizing Δ depth- $(t-1)$ components with fixed labels at leaves and parent.
- Each takes $M(\Delta, t-1, n) = O(n^{\Delta^{t-1}+1})$ time.

99

Extending the 2-approximation to a PTAS:

- The partition P_i , $0 \leq i < t$ (assuming binary tree)
 - Fix a uniform lifting choice V
 - The top component is a binary tree of height i
 - All nodes on the lifting path of head/leaf of a component are also heads
 - There are $t \times 2^d$ different components, where d is the height of the tree
 - These components put all nodes of the tree on boundaries with *roughly equal frequency*

L -type components:

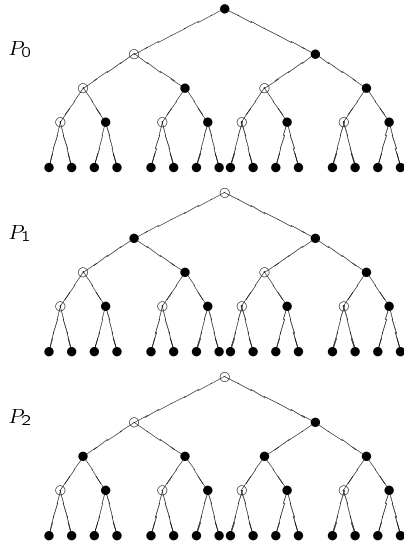


R -type components:

100

Extending the 2-approximation to a PTAS (cont'd):

- A partition example: $t = 3, r = 4, V = (R, R, R)$



101

Extending the 2-approximation to a PTAS (cont'd):

- With respect to each partition P_i , the best depth- t component tree is denoted by T^i
- Compute T^i by optimally lifting leaves to the boundaries and locally optimize the other nodes
For P_i , levels $i, i+t, i+2t, \dots$, form the boundaries
 - For each boundary node v and lifted sequence s ,
 $D[v, s]$ — cost of an optimal depth- t component subtree with root label s
 - $D[v, s]$ can be computed iteratively from bottom to top (dynamic programming)
Reduced to the optimization of tree of depth- $(t-1)$
 - The size of local optimization is $r = \Delta^{t-1} + 1$
 - Running time:
 $O(k\Delta^{t-1+2} \times M(\Delta, t-1, n)) = O(k\Delta^{t-1+2}n\Delta^{t-1+1})$,

Lemma. $\sum_{i=0}^{t-1} c(T^i) \leq (t+3)c(T^*)$.

Corollary. For some i , $c(T^i) \leq (1 + \frac{3}{t})c(T^*)$.

Key observation: Let T be a full binary tree. For each leaf $\ell \in L(T)$, there is a mapping π_ℓ from $I(T)$ to $L(T)$ such that

- $\forall v \in I(T), \pi_\ell(v)$ is a descendant of v
- The paths from nodes v to their images $\pi_\ell(v)$ are disjoint
- \exists unused path from root to ℓ

102

PTAS implementation and experimentation:

- TAAR — Tree Alignment and Reconstruction
- DNAPARS (in PHYLIP)
 - Arrange the sequences (species) in some order
 - Construct a tree for the first 3 sequences
 - For each of the rest sequences do
Add the sequence to the current tree to achieve the minimum cost
(It assumes a given MSA)
 - Assumption on MSA can be removed if the cost is measured using tree alignment

103

MSA in practice:

There are many multiple alignment programs available either commercially or as freewares, such as

CLUSTAL W, DIALIGN, MULTAL, MST, ESEE, ...

It seems all are based on the same idea: *clustering* and *progressive alignment*.

The Generic Algorithm

1. Find the pair of the most similar sequences.
2. Optimally align the pair.
3. Record the alignment as either a profile or a consensus sequence, and substitute it for the pair.
4. Repeat until only one profile/sequence is left.
5. Recover the multiple alignment from the final profile or by aligning each sequence with the final consensus sequence.

Comments

1. Quite efficient.
2. No clear objective function.
3. Some also use the evolutionary tree as the guide tree.

104

CLUSTAL W — king of progressive alignment:

Step 1: Simple pairwise alignments and distance

- Quick approximate or dynamic programming (k -tuple match)
- $\text{score} = 1 - \text{percent identity} - \text{gap}$

Step 2: Compute guide tree using neighbor joining (NJ)

- Branch lengths
- Rooted at "mid-point"
- Weight of sequence = $\sum_{e \in P} \frac{w(e)}{\# \text{ of sequences sharing } e}$

Step 3: Progressive alignment

- traverse the tree in post-order
- align a pair of alignments using dynamic programming (Smith-Waterman, linear-space)
- existing gaps are frozen
- column score = average score between each pair of residues from different alignments (*i.e.* variant of SP-score)
- space results in 0 (worst case)

Agenda:

- Three computation models
- Algorithms & approximations

Reading:

- Ref[JXZ, 2002]: chapter 14.
- T. Jiang *et al.* *A general edit distance between RNA structures*. Journal of Computational Biology. (2002) 9, 371–388.
- G. Lin *et al.* *The longest common subsequence problem for sequences with nested arc annotations*. ICALP 2001. LNCS 2076, pp. 444–455.
- K. Zhang *et al.* *Computing similarity between RNA structures*. CPM 1999. LNCS 1645. pp. 281–293.
- V. Bafna *et al.* *Computing similarity between RNA strings*. CPM 1995. LNCS 937. pp. 1–16.

Lecture 9: RNA Sequence and Structure Alignment

Backgrounds:

- RNA performs a wide range of functions (HIV)
- Presumption: a preserved function corresponds to a preserved structure
- RNA structures:
 - primary — sequence of nucleotides
 - secondary & tertiary — bonded pair of two complementary bases
 - secondary — base pairs
 - tertiary — base pair & spatially close pair of nucleotides weakly bonded
 - one base involves in at most one base pair
 - secondary structure non-crossing / nested
- Representing a bond between a pair by drawing an arc connecting them

arc-annotated sequence

A tRNA and its associated annotated sequence:



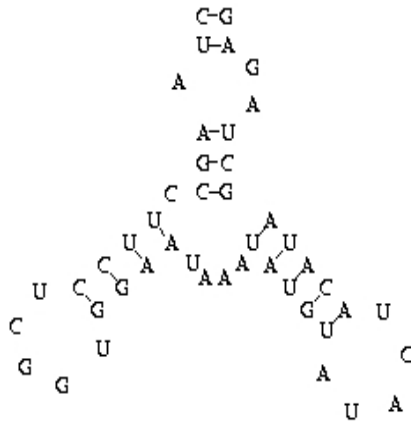
Lecture 9: RNA Sequence and Structure Alignment

Comparison taking sequence & structure information:

- Intended: the measure of similarity becomes more accurate
...
- Possible mutations: looking at the structures
 - nucleotide mutations (replacement, mismatch, substitution, ...)
if that base is involved in a base pair, then the bond could disappear — depending on the other base change
 - single nucleotide indel
if that base is involved in a base pair, then the bond disappears — more costly ?
 - a base pair indel
 - a base pair mutation (the bond maintains, e.g. A-U → C-G — more costly than single base mutation?
 - In short, *mutation operations have to be carefully defined*
...
- Three existing comparison models (sequence + secondary structure):
 - More focus on secondary structure units
Less on base pair mutations
Even less on single nucleotide mutations
 - No secondary structure units
Separate base pair mutations from single base mutations
 - No secondary structure units
Distinguish but not really separate the two sets of mutations

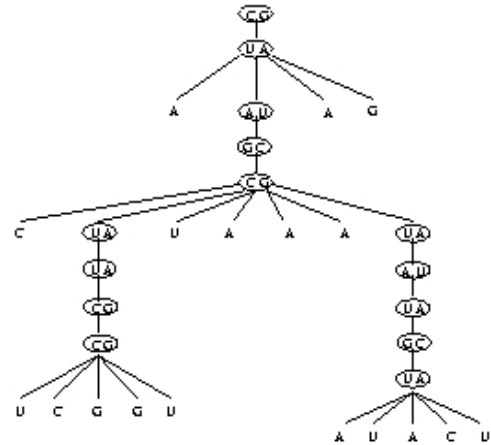
Tree edit distance:

- Using the planar secondary structure itself.



109

Tree (rooted, ordered) representation of the above RNA secondary structure.



Tree edit compares two trees taking into account secondary structure changes, base pair mutations, and single base mutations

110

If a base pair is not matched with another base pair, then delete it:

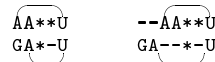


- Secondary structure vs. tertiary structure — solvable
Algorithm is similar to some to be described later ...
- Tertiary structure vs. tertiary structure — NP-hard
Approximations ???

The above (tree edit) model seems a bit too strong. We consider a more relaxed model in next page.

111

If a base pair is not matched with another base pair, then at least one of its bases should be deleted:



- Levels of restrictions on arcs:
 - no sharing of end bases
 - no crossing
 - no nesting
 - no arcs

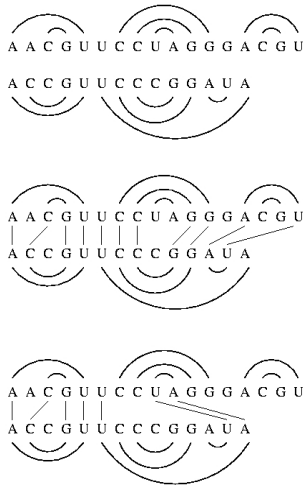
LAPCS: Longest Arc-Preserving Common Subsequence Problem:

Given two arc-annotated sequences, find a longest common subsequence $c_1c_2\dots c_k$ such that (c_i, c_j) is an arc in one sequence if and only if it is an arc in the other.

- unlimited — no restriction,
- crossing — restriction 1,
- nested — restrictions 1 & 2,
- chain — restrictions 1, 2, & 3,
- plain — restriction 4.

112

An example pair of arc-annotated sequences. The first common subsequence ACGUCCGGAU is arc-preserving but the second common subsequence ACGUUUA is not.



113

Best-to-date results for LAPCS(\cdot, \cdot):

	plain	chain	nested	crossing	unlimited
unlimited	NP-hard [Evans, 1999] inapprox. within ratio n^ϵ , for $\epsilon \in (0, \frac{1}{4})$				
crossing	NP-hard [Evans, 1999] MAX SNP-hard 2-approx.				
nested	$O(nm^3)$		NP-hard 2-approx.		
chain	$O(nm)$ [Evans, 1999]				
plain	$O(nm)$				

Special Cases for LAPCS(nested, nested):

- unary
- c -diagonal, c -fragmented

Results:

- still NP-hard; $\frac{4}{3}$ -approximable
- still NP-hard; admit PTAS

114

LAPCS(crossing, plain) is NP-hard:

- General procedure of proof of NP-hardness of problem A :
 - Find some appropriate known NP-hard problem B
 - For every instance I of B , construct an instance J of A
 - Prove that I is a yes-instance iff J is
- For LAPCS(crossing, plain):
 - An appropriate known NP-hard problem is Maximum Independent Set of vertices in Cubic graphs (MIS-cubic):
Given a simple cubic graph $G = (V, E)$, where $|V| = n$, and an integer k , is there a subset $S \subset V$ such that
 1. no two vertices in S are adjacent in G ,
 2. $|S| \geq k$?
 - Reduction from MIS-cubic:
 - * $S_1 = (\text{AAAACCGGG})^n$
 - * $S_2 = (\text{AAAAGGCC})^n$
 - * an edge between v_i and v_j corresponds to an arc connecting a G from the i th segment and a G from the j th segment
 - * making sure that every G is used exactly once
 - * set $k' = k + 6n$
 - Prove that G has an independent set of size at least k iff the constructed LAPCS(crossing, plain) has an APCS of length at least k' .

115

2-Approximation for LAPCS(crossing, crossing):

Given (S_1, P_1) and (S_2, P_2) ,

1. Compute a classical LCS for sequences S_1 and S_2 , denoted S ;
2. If there is exactly one arc that connects two bases in S , connect these two bases by an edge — this constructs a graph G on bases in S ;
3. Delete the minimum number of vertices (bases in S) from G to make the remaining graph trivial
— equivalently, find an MIS for G ;
4. Take the vertices remained as the approximate subsequence T .

5. Properties:

- G has the maximum degree (at most) 2.
- An MIS of G can be computed easily.
- The size of MIS is at least half the number of vertices.
- Therefore, $|T| \geq \frac{1}{2}|S|$.

6. Conclusion:

$$|T| \geq \frac{1}{2}|S| \geq \frac{1}{2}|S^*|$$

116

LAPCS(crossing, plain) is MAX-SNP-hard:

- General procedure of proof of MAX SNP-hardness of problem A :
 - Find some appropriate known MAX SNP-hard problem B
 - For every instance I of B , construct an instance J of A
 - Show $OPT(J) \leq a \times OPT(I)$
 - Prove that an approximate solution S to I implies an approximate solution T to J such that
 - * T can be obtained from S in polynomial time
 - * $OPT(I) - |S| \leq b \times (OPT(J) - |T|)$
namely, the error rate guaranteed
- Reduction from MIS for cubic graphs:
 - $S_1 = (AAAACCGGG)^n$
 - $S_2 = (AAAAGGCC)^n$
 - an edge between v_i and v_j corresponds to an arc connecting a G from the i th segment and a G from the j th segment
 - making sure that every G is used exactly once
- Prove that v_i is in the independent set if and only if AAAAGGG from the i th segment is in the common subsequence (otherwise AAAACC).
- $LAPCS(\text{crossing, plain}) = MIS(G) + 6n \leq 25 \cdot MIS(G)$.
- $|k - MIS(G)| \leq |k' - LAPCS(\text{crossing, plain})|$.

117

Dynamic Programming for LAPCS(nested, plain):

- The key ideas:
 - Notice that at most one base for every base pair can be in (any) LAPCS
 - Ordinary DP reduces the computation to neighboring entries, which is not true anymore
 - Consider the two bases at the same time. How ???
 - * the nested arc structure enables us to do this ...
 - * for every pair of indices (i, i') , such that no arc can have one base inside, while the other outside interval $[i, i']$:
compute how $(S_1[i, i'], P_1[i, i'])$ is aligned with $S_2[j, j']$

118

Dynamic Programming for LAPCS(nested, plain) (cont'd):

Step 1: Given (S_1, P_1) and (S_2, \emptyset) , $DP(i, i'; j, j')$ records the length of LAPCS of $S_1[i, i']$ and $S_2[j, j']$, where no arc has exactly one base inside $[i, i']$.

Phase 1: if i' is free,

$$DP(i, i'; j, j') = \max \begin{cases} DP(i, i' - 1; j, j' - 1) + \chi(S_1[i'], S_2[j']), \\ DP(i, i' - 1; j, j'), \\ DP(i, i'; j, j' - 1); \end{cases}$$

otherwise,

$$DP(i, i'; j, j') = \max_{j \leq j'' \leq j'} \{ DP(i, u(i')_l - 1; j, j_1'') + DP(u(i')_l, i'; j'', j') \}.$$

Phase 2: $(i_1, j_1) \in P_1$,

$$DP(i_1, j_1; j, j') = \max \begin{cases} DP(i_1 + 1, j_1 - 1; j + 1, j') + \chi(S_1[i_1], S_2[j]), \\ DP(i_1 + 1, j_1 - 1; j, j' - 1) + \chi(S_1[i_1], S_2[j']), \\ DP(i_1 + 1, j_1 - 1; j, j'), \\ DP(i_1, j_1; j, j' - 1), \\ DP(i_1, j_1; j + 1, j'); \end{cases}$$

Step 2: if i' is free,

$$DP(1, i'; j, j') = \max \begin{cases} DP(1, i' - 1; j, j' - 1) + \chi(S_1[i'], S_2[j']), \\ DP(1, i' - 1; j, j'), \\ DP(1, i'; j, j' - 1); \end{cases}$$

otherwise,

$$DP(1, i'; j, j') = \max_{j \leq j'' \leq j'} \{ DP(1, u(i')_l - 1; j, j_1'') + DP(u(i')_l, i'; j'', j') \}.$$

$DP(1, n_1; 1, n_2)$ gives the length of the annotated LAPCS. Simple back-tracing gives the subsequence.

119

Lecture 10: Sequence & Structure Alignment

Agenda:

- More algorithms & approximations for LAPCS
- Models on protein structure comparison

Reading:

- T. Jiang *et al.* A general edit distance between RNA structures. Journal of Computational Biology. (2002) 9, 371–388.
- G. Lin *et al.* The longest common subsequence problem for sequences with nested arc annotations. ICALP 2001. LNCS 2076, pp. 444–455.
- D. Goldman *et al.* Algorithmic aspects of protein structure similarity. IEEE FOCS 1999. pp. 512–521.

120

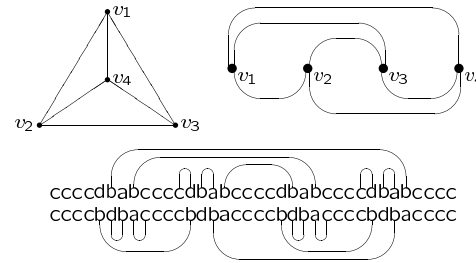
LAPCS(nested, nested) is NP-hard:

- Similar idea as in the MAX-SNP-hardness proof of LAPCS(crossing, plain), with carefully found problem: **MIS on Planar Cubic Bridgeless Graphs**.
 - MIS on planar cubic bridgeless graphs is NP-hard.
 - A planar cubic bridgeless graph has a perfect matching, which can be computed in linear time.
 - Planar Cubic Bridgeless Graphs are subhamiltonian. A graph is *subhamiltonian* if it is a subgraph of a Hamiltonian planar graph.
 - There is a linear time algorithm which, for any planar cubic bridgeless graph, finds a Hamiltonian supergraph of maximum degree at most 5 and a 2-page book embedding in each page every vertex has degree at least 1 (and thus at most 2).
- This time,
 - $S_1 = (CCCCDBAB)^nCCCC$, n — number of vertices
 - $S_2 = (CCCCBDBA)^nCCCC$
 - an edge between v_i and v_j corresponds to an arc connecting a B from the i th segment and a B from the j th segment
 - in S_1 , if v_i has degree 1, then an arc connecting last C and D and an arc connecting B and A; in S_2 , if v_i has degree 1, then an arc connecting next C and A and an arc connecting B and D; making sure that every B is used exactly once.

121

LAPCS(nested, nested) is NP-hard (cont'd):

- An example reduction from K_4 (which is planar, cubic, bridgeless)



- Prove that v_i is in the independent set if and only if the two B from the i th segment are in the common subsequence (otherwise D is in the common subsequence).
- Therefore, there is an MIS of size k if and only if there is an APCS of length $4n + 4 + 2k + (n - k) = 5n + 4 + k$.
- This is not a MAX-SNP-hardness proof !!!

Why ?

because MIS on planar cubic bridgeless graphs admits a PTAS

122

More NP-hardness results:

- Observations:
 - the matches are within segments
 - the possible matches are of forms $\langle 2i-1, 2i-1 \rangle$, $\langle 2i-1, 2i \rangle$, $\langle 2i, 2i-1 \rangle$, and $\langle 2i, 2i \rangle$
 - we are limiting ourselves to
 - * cut S_1 and S_2 into fragments of length 2
 - * the possible matches are required to be inside the fragments
 - * called 2-fragmented LAPCS(nested, nested)
 - * it is already NP-hard
- More hardness results:
 - ℓ -fragmented LAPCS(nested, nested) is NP-hard, $\ell \geq 2$
 - c -diagonal LAPCS(nested, nested) is NP-hard, $c \geq 1$
- Some positive results:
 - 1-fragmented LAPCS(nested, nested) is in P
 - 0-diagonal LAPCS(nested, nested) is in P

123

PTAS for ℓ -fragmented LAPCS(nested, nested):

- Observation:
 - It is closely related to MIS on planar (cubic bridgeless) graphs.
 - MIS on planar graphs admits a PTAS.
- Key ideas:
 - Inside each fragment, there are at most ℓ^2 possible matches, of which some are in fact conflicting each other
 - Make every possible match into a vertex
 - Conflicting matches are connected via an edge
 - To compute an MIS
 - **Problem:** graph is NOT necessarily planar ... inside and long-range
- More ideas:
 - Use a super-vertex to represent the subset of vertices inside a fragment
 - Merge multiple edges, if any, into one (super-)edge
 - Because of the nested arc structures, the new graph is planar

124

PTAS for ℓ -fragmented LAPCS(nested, nested) (cont'd):

- Further ideas: for any integer k ,
 - Compute a k -cover $\{U_1, U_2, \dots, U_k\}$ for this planar graph (polynomial time)
 - For each subset U_i , which is $(k-1)$ -outerplanar, compute a tree decomposition of width at most $3k-4$ (polynomial time)
 - Substitute back the subset of vertices for every super-vertex
 - This substitution increases the tree width to at most $(3k-4)\ell^2$
 - Compute an MIS for U_i (polynomial time)
 - Output the maximum MIS among the k ones computed
 - It is of size at least $\frac{k-1}{k}$ of the optimum

Theorem: ℓ -fragmented LAPCS(nested, nested) admits a PTAS ($\ell \geq 2$) (it runs in $O(2^{(3k-4)c^2} k^3 c^5 (|S_1| + |S_2|))$ time and outputs an APCS of length at least $\frac{k-1}{k}$ times the optimum).

Corollary: α -fragmented LAPCS(nested, nested) admits a PTAS, $c \geq 1$.

125

A $\frac{4}{3}$ -approximation for Unary LAPCS(nested, nested):

- **Unary LAPCS(nested, nested) is still NP-hard ...**
Proof similar (can you prove it ???)
- Needs PTAS, or better approximations:
Given unary (S_1, P_1) and (S_2, P_2) ,
- Key ideas in the $\frac{4}{3}$ -approximation:
Count how many arc-matches in an LAPCS?
Count how many base matches in an LAPCS are not involved in any arc-match?
left bases, right bases, of some arcs in P_1 ?
What do they mean ???

Definition. Left-Priority subsequence:

T is a subsequence of S_1 and $(i_1, i_2) \in P_1$, T cannot contain $S_1[i_2]$ unless it contains $S_1[i_1]$.

Definition. Lep-LAPCS:

Left-Priority Longest Arc-Preserving Common Subsequence.

126

A $\frac{4}{3}$ -approximation for Unary LAPCS(nested, nested) (cont'd):

- Compute a Lep-LAPCS T_1 in $O(n_1^3 n_2)$ time. Denote the length by ℓ_1 .
 - How ?
 - While a base-pair is cut, leave its left base, remove its right base
Since left base has the priority ...
 - **Please fill in the details here**
- Compute a Rip-LAPCS T_2 in $O(n_1^3 n_2)$ time. Denote the length by ℓ_2 .
- Compute a longest APCS T_0 in $O(n_1 n_2)$ time, which does not contain any arc-match. Denote the length by ℓ_0 .
 - How ?
 - Remove the right bases from all the base pairs ...
 - Then compute the classical LCS ...
- Can prove

$$\max\{\ell_1, \ell_2, \ell_0\} \geq \frac{3}{4} \ell^*,$$

where ℓ^* is the length of an LAPCS.

127

Protein structure alignment:

- Measure of similarity:
 - root-mean-square distance
the Euclidean distance between C_α atoms in the two aligned amino acids
minimize the sum / maximum
 - similarity of distance matrices
compute the distance matrices for proteins
align the most similar local sub-matrices (hexa-peptides)
 - scores based on secondary structure alignment
 - scores based on hydrogen bonding pattern
 - ...
 - maximize the aligned **contact**
- Usually NP-hard (or even harder)
- Heuristics, Monte Carlo simulations, etc.

128

Contact map:

- Construction:
 - Linearly order the amino acids
 - Compute the Euclidean distance between every pair of amino acids
 - Connect by an edge for a pair of distance less than pre-defined threshold
 - Hei, ... arc-annotated sequence, again
- Properties:
 - One amino acid could be close to a few other amino acids (not sequentially adjacent)
 - a few — 0 – 4, usually
 - (some assume that such pair should be at least 4 positions apart — does it make the problem easier ???)
- Computational problem — Contact Map Overlap (CMO):
 - two contact maps (n, E) and (m, F)
 - find two subsets $S \subseteq \{1, 2, \dots, n\}$ and $T \subseteq \{1, 2, \dots, m\}$
 - * $|S| = |T|$
 - * there is an order-preserving bijection $f: S \rightarrow T$
 - * $|\{(u, v) \in E : u, v \in S, (f(u), f(v)) \in T\}|$ is maximized
 - Hei, ... unary, again

129

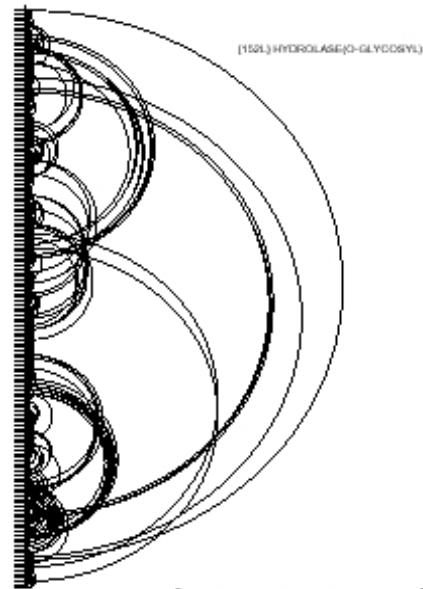
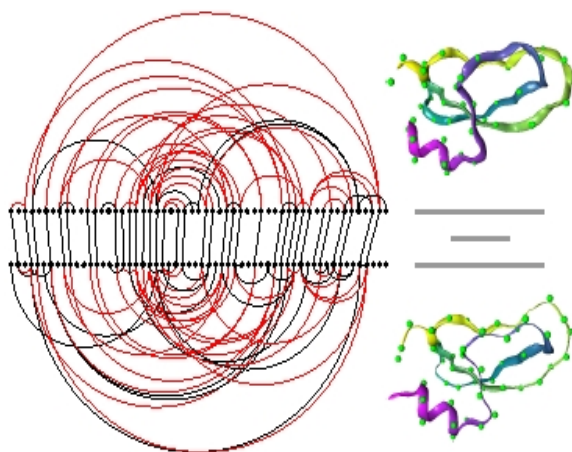


Figure 1: The contact map graph of HYDROLASE(O-GLYCOSYL)

130



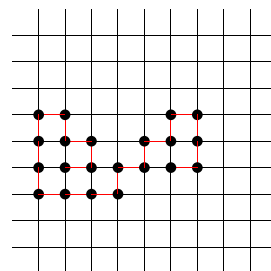
131

Contact map (2):

Theorem The CMO problem is MAX SNP-hard, even when both contact maps have maximum degree 1.

well, ... not surprising :-)

- Special cases:
 - self-avoiding walk on the 2D grid

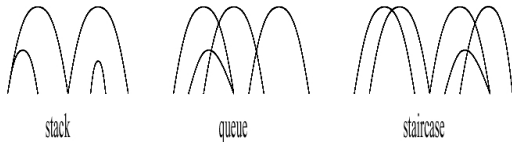


- *stack*: no-crossing (e.g. RNA secondary structures)
- *queue*: not-nested unless sharing an endpoint
- *staircase*:
 - sets of crossing edges
 - no two edges from different sets cross, but can share an endpoint

Theorem A queue can be decomposed into 2 staircases.

132

Contact map (3):



Theorem For two degree-2 contact maps, of which one is either a stack or staircase, the CMO problem can be solved in $O(n^3m^3)$ time.

What you can expect ??? [Dynamic programming](#) (check paper for more details)

Corollary All known RNA tertiary structures (except for one) can be decomposed into 2 degree-1 stacks.

The CMO problem on RNA structures is NP-hard. Nonetheless, it can be approximated within ratio 2. (CMO for RNA secondary structures can be solved in $O(n^3m^3)$ time.)

- **Augmented staircase:** staircase + stack
such that for every staircase edge e and every stack edge f :
 - e and f disjoint
 - e and f share an endpoint
 - e contains f

Theorem For two degree-2 contact maps, of which one is an augmented staircase, the CMO problem can be solved in $O(n^3m^3)$ time.

Contact map (4):

Theorem A self-avoiding walk can be decomposed into 2 stacks and 1 queue.

Corollary A self-avoiding walk has maximum degree-2 except the head and tail nodes. Therefore, the CMO problem on self-avoiding walks can be approximated within ratio 4.

Theorem A self-avoiding walk can be decomposed into 1 stack and 2 augmented staircases.

Corollary The CMO problem on self-avoiding walks can be approximated within ratio 3.

Research problems:

- Designing better approximations
PTAS, or prove the MAX SNP-hardness (on self-avoiding walks)
- [Prove the CMO problem is NP-hard on queues](#)
Or design an efficient algorithm

Lecture 11: Protein Structure Alignment

Agenda:

- More models on protein structure comparison

Reading:

- L. Holm *et al.* *Protein structure comparison by alignment of distance matrices*. Journal of Molecular Biology. (1993) 233, 123–138.
- L. Holm *et al.* *Mapping the protein universe*. Science. (1996) 273, 595–602.
- V. Maiorov *et al.* *Significance of root-mean-square deviation in comparing three-dimensional structures of globular proteins*. Journal of Molecular Biology. (1994) 235, 625–634.
- I. Shindyalov *et al.* *Protein structure alignment by incremental combinatorial extension (CE) of the optimal path*. Protein Engineering. (1998) 9, 739–747.
- D. Goldman *et al.* *Algorithmic aspects of protein structure similarity*. IEEE FOCS 1999. pp, 512–521.

Lecture 11: Protein Structure Alignment

Protein structure alignment:

- Measure of similarity:
 - root-mean-square distance
the Euclidean distance between C_α atoms in the two aligned amino acids
minimize the sum / maximum
 - distance matrices similarity
compute the distance matrices for proteins
align the most similar local sub-matrices (hexa-peptides)
 - scores based on secondary structure alignment
 - scores based on hydrogen bonding pattern
 - ...
 - maximize the aligned **contact**
- Usually NP-hard (or even harder)
- Heuristics, Monte Carlo simulations, etc.

RMSD (1):

- For functions, mainly determined by the backbone structure / folding
- Protein typical represented by its virtual C_α atom chain of residues
- About 1 tenth atoms represented
- For each C_α atom, we have its 3D coordinates x, y, z
- Formally, protein S_1 and S_2 of length n
 - for every C_α atom of the i th amino acid in S_1 , its 3D coordinates x_i, y_i, z_i
 - for every C_α atom of the j th amino acid in S_2 , its 3D coordinates x'_j, y'_j, z'_j
- $d_{S_1}(i, j)$ — Euclidean distance between the C_α atoms of the i th and j th amino acids in S_1
- $d_{S_2}(i, j)$ — Euclidean distance between the C_α atoms of the i th and j th amino acids in S_2
- Minimize the following quantity:

$$D^2(S_1, S_2) = \frac{\sum_{i < j} (d_{S_1}(i, j) - d_{S_2}(i, j))^2}{\frac{n(n-1)}{2}}$$

Note: doesn't assume the alignment ...

137

RMSD (2):

- For functions, mainly determined by the backbone structure / folding
- Protein typical represented by its virtual C_α atom chain of residues
- About 1 tenth atoms represented
- For each C_α atom, we have its 3D coordinates x, y, z
- Formally, protein S_1 and S_2 of length n
 - for every C_α atom of the i th amino acid in S_1 , its 3D coordinates x_i, y_i, z_i
 - for every C_α atom of the j th amino acid in S_2 , its 3D coordinates x'_j, y'_j, z'_j
- Rigid body superpositioning
 - translate both of them such that their centroids are at the origin
 - rotate (one of the two) the coordinate system such that the following is minimized:

$$D^2(S_1, S_2) = \frac{1}{n} \sum_{i=1}^n (\mathbf{R}(x_i, y_i, z_i) - (x'_i, y'_i, z'_i))^2$$

which is *coordinate RMSD*

Note: assuming the sequence alignment ...

138

RMSD (3):

- An important job is to set up the threshold δ to tell that
 - $D(S_1, S_2) < \delta$ indicates S_1 and S_2 are similar
 - $D(S_1, S_2) \geq \delta$ indicates S_1 and S_2 are structurally dissimilar
 Done by empirical studies ...
- For the rigid body superpositioning
 - need assuming an alignment — not really reasonable
 - gaps (allowing indels) make it more computation intensive — NP-hard

139

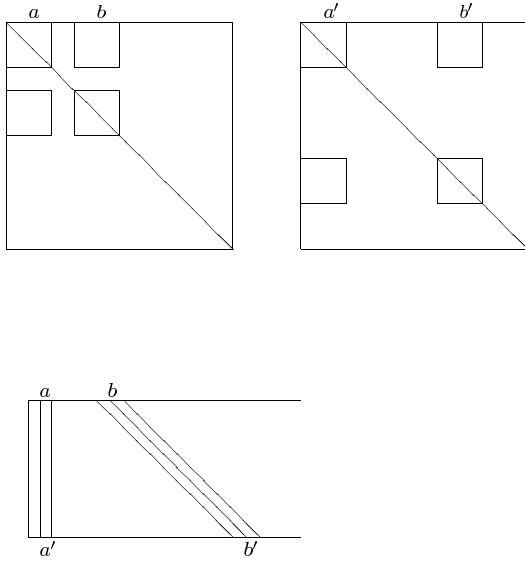
Distance Matrix (1):

- Formally, protein S_1 and S_2 of length n and m , respectively
 - for every C_α atom of the i th amino acid in S_1 , its 3D coordinates x_i, y_i, z_i
 - for every C_α atom of the j th amino acid in S_2 , its 3D coordinates x'_j, y'_j, z'_j
- $d_{S_1}(i, j)$ — Euclidean distance between the C_α atoms of the i th and j th amino acids in S_1
 M_{S_1} — distance matrix for S_1
- $d_{S_2}(i, j)$ — Euclidean distance between the C_α atoms of the i th and j th amino acids in S_2
 M_{S_2} — distance matrix for S_2
- A sliding window of size $L \times L$
 - detect one submatrix of size $L \times L$ each distance matrix achieving the maximum similarity measured by RMSD, or other scores
 - each submatrix pair tells there are two segments from each protein having similar contact / distance
 - for this pair of submatrices, check the corresponding intra-segments similarities
 - construct in this way candidate segment-pairs whose local structures are similar
 - chain them into a connected alignment

Note: No longer sequence alignment maps structure alignment ...

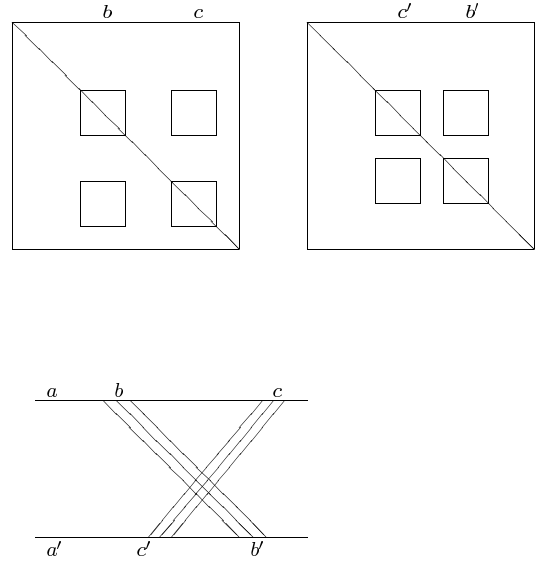
140

Distance Matrix (2):



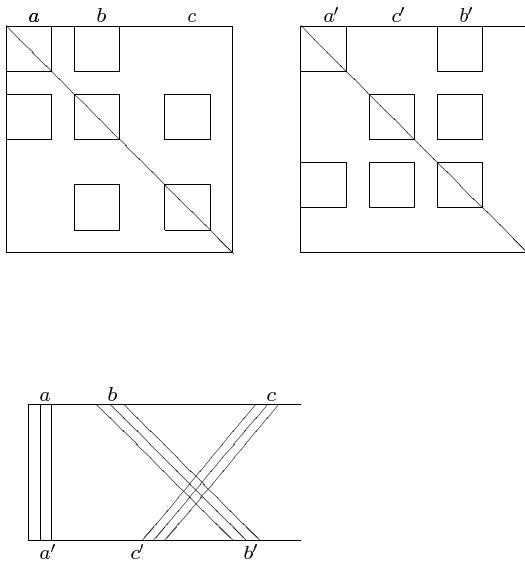
141

Distance Matrix (3):



142

Distance Matrix (4):



143

Incremental Combinatorial Extension (1):

- **Aligned Fragment Pair (AFP):**
 - pair of fragments (of length m , gapless), one from each protein
 - structure similarity
 - measure of similarity

$$D = \frac{1}{m^2} \left(\sum_{k=0}^{m-1} \sum_{\ell=0}^{m-1} |d_{S_1}(k, \ell) - d_{S_2}(k, \ell)| \right)$$

$d_{S_1}(k, \ell)$ — inter-residue distance

- **Feasible combinations**
 - non-overlapping fragments
 - following sequential ordering
 - gap allowed in between fragments
 - gap size bounded — for the sake of computation
- **Evaluation of a pair of combined AFPs**
 - the *independent* set of inter-residue distance

$$D_{ij} = \frac{1}{m} \left(\sum_{(k, \ell) \in S} |d_{S_1}(k, \ell) - d_{S_2}(k, \ell)| \right)$$

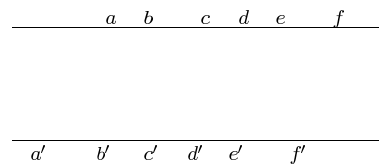
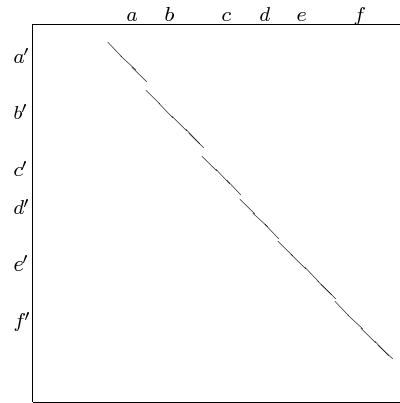
144

Incremental Combinatorial Extension (2):

- Evaluation of a path of feasible AFPs
 - RMSD
 - ignore gaps inserted
 - during extension ignore the statistical significance
 - when terminated evaluate the statistical significance
 - optimization to final path (gaps for example)
- Extension directions
 - starting point
 - * all possible ones
 - extension
 - * all possible ones
 - * best one
 - * best a few (intermediate)

145

Incremental Combinatorial Extension (3):



146

Lecture 12: Exact String Matching

Agenda:

- Fundamental preprocessing
- 1st algorithm
- 2nd algorithm — Boyer-Moore algorithm

Reading:

- Ref[Gusfield, 1997]: pages 1–23

147

Lecture 12: Exact string matching

Genomic database search:

Given a query sequence $Q = a_1a_2 \dots a_n$, find all sequences in the database that are similar to Q .

A database may contain millions of sequences, totaling in billions of bases.

The quadratic time DP algorithm is **NOT** fast enough!

Ideas in BLAST: [Altschul et al., 1990]

- Screen out all sequences which don't share a common substring of length w with Q .
- Often $w = 11$ for DNA and $w = 4$ for protein.
- Consider $n - w + 1$ substrings $a_i a_{i+1} \dots a_{i+w-1}$ of Q , for $i = 1, 2, \dots, n - w + 1$.
- This becomes a *multiple keyword search* problem.

Ideas in PatternHutter: [Li et al., 2002]

- Looking for appearances of length m substrings with at least w matches.
- This becomes an *approximate multiple keyword search* problem.

148

Multiple keyword search:

Problem Given words W_1, W_2, \dots, W_k , and text T , decide if any W_i appears in T .

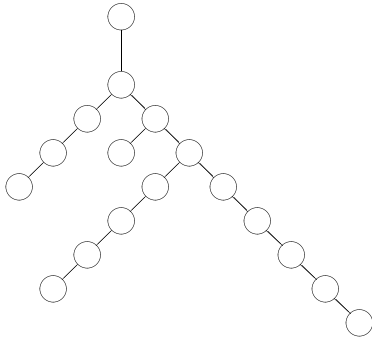
Theorem The problem can be solved in $O(|T| + \sum_{i=1}^k |W_i|)$ time.

Example

$W_1 = \text{pink pig}$
 $W_2 = \text{pig}$
 $W_3 = \text{pink point}$
 $W_4 = \text{park}$

$T = \text{pink pink pig} \dots$

A kind of tree:



149

Definitions:

- string:
A string S is an ordered list of characters written contiguously from left to right.
- substring:
For any string S , $S[i..j]$ is the *contiguous* substring of S that starts at position i and ends at position j .
It is empty if $i > j$.
- prefix:
A substring $S[1..j]$
- suffix:
A substring $S[i..|S|]$
- $S[i]$:
The i th character
- proper substring, proper prefix, proper suffix
- match, mismatch
- The problem:
Given a string P called *pattern* and a long string T called *text*, the **Exact Matching** problem is to find **all occurrences** of P in T .

150

Naive method and its speedup:

- Naive method:
 - align the left end of P with the left end of T
 - compare letters of P and T from left to right, until
 - either a mismatch is found (not an occurrence)
 - or P is exhausted (an occurrence)
 - shift P one position to the right
 - restart the comparison from the left end of P
 - repeat this process till the right end of P shifts past the right end of T

Running time analysis:

$$n = |P|$$

$$m = |T|$$

The worst case number of comparisons is $n \times (m - n + 1)$ ($\Theta(nm)$)

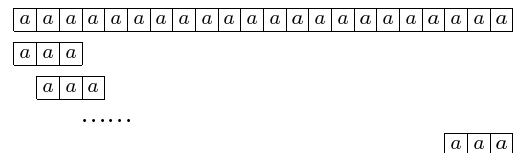
A worst case: $P = \text{aaa}$ and $T = \text{aaaaaaaaaaaaaaaaaaaaa}$

- Not useful as in current database there are trillions of letters (billions of sequences), typically when P is long ...
- Speed it up !!!

151

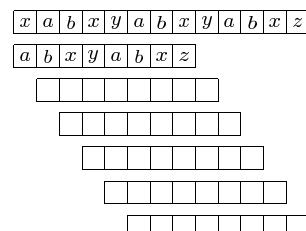
Naive method and its speedup (2):

- Where to speed up?



1. shift P more than one letter, when mismatch occurs but never shift so far as to miss an occurrence
2. after shifting, skip over parts of P to reduce comparisons

- An example: $P = \text{abxyabxz}$, $T = \text{xabxyabxyabxz}$

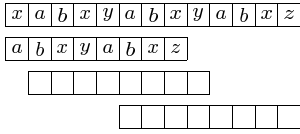
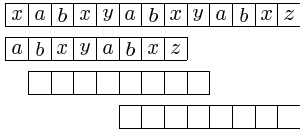


152

Naive method and its speedup (3):

- An example: $P = abxyabxz$, $T = xabxyabxyabxz$

Shifting more than one letter

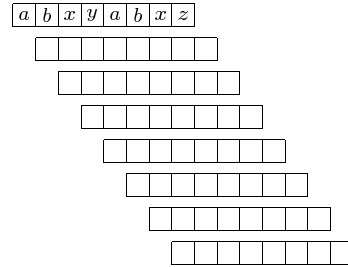
Skipping over parts of P to reduce comparisons

153

Fundamental preprocessing:

- Can be on pattern P (query sequence) or text T (database)
- Given a string S and a position $i > 1$
 $Z_i(S)$ — length of longest common prefix of S and $S[i..|S|]$

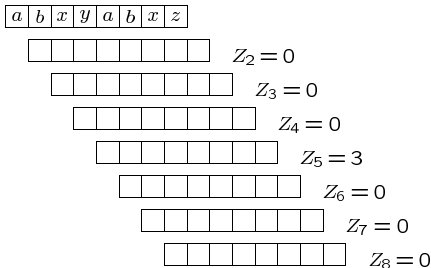
- Example, $S = abxyabxz$



154

Fundamental preprocessing (2):

- Can be on pattern P (query sequence) or text T (database)
- Given a string S and a position $i > 1$
 $Z_i(S)$ — length of longest common prefix of S and $S[i..|S|]$
- Example, $S = abxyabxz$

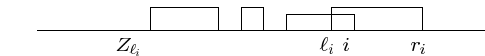


- Intention:
 - Concatenate P and T , inserted by an extra letter \$:
 $S = P\$T$
 - Every i , $Z_i \leq |P|$
 - Every $i > |P| + 1$ and $Z_i = |P|$
records the occurrence of P in T
- Question: running time to compute all the Z_i 's ???
The naive method runs in $\Theta((n+m)^2)$ time ...

155

Fundamental preprocessing (3):

- Goal: linear time to compute all the Z_i 's ...
- Z -box: for $Z_i > 0$, it is box starting at i and ending at $i + Z_i - 1$
- r_i — largest $j + Z_j - 1$ over all $1 < j \leq i$ such that $Z_j > 0$

 ℓ_i — the j with Z -box ending at r_i (tie breaks arbitrarily)

- Computing Z_k :
 - given Z_i for all $1 < i \leq k-1$
 - $r = r_{k-1}$
 - $\ell = \ell_{k-1}$
 - 1. $k > r$: compute Z_k explicitly (updating accordingly r and ℓ if $Z_k > 0$)
 - 2. $k \leq r$: k is in the Z -box starting at ℓ (equivalently substring $S[\ell..r]$)
therefore, $S[k] = S[k - \ell + 1]$, $S[k+1] = S[k - \ell + 2]$, ..., $S[r] = S[r - \ell + 1]$
in other words, $Z_k \geq \max\{Z_{k-\ell+1}, r - k + 1\}$
 - (a) $Z_{k-\ell+1} < r - k + 1$: $Z_k = Z_{r-k+1}$ and r, ℓ remain unchanged
 - (b) $Z_{k-\ell+1} \geq r - k + 1$: $Z_k \geq r - k + 1$ and start comparison between $S[r+1]$ and $S[r - k + 2]$ until a mismatch is found (updating r and ℓ accordingly if $Z_k > r - k + 1$)

156

Fundamental preprocessing (4):

- Conclusions:
 1. Z_k is correctly computed
 2. there are a constant number of operations besides comparisons for each k
 - $|S|$ iterations
 - whenever a mismatch occurs, the iteration terminates
 - whenever a match occurs, r is increased
 3. in total at most $|S|$ mismatches and at most $|S|$ matches
 4. running time $\Theta(|S|)$
 5. Space complexity also $\Theta(|S|)$

Theorem: There is a $\Theta(|S|)$ -time $\Theta(|S|)$ -space algorithm which computes Z_i for all $1 < i \leq |S|$.

Corollary: There is a $\Theta(n + m)$ -time $\Theta(n + m)$ -space algorithm which finds all the occurrences of P in T , where $n = |P|$ and $m = |T|$.

- Notes on the algorithm:
 - alphabet-independent
 - space requirement can be reduced to $\Theta(n)$
why ??? how ???
 - not well suited for multiple patterns searching ...
 - strictly linear — every letter in T has to be compared at least once
- more algorithms to be introduced / designed ...

157

An example:

- $P = abxabxab$, $T = dababxababxab$

- Case 1:

d	a	a	a	b	x	a	b	a	b	x	a	b	x	a	b
a	b	x	a	b	x	a	b								
a	b	x	a	b	x	a	b								

- Case 2:

d	a	a	a	b	x	a	b	a	b	x	a	b	x	a	b
a	b	x	a	b	x	a	b								
a	b	x	a	b	x	a	b								

- Notes:
 - right-to-left comparison
 - won't miss any occurrence
 - some $T[i]$ won't be compared — achieving sublinear time
(in the above example, $T[2]$ and $T[1]$)

158

The Boyer-Moore algorithm:

- Left-to-right shifting (like the naïve algorithm)
- Rule 1: *Right-to-left comparison*
- Rule 2: *Bad character rule*
 - for each $x \in \Sigma$, $R(x)$ denotes the right-most occurrence of x in P (0 if doesn't appear)
 - when a mismatch occurs, $T[k]$ against $P[i]$, shift P right by $\max\{1, i - R(T[k])\}$ places
this makes $T[k]$ against $P[R(T[k])]$, if to the left
 - $|\Sigma|$ space to store R -values
 - not saving anything if $R(T[k])$ to the right
Extended bad character rule: try the rightmost (but to the left) occurrence of $T[k]$
require n space
- Rule 3: *Good suffix rule*
 - when a mismatch occurs, $T[k]$ against $P[i]$
 - find the rightmost occurrence of $P[(i + 1)..n]$ in P such that the letter to the left *differs* $P[i]$
 - shift P right such that this occurrence of $P[(i + 1)..n]$ is against $T[(k + 1)..(n + k - i)]$
 - if there is no occurrence of $P[(i + 1)..n]$:
find the longest prefix of P matches a suffix of $P[(i + 1)..n]$
shift P right such that this prefix is against the corresponding suffix

159

Lecture 14: Exact String Matching

Agenda:

- 3rd algorithm — Knuth-Morris-Pratt algorithm
- Applications

Reading:

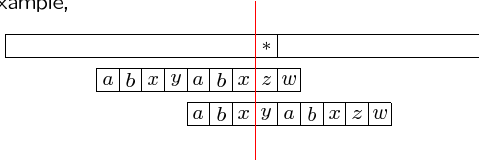
- Ref[Gusfield, 1997]: pages 23–66

160

The Knuth-Morris-Pratt algorithm:

- History:
 - best known
 - not the method of choice, inferior in practice
 - nonetheless can be simply explained and proved
 - basis of Aho-Corasick algorithm for multiple pattern search

- Example,



- (back to) **left-to-right comparison rule**
 - shift P more places without missing any occurrence
- Definition: $sp_i(P), 2 \leq i \leq n$ ($sp_1(P) = 0$)
the length of **longest proper suffix** of $P[1..i]$ that matches a prefix of P

a b x y a b x z w

$sp_1 = sp_2 = sp_3 = sp_4 = 0$
 $sp_5 = 1, sp_6 = 2, sp_7 = 3$
 $sp_8 = 0, sp_9 = 0$

161

The Knuth-Morris-Pratt algorithm (2):

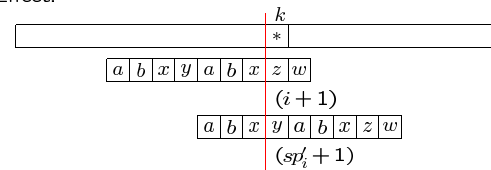
- Definition: $sp'_i(P), 2 \leq i \leq n$ ($sp'_1(P) = 0$)
the length of **longest proper suffix** of $P[1..i]$ that matches a prefix of P ,
with the **additional condition that character $P[i+1]$ differs from $P[sp'_i+1]$** .

a b x y a b x z w

Obviously, $sp'_i \leq sp_i$ for any $i \dots$

$sp'_1 = sp'_2 = sp'_3 = sp'_4 = 0$
 $sp'_5 = 0$
 $sp'_6 = 0$
 $sp'_7 = 3$
 $sp'_8 = 0$
 $sp'_9 = 0$

- Shifting rule: shift P to the right $(i - sp'_i)$ spaces ($i = n$ if an occurrence)
- Effect:

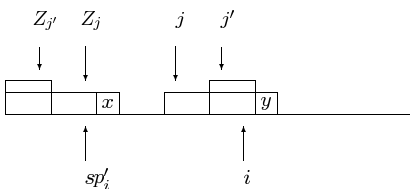


$T[(k - sp'_i)..(k - 1)]$ matches $P[1..sp'_i]$
and thus can skip sp'_i comparisons ...

162

The Knuth-Morris-Pratt algorithm (3):

- Correctness
- Z_j — length of the longest common prefix of P and $P[j..n]$
 sp'_i — length of the longest proper suffix of $P[1..i]$ that matches a prefix of P , with the **additional condition that character $P[i+1]$ differs from $P[sp'_i+1]$**



Therefore,

$$sp'_i = \max_j \{Z_j \mid Z_j = i - j + 1\}$$

- Preprocessing sp'_i 's in linear time $\Theta(n)$

163

The Knuth-Morris-Pratt algorithm (4):

- Running time:
 - in total s phases (of comparison/shift), $s \leq m$
 - every 2 consecutive phases overlap one letter (the mismatched one) from T
 - Therefore, in total $m + s \leq 2m$ comparisons

Question: any letter from T is skipped for comparison?

- A real-time algorithm (def):
any letter is compared at most a constant times
- KMP is **NOT** a real-time algorithm
Question: can you provide an example?
- Converting KMP into a real-time algorithm — more detailed preprocessing
Exercise: figure out the details.

164

The Knuth-Morris-Pratt algorithm (5)

— a way to multiple pattern search:

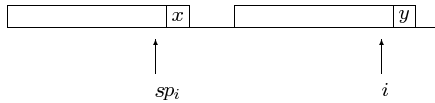
The initial preprocessing idea:

- Preprocessing to compute sp_i values

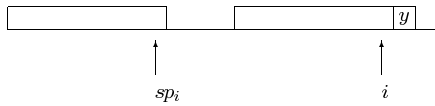
sp_i — length of the longest proper suffix of $P[1..i]$ that matches a prefix of P

sp'_i — length of the longest proper suffix of $P[1..i]$ that matches a prefix of P , with the **additional condition that character $P[i+1]$ differs from $P[sp'_i+1]$**

$$sp'_i = \begin{cases} sp_i, & \text{if } P[sp_i+1] \neq P[i+1] \\ sp_{\{sp_i\}}, & \text{otherwise} \end{cases}$$



- Computing sp_{i+1} :



165

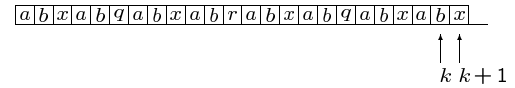
The Knuth-Morris-Pratt algorithm (6)

— the initial preprocessing

- Computing sp_{i+1} :

$$sp_{i+1} = \begin{cases} sp_i + 1, & \text{if } P[sp_i+1] = P[i+1] \\ sp_{\{sp_i\}} + 1, & \text{if } P[sp_{\{sp_i\}}+1] = P[i+1] \\ \dots, & \text{if } P[sp_{\{sp_{\{sp_i\}}\}}+1] = P[i+1] \end{cases}$$

- An example,



166

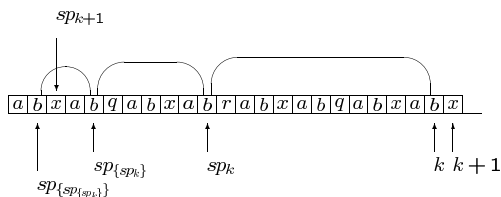
The Knuth-Morris-Pratt algorithm (7)

— the initial preprocessing

- Computing sp_{i+1} :

$$sp_{i+1} = \begin{cases} sp_i + 1, & \text{if } P[sp_i+1] = P[i+1] \\ sp_{\{sp_i\}} + 1, & \text{if } P[sp_{\{sp_i\}}+1] = P[i+1] \\ sp_{\{sp_{\{sp_i\}}\}} + 1, & \text{if } P[sp_{\{sp_{\{sp_i\}}\}}+1] = P[i+1] \\ \dots, & \end{cases}$$

- An example,



167

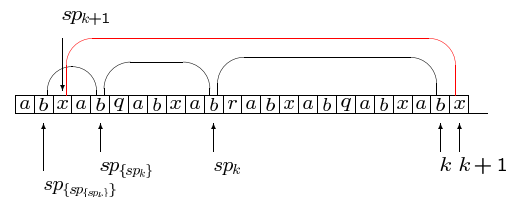
The Knuth-Morris-Pratt algorithm (8)

— the initial preprocessing

- Computing sp_{i+1} :

$$sp_{i+1} = \begin{cases} sp_i + 1, & \text{if } P[sp_i+1] = P[i+1] \\ sp_{\{sp_i\}} + 1, & \text{if } P[sp_{\{sp_i\}}+1] = P[i+1] \\ sp_{\{sp_{\{sp_i\}}\}} + 1, & \text{if } P[sp_{\{sp_{\{sp_i\}}\}}+1] = P[i+1] \\ \dots, & \end{cases}$$

- An example,



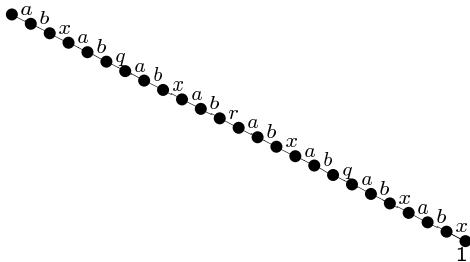
- Preprocessing can be done in linear time

Proof.

168

Keyword tree:

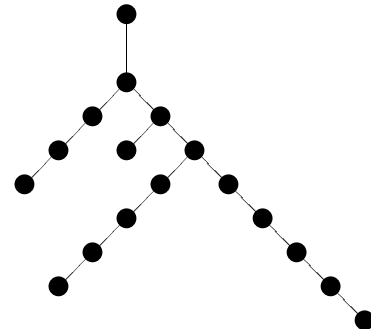
- Given a set of patterns $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$, the keyword tree \mathcal{K} is a tree:
 - rooted, directed
 - each edge is labeled with one letter
 - edges coming out of a node have distinct labels
 - every pattern P_i is spelled out
 - every leaf maps to some pattern
- A keyword tree containing one keyword
 $P_1 = abxabqabxabrabxabqabxabx$:



169

Keyword tree (2):

- A keyword tree containing multiple keywords:
 - $P_1 = \text{pink pig}$
 - $P_2 = \text{pig}$
 - $P_3 = \text{pinpoint}$
 - $P_4 = \text{park}$



- A keyword tree can be constructed in linear time (in the sum of lengths of patterns).
 - provided the alphabet is fixed (and thus of constant size)

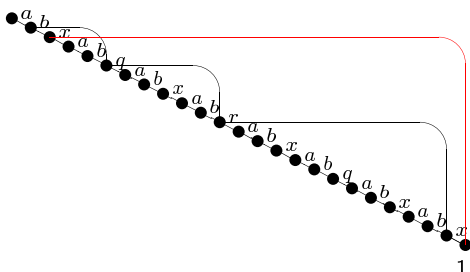
170

Multiple pattern matching problem:

- Given a set of pattern $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$ and a text (database) T
 - find all the occurrences of all the patterns ...
- n — sum of the lengths of patterns
 m — length of text
- Previous results imply a search algorithm in $\Theta(n + km)$ time

there are algorithms running in $\Theta(n + m + \ell)$ time, where
 ℓ — total number of occurrences of all the patterns

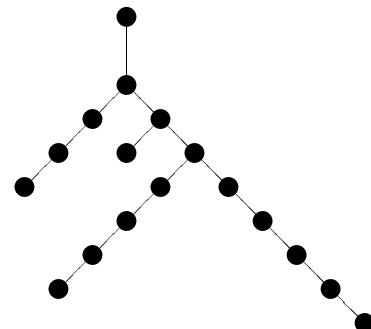
- Using keyword tree of \mathcal{P} ...
- An idea to speed up — from KMP



171

Multiple pattern matching problem (2):

- The keyword tree preprocessing:
 - $P_1 = \text{pink pig}$
 - $P_2 = \text{pig}$
 - $P_3 = \text{pinpoint}$
 - $P_4 = \text{park}$



- $lp(v)$ — the length of the longest proper suffix of string $\mathcal{L}(v)$ that is a prefix of some pattern in \mathcal{P}
- $lp(v)$ for all the v 's in \mathcal{K} can be computed in linear time — $\Theta(n)$
- failure links $v \rightarrow n_v$

172

Aho-Corasick algorithm:

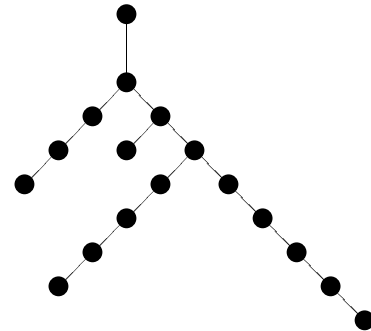
- Create the keyword tree
- Computer the $lp(v)$ and n_v for every v in the keyword tree
- During the searching
 - every occurrence is reported $\Theta(\ell)$
 - whenever a mismatch, T shifted $lp(v)$ spaces to the left
 - whenever a match, $T[i]$ never compared again
- Running time $\Theta(m + \ell)$
- Therefore, in total $\Theta(n + m + \ell)$ time

173

Aho-Corasick algorithm (2):

- The keyword tree:

$P_1 = \text{pink pig}$
 $P_2 = \text{pig}$
 $P_3 = \text{pinpoint}$
 $P_4 = \text{park}$



- $T = \text{pinkpinkpig}$

174

Exact string matching applications:

- Sequence-tagged-sites
- Exact string matching with wild cards
- Two-dimensional exact matching
- Regular expression pattern matching

Reading: Ref[Gusfield, 1997], pages 61–66

175

Agenda:

- Introduction
- Construction
- Applications

Reading:

- Ref[Gusfield, 1997]: pages 87–168

176

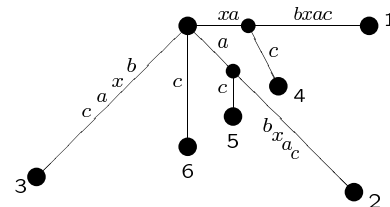
Introduction:

- Given a finite alphabet Σ , a string S of length m
E.g., $S = abxabc$
- Suffix tree of S :
 - rooted, directed
 - edges labeled by (non-empty) substrings of S
 - could be a single letter
 - edges coming out of a node start with distinct letters
 - exactly m leaves
 - leaf i spells out suffix $S[i..m]$
- Keyword tree for a set of patterns:
 - rooted, directed
 - edges labeled with letters
 - edges coming out of a node have distinct labels
 - every pattern is spelled out
 - every leaf maps to some pattern

177

Introduction (2):

- Given a finite alphabet Σ , a string S of length m
E.g., $S = xabxabc$
- Suffix tree of S :
 - rooted, directed
 - edges labeled by (non-empty) substrings of S
 - could be a single letter
 - edges coming out of a node start with distinct letters
 - exactly m leaves
 - leaf i spells out suffix $S[i..m]$



- Have to assume $sp_{m-i+1}(S[i..m]) = 0 \dots$ Why ???

Solution: if $sp_{m-i+1}(S[i..m]) \neq 0$, append to it an extra letter $\$ \notin \Sigma \dots$

178

First construction:

- Assuming now $sp_{m-i+1}(S[i..m]) = 0$
- Make every suffix as a pattern $P_i = S[i..m]$
- Apply the linear time keyword tree construction algorithm
- Concatenate "paths" into "edges"
- Running time:

linear in the sum of the lengths of patterns — $\sum_{i=1}^m |P_i|$
 $= \frac{m(m+1)}{2}$
 — a $\Theta(m^2)$ construction algorithm :-)
- Next goal: design a linear time algorithm $\Theta(m)$

179

Why suffix tree — 1st application:

- Suppose in $\Theta(m)$ time we can build the suffix tree for text T
- Given any pattern P — at any time
 - match letters of P along the suffix tree
 - ... until
 - either no more matches are possible

P doesn't occur anywhere in T

 - or P is exhausted

the labels of the leaves inside the subtree under the last matching edge are the starting positions of the occurrences
- Conclusion:
 - Exact string matching done in $\Theta(m + n + \ell)$ time
 - Exact multiple strings matching done in $\Theta(m + n + \ell)$ time
 - ℓ — number of occurrences

Other applications:

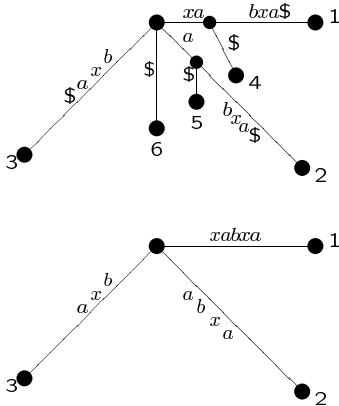
- Multiple keyword search
- Longest repeating substring
- Longest common substring of two (or more) strings

180

Ukkonen's linear time construction:

- Implicit suffix tree for $S\$$ — from the suffix tree
 1. remove every copy of $\$$
 2. remove every edge without labels
 3. remove the degree-2 internal node (except the root) need to concatenate the edge labels ...

E.g., $S = xabxa$



- T_m contains almost all the information about the suffix tree

181

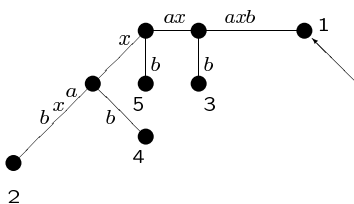
Ukkonen's linear time construction (2):

- T_i — implicit suffix tree for $S[1..i]\$$
- Construct T_i incrementally
 - from T_i to T_{i+1}
 - need to add $S[i+1]$ to every suffix of $S[1..i]$ and the empty string
there are $i+1$ of them ...
 - append $S[i+1]$ to suffix $S[j..i]$ — becoming a suffix of $S[1..(i+1)]$
 $j = 1, 2, \dots, i, i+1$
 1. if $S[j..i]$ ends at a leaf, append $S[i+1]$ to the corresponding edge label
 2. if $S[j..i]$ ends at an internal node
 - (a) if there is an edge out of the node with label begins with $S[i+1]$, done
 - (b) otherwise add an edge and a leaf, with edge label ' $S[i+1]$ '
 3. if $S[j..i]$ doesn't end at any node
 - (a) if the succeeding letter in the edge label is ' $S[i+1]$ ', done
 - (b) otherwise add an internal node right after (break into two edges) and an edge adjacent with a leaf. label for the new edge is ' $S[i+1]$ '
- Straightforward implementation $\Theta(m^3)$

182

Ukkonen's linear time construction (3) — speedup:

- How?
- Key thing: locate the ending position of $S[j..i]$ in T_i
 $S[1..5] = axaxb$, try adding $S[6] = a$:

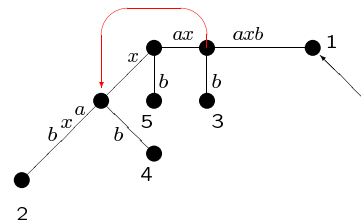


- $j = 1$ — easy (use a pointer pointing to the longest path in T_i)
append a to the edge label (entering the leaf spells out $S[1..i]$)
- denote the leaf edge as $(v, 1)$
 1. if v is the root: $j = 2$ is done straightforwardly
 2. if v isn't the root:
there is another node, denoted $s(v)$, such that if root-to- v spells out $S[1..l]$ then root-to- $s(v)$ spells out $S[2..l]$
 3. when we have the information on $s(v)$, continue search from it
not necessarily from the root again ...

183

Ukkonen's linear time construction (4) — speedup:

- $S[1..5] = axaxb$, try adding $S[6] = a$:



In this example, save a few comparisons between x and $x \dots$

- Notes:
 - need to record $s(v)$ if a new node v is created
 - it doesn't give a "faster" algorithm
- Trick 1 — skip/count
In this example, we know that a is appended 3 letters after $S[1..l] = ax$
Therefore, if we can record for every edge, the length of its label ...

184

Ukkonen's linear time construction (5) — speedup:

- Analysis of trick 1:
 - every node has a *depth* — # of nodes on the path from root
 - the depth of v is at most one greater than the depth of $s(v)$
 - constructing T_{i+1} from T_i :
 - * decrease node depth at most $2m$
 - * could increase a lot but bounded by $3m$ — why? since the maximum depth is m

so construction done in $O(m)$ time

Applying Trick 1 gives an $O(m^2)$ time suffix tree building algorithm

- One observation to overcome the $\Theta(m^2)$ -barrier
total number of letters in the edge labels could reach $\Theta(m^2)$
...
- An alternate way to represent labels

using position intervals $[start, end]$ to represent label $S[start..end]$

185

Agenda:

- Introduction
- Construction
- Applications

Reading:

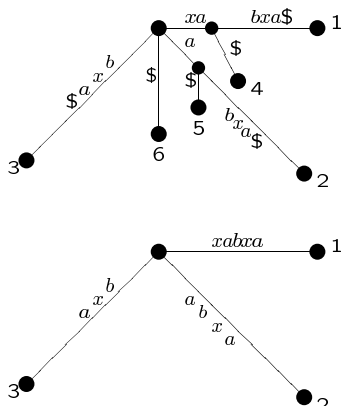
- Ref[Gusfield, 1997]: pages 87–168

186

Ukkonen's linear time construction:

- Implicit suffix tree for $S\$$ — from the suffix tree
 1. remove every copy of $\$$
 2. remove every edge without labels
 3. remove the degree-2 internal node (except the root) need to concatenate the edge labels ...

E.g., $S = xabxa$



- T_m contains almost all the information about the suffix tree

187

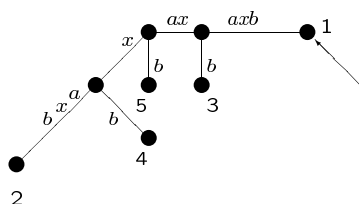
Ukkonen's linear time construction (2):

- T_i — implicit suffix tree for $S[1..i]\$$
- Construct T_i incrementally
 - from T_i to T_{i+1}
 - need to add $S[i+1]$ to every suffix of $S[1..i]$ and the empty string
there are $i+1$ of them ...
 - append $S[i+1]$ to suffix $S[j..i]$ — becoming a suffix of $S[1..(i+1)]$
 $j = 1, 2, \dots, i, i+1$
 1. if $S[j..i]$ ends at a leaf, append $S[i+1]$ to the corresponding edge label
 2. if $S[j..i]$ ends at an internal node
 - (a) if there is an edge out of the node with label begins with $S[i+1]$, done
 - (b) otherwise add an edge and a leaf, with edge label ' $S[i+1]$ '
 3. if $S[j..i]$ doesn't end at any node
 - (a) if the succeeding letter in the edge label is ' $S[i+1]$ ', done
 - (b) otherwise add an internal node right after (break into two edges) and an edge adjacent with a leaf. label for the new edge is ' $S[i+1]$ '
- Straightforward implementation $\Theta(m^3)$

188

Ukkonen's linear time construction (3) — speedup:

- How?
- Key thing to do: to locate the ending position of $S[j..i]$ in T_i
 $S[1..5] = axaxb$, try adding $S[6] = a$:

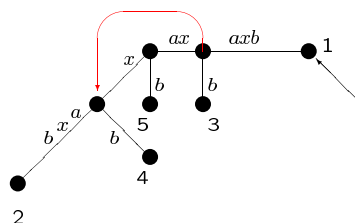


- $j = 1$ — easy (use a pointer pointing to the longest path in T_i)
 append a to the edge label (entering the leaf spells out $S[1..i]$)
- denote the leaf edge as $(v, 1)$
 1. if v is the root: $j = 2$ is done straightforwardly
 2. if v isn't the root:
 there is another node, denoted $s(v)$, such that if root-to- v spells out $S[1..i]$ then root-to- $s(v)$ spells out $S[2..i]$
 3. when we have the information on $s(v)$, continue search from it
 not necessarily from the root again ...

189

Ukkonen's linear time construction (4) — speedup:

- $S[1..5] = axaxb$, try adding $S[6] = a$:

In this example, save a few comparisons between x and $x \dots$

- Notes:
 - need to record $s(v)$ if a new node v is created
 - it doesn't give a "faster" algorithm
- Trick 1 — skip/count
 In this example, we know that a is appended 3 letters after $S[1..i] = ax$
 Therefore, if we can record for every edge, the length of its label ...

190

Ukkonen's linear time construction (5) — speedup:

- Analysis of trick 1:
 - every node has a *depth* — # of nodes on the path from root
 - the depth of v is at most one greater than the depth of $s(v)$
 - constructing T_{i+1} from T_i :
 - * decrease node depth at most $2m$
 - * could increase a lot but bounded by $3m$ — why?
 since the maximum depth is m

so construction done in $O(m)$ timeApplying Trick 1 gives an $O(m^2)$ time suffix tree building algorithm

- One observation to the $\Theta(m^2)$ -barrier
 Total number of letters in the edge labels could reach $\Theta(m^2)$
 ...
- An alternate way to represent labels is **necessary**

Use position intervals $[start, end]$ to represent label $S[start..end]$

191

Ukkonen's linear time construction (6) — speedup:

- Another observation:
 When $S[j..i]$ **doesn't** end at a leaf and there is an extending edge whose label starts with ' $S[i+1]$ ', we are done
 "we are done" TOO when considering
 $S[(j+1)..i]$, $S[(j+2)..i]$, ..., $S[(i+1)..i] = \emptyset$
 Why ??? — proved by contradiction ...
- Trick 2: whenever this happens, T_{i+1} is built
- 3rd observation:
 Once a leaf, always a leaf
 — there is no case which extends a leaf
- Trick 3: use a parameter e to denote the last position thus to skip the extensions (only need to update e once per iteration)
 Recall that a leaf edge is labeled $[\ell, e]$...

192

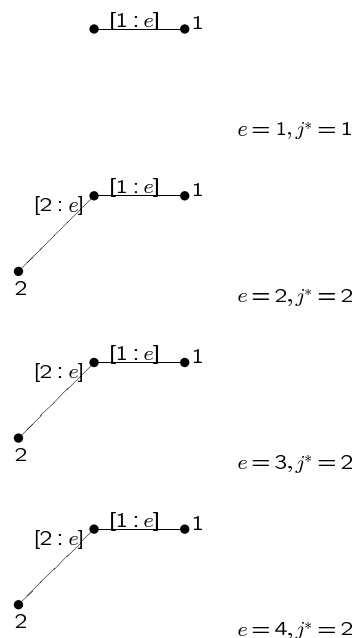
Ukkonen's linear time construction (7) — analysis:

- Appending rules (recall):
append $S[i+1]$ to suffix $S[j..i]$, for $j = 1, 2, \dots, i, i+1$
 1. if $S[j..i]$ ends at a leaf, append $S[i+1]$ to the corresponding edge label
 2. if $S[j..i]$ ends at an internal node
 - (a) if there is an edge out of the node with label begins with $S[i+1]$, done
 - (b) otherwise add an edge and a leaf, with edge label ' $S[i+1]$ '
 3. if $S[j..i]$ doesn't end at any node
 - (a) if the succeeding letter in the edge label is ' $S[i+1]$ ', done
 - (b) otherwise add an internal node right after (break into two edges) and an edge adjacent with a leaf. label for the new edge is ' $S[i+1]$ '
- Analysis of Tricks 1 + 2 + 3 — amortized analysis:
 - increment e to skip the first j^* (from last phase) extensions — **why we can skip them?**
 - apply trick 1 to continue until trick 2 can be applied, say at j^{th} extension
set $j^* = j - 1$ — **update j^* for next phase use**
 - next phase we can skip the first j^* extensions ...
 - every two consecutive phases overlap at most 1 index
 - conclusion: in linear time

193

Ukkonen's linear time construction (8) — example:

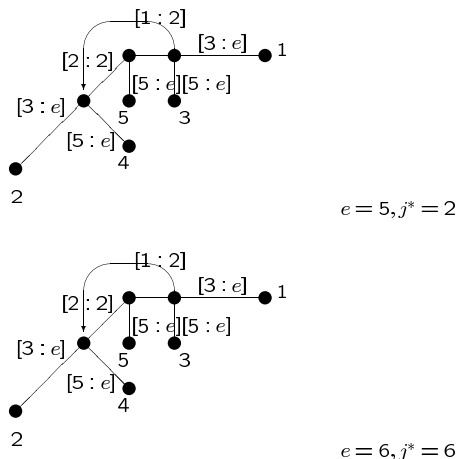
- $S[1..6] = axaxba$:



194

Ukkonen's linear time construction (9) — example:

- $S[1..6] = axaxba$:



195

Ukkonen's linear time construction (10) — algorithm:

- Append \$ to S and execute on T_m
 - Correctness:
Every suffix is spelled out by some root-to-leaf path

no suffix is a prefix of some other suffix
 - Generalized suffix tree for a set of strings — How?
 - concatenate strings into one, by adding some extra letters
 - build suffix tree for one string, then on top of it build for another, then on top of it build for another, ...
 - Applications:
 1. **exact pattern search**
 2. **longest common substring** (NOT longest common subsequence)
 3. **tandem repeat**
 4. suffix array — a good tool to ...
 - 5.
 - 6.
 - 7.
 -
- This list will be lengthened, by YOU ... :-)

196

Ukkonen's linear time construction — question:

- Going back to the beginning, why **implicit** suffix trees ???

Why not build the suffix tree directly ???

197

Agenda:

- Introduction to genome-level mutations
- Reversals (or Inversions)
- Signed Reversals

Reading:

- Ref[JXZ, 2002]: chapter 6.
- Ref[Gusfield, 1997]: pages 492–498
- Hannenhalli & Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1999), 1–27.

198

Gene-level mutations vs genome-level mutations

- Gene-level mutations:
 - single nucleotide/amino acid substitution
 - single nucleotide/amino acid inser-/dele-tion (space)
 - block of nucleotides/amino acids inser-/dele-tion (gap)
- Gene-level mutations reflect:
 - how does a single gene evolve over the time
 - how are a family of genes related to each other
- Genome-level (chromosome-level) mutations:
 - duplication with modification
 - reversal
 - transposition
 - translocation (special cases: fusion and fission)
- Genome-level (chromosome-level) mutations reflect:
 - evolution of the whole genome (might not be seen at the gene-level comparisons)
 - how do species diverge and relate to each other
 - deeper evolutionary relationship

199

Genome-level mutations: an example

Complete mitochondrial genomes:

gene name	label
ND2	1
COX1	2
COX2	3
ATP8	4
ATP6	5
COX3	6
ND3	7
ND5	8
ND4	9
ND4L	10
ND6	11
CYTB	12
ND1	13
<hr/>	
Anopheles gambiae	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
Apis mellifera ligustica	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
Artemia franciscana	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
Drosophila yakuba	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
Balaenoptera musculus	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11, 12)
Balaenoptera physalus	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11, 12)
Bos taurus	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11, 12)
Cyprinus carpio	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11, 12)
Didelphis virginiana	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11, 12)
Halichoerus grypus	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11, 12)
Mus musculus	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11, 12)
Phoca vitulina	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11, 12)
Xenopus laevis	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11, 12)
Gallus gallus	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11)
Strongylocentrotus purpuratus	(13, 1, 2, 10, 3, 4, 5, 6, 7, 9, 8, 11, 12)
Paracentrotus lividus	(13, 1, 2, 10, 3, 4, 5, 6, 7, 9, 8, 11, 12)
Petromyzon marinus	(12, 13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 11)

200

The general problem:

- Assumptions:
 - a same gene found from both species
 - ignore the gene sequence difference — identical
 - ignore multiple copies — use one copy
 - label genes using integers
- For example,

Anopheles gambiae	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
Gallus gallus	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11)
- Question:
 - how were these two genomes evolved from some common ancestral genome?
(which also contains the same set of genes)
 - equivalently, how was one genome evolved from the other?
- A reduction:
 - assuming one species genome is the identity permutation ...
 - how was a permutation evolved from the identity permutation?
- Measurement:

Applying the [parsimony](#) rule:

Nature usually takes the path with least resistance, so the scenario with the least amount of genome-level changes is likely to be the most probable one,

201

The concrete technical problems:

- genome rearrangements by reversals only
- genome rearrangements by transpositions only
- genome rearrangements by reversals and transpositions
- genome rearrangements by all mutations
- signed genome rearrangements by reversals only
- signed genome rearrangements by transpositions only
- signed genome rearrangements by reversals and transpositions
- etc.*

202

Genome rearrangements by reversals:

- Formal description of the problem:
 - $\pi = (\pi_1, \pi_2, \dots, \pi_{i-1}, \pi_i, \pi_{i+1}, \dots, \pi_{j-1}, \pi_j, \pi_{j+1}, \dots, \pi_n)$ a permutation on $(1, 2, \dots, n)$
 - a reversal $r(i, j)$, $1 \leq i < j \leq n$, on π makes π into $\pi' = (\pi_1, \pi_2, \dots, \pi_{i-1}, \pi_j, \pi_{j-1}, \dots, \pi_{i+1}, \pi_i, \pi_{j+1}, \dots, \pi_n)$
 - given any permutation, find a shortest series of reversals that transforms π into the identity permutation I .

- An example,

Anopheles gambiae	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
Gallus gallus	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11)

$(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11) + R(9, 11) \rightarrow$
 $(13, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 11) + R(12, 13) \rightarrow$
 $(13, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) + R(2, 13) \rightarrow$
 $(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1) + R(1, 13) \rightarrow$
 $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$

Question 1: are 4 reversals really necessary?

Question 2: is it always possible to transform one permutation into identity by reversals only?

203

Genome rearrangements by reversals (2):

- Question 2: is it always possible to transform one permutation into identity by reversals only?

Answer: yes.

- Question 1: are 4 reversals really necessary?

Let us see.

- Definition: a **breakpoint** in permutation π occurs between numbers π_i and π_{i+1} , for $1 \leq i < n$, if and only if $|\pi_i - \pi_{i+1}| \neq 1$.

There is a breakpoint at the front of the permutation if $\pi_1 \neq 1$; and there is a breakpoint at the end of the permutation if $\pi_n \neq n$.

— Another way to define breakpoints is to add $\pi_0 = 0$ and $\pi_{n+1} = n + 1$ to get a permutation on $n + 2$ numbers.

- Key observation:
 - a permutation without breakpoints if and only if it is the identity
 - new goal: reduce the number of breakpoints to 0
 - every reversal reduces this number by at most 2

Theorem Let $b(\pi)$ denote the number of breakpoints in π . The number of reversals to transform π into the identity is at least $\frac{b(\pi)}{2}$.

204

Genome rearrangements by reversals (3):

- The example (13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11):
The number of breakpoints in it is 5. So, it requires at least 3 reversals.
- Idea in designing an algorithm:
Try to find a segment that, by reversing it we can reduce the number of breakpoints!!!
 - definition: a **strip** is a *maximal* segment containing no breakpoints inside
 - the example (13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11) contains 4 strips: (13), (1, 2, 3, 4, 5, 6, 7), (10, 9, 8), (12, 11)
 - definitions: **decreasing / increasing strip**
 - reversal intervals should contain a **full strip** — otherwise will be creating new breakpoints
(can be shown true precisely — can you ???)

Lemma Whenever there is a decreasing strip, there is a reversal which decreases $b(\pi)$ by at least 1. Furthermore, the reversal can be determined in linear time.

Proof.

- for the example (13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11), the reversal is $R(9, 11)$

205

Genome rearrangements by reversals (4):

- Designing an algorithm:
 - need to consider the case where there is no decreasing strips ...
easy —
 - when there is none, simply inverse an increasing strip (of length ≥ 2)
 - pseudocode:
Alg1 π

```

while ( $b(\pi) > 0$ ) do
  if (there is a decreasing strip) then
    find the reversal to reduce  $b(\pi)$  by at least 1
  else
    inverse one increasing strip of length at least 2

```
 - analysis:
 - two consecutive reversals reduce $b(\pi)$ by at least 1
 - at most $2b(\pi)$ reversals transform π into identity
 - at least we need $\frac{b(\pi)}{2}$ reversals

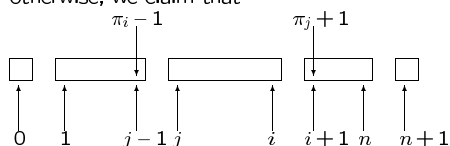
Conclusion. Alg1 is a 4-approximation.

- Genome rearrangements by reversals is NP-hard.
- We answered Question 2 affirmatively:
Is it always possible to transform one permutation into identity by reversals only?

206

Genome rearrangements by reversals (5):

- Improve the above design:
 - idea 1:** can we guarantee to reduce $b(\pi)$ and, at the same time, leave a decreasing strip?
 - idea 2:** if we can't, can we guarantee to reduce $b(\pi)$ by 2?
— such that on average, every reversal reduces $b(\pi)$ by at least 1
- Search for the desired reversal:
 - assuming there is a decreasing strip ...
 - examine the smallest number, say π_i , in any decreasing strip
— try to remove the breakpoint involves π_i and $\pi_i - 1$ and leave a decreasing strip — if successful, we are done
 - otherwise examine the largest number, say π_j , in any decreasing strip
— try to remove the breakpoint involves π_j and $\pi_j + 1$ and leave a decreasing strip — if successful, we are done
 - otherwise, we claim that



- therefore, reversal $R(j, i)$ reduces $b(\pi)$ by 2

207

Genome rearrangements by reversals (6):

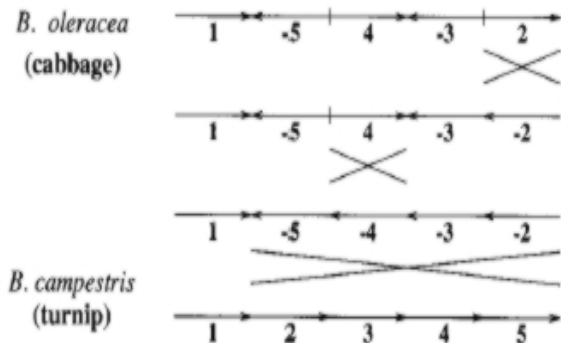
- The above is a 2-approximation algorithm.
- This is a *look-ahead* style algorithm:
Whenever you execute one step, you also prepare for the next step ...
- More carefully,
There is a 1.75-approximation algorithm for genome rearrangements by reversals.
- When genes have directions — signed genome rearrangements by reversals

Good news: polynomial time solvable (Hannenhalli & Pevzner)!
The speed has been improved to almost linear-time.
- A closely related problem: Breakpoint graph decomposition
 - 1.5-approximation
 - 1.46...-approximation
 - 1.42...-approximation
 - 1.375-approximation

208

Signed genome rearrangements by reversals (7):

Transforming the mitochondrial DNA of turnip into that of cabbage by reversals.



209

Agenda:

- More on reversals
- Transpositions
- Reversal & transpositions

Reading:

- Ref[JXZ, 2002]: Chap 6
- Ref[Pevzner, 2000]: pages 175–228
- A. Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*. 12(1999), 91–110.
- V. Bafna *et al.* Sorting permutations by transpositions. *Proceedings of SODA'95*. Pages 614–623.
- Q. Gu *et al.* A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science*. 210(1999), 327–339.

210

Genome rearrangements by reversals:

- Formal description of the problem:
 - $\pi = (\pi_1, \pi_2, \dots, \pi_{i-1}, \pi_i, \pi_{i+1}, \dots, \pi_{j-1}, \pi_j, \pi_{j+1}, \dots, \pi_n)$ a permutation on $(1, 2, \dots, n)$
 - a reversal $r(i, j)$, $1 \leq i < j \leq n$, on π makes π into $\pi' = (\pi_1, \pi_2, \dots, \pi_{i-1}, \pi_j, \pi_{j-1}, \dots, \pi_{i+1}, \pi_i, \pi_{j+1}, \dots, \pi_n)$
 - given any permutation, find a shortest series of reversals that transforms π into the identity permutation $\mathcal{I} = (1, 2, \dots, n-1, n)$.

- An example,

Anopheles gambiae	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
Gallus gallus	(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11)

$(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11) + R(9, 11) \rightarrow$
 $(13, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 11) + R(12, 13) \rightarrow$
 $(13, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) + R(2, 13) \rightarrow$
 $(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1) + R(1, 13) \rightarrow$
 $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$

Question 1: are 4 reversals really necessary?

Still trying to answer ...

Question 2: is it always possible to transform one permutation into identity by reversals only?

Yes.

211

Breakpoint graph:

- Given a permutation $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, extend it to have $\pi_0 = 0$ and $\pi_{n+1} = n + 1$
- Definition: a **breakpoint** in permutation π occurs between numbers π_i and π_{i+1} , for $0 \leq i \leq n$, if and only if $|\pi_i - \pi_{i+1}| \neq 1$.
- Key observation:
 - a permutation without breakpoints if and only if it is the identity
 - new goal: reduce the number of breakpoints to 0
 - every reversal reduces this number by at most 2

Theorem Let $b(\pi)$ denote the number of breakpoints in π . Let $d(\pi)$ denote the minimum number of reversals to transform π into \mathcal{I} . Then,

$$d(\pi) \geq \frac{b(\pi)}{2}.$$

Theorem Genome rearrangement by reversals admits a 2-approximation algorithm.

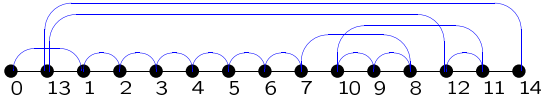
- Breakpoint graph $G(\pi)$ of π :
 - $n + 2$ vertices $\{0, 1, 2, \dots, n, n + 1\}$
 - a black edge connecting π_i and π_{i+1} for every $i : 0 \leq i \leq n$
 - a gray edge connecting π_i and π_j if $|\pi_i - \pi_j| = 1$
- $\pi = (0, 13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11, 14)$ and $n = 13$:



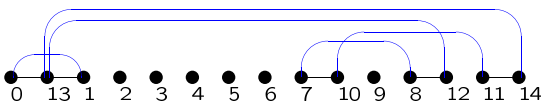
212

Breakpoint graph (2):

- Breakpoint graph $G(\pi)$ of π :
 $\pi = (0, 13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11, 14)$ and $n = 13$:



- Properties:
 - every vertex is incident with the same number of black and gray edges
 - therefore, exists an Eulerian alternating cycle
 - therefore, exists an alternating cycle decomposition
 - $c(\pi)$ — the maximum number of cycles in any alternating cycle decomposition
- For the above example, besides the 1-cycles (number of black edges therein), we have one more alternating cycle:



Can show this is a maximum alternating cycle decomposition
 $\rightarrow c(\pi) = 10$

213

Breakpoint graph (3):

- Breakpoint graph $G(\pi)$ of π :
 $\pi = (0, 13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11, 14)$ and $n = 13$:
 has $c(\pi) = 10$.

Lemma. For every permutation π , apply one reversal to transform it into π' . Then, $|c(\pi) - c(\pi')| \leq 1$.

Proof. By distinguishing the cases where the two black edges involved are in one cycle or two cycles in a maximum alternating cycle decomposition.

- In other words: a reversal increases $c(\pi)$ by at most 1.
- Observation: $c(I) = n + 1$.

Theorem. Let $d(\pi)$ denote the minimum number of reversals to transform π into I . Then,

$$d(\pi) \geq (n + 1) - c(\pi).$$

Note: $(n + 1) - c(\pi) \geq \frac{b(\pi)}{2}$ — why ???

- For $\pi = (0, 13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11, 14)$, $c(\pi) = 10$.
 Therefore, $d(\pi) \geq 14 - 10 = 4$. So,

Question 1: are 4 reversals really necessary?

Yes. And thus that is an optimal series of reversals.

214

Further notes on genome rearrangement by reversals:

- For almost all biological instances, $d(\pi) = (n + 1) - c(\pi)$
 - also called *Sorting by Reversals* (SBR)
 - NP-hard and Max SNP-hard
 - the best-of-the-art approximation ratio 1.375 (claimed, prior to this the best is 1.5)
- An independent research problem *Breakpoint Graph Decomposition* (BGD)
 - $(n! - (n - 5)!) / n!$ instances out of $n!$: SBR = BGD
 - NP-hard and Max SNP-hard
 - the best-of-the-art approximation ratio is 1.4193

215

Genome rearrangements by transpositions:

- Formal description of the problem:
 - $\pi = (\pi_1, \dots, \pi_{i-1}, \pi_i, \dots, \pi_{j-1}, \pi_j, \dots, \pi_{k-1}, \pi_k, \dots, \pi_n)$ a permutation on $(1, 2, \dots, n)$
 - a transposition $t(i, j, k)$, $1 \leq i < j < k \leq n + 1$, on π makes π into
 $\pi' = (\pi_1, \dots, \pi_{i-1}, \boxed{\pi_j, \dots, \pi_{k-1}}, \boxed{\pi_i, \dots, \pi_{j-1}}, \pi_k, \dots, \pi_n)$
 - given any permutation, find a shortest series of transpositions that transforms π into the identity permutation $I = (1, 2, \dots, n - 1, n)$.

Questions:

- Can we always transform one permutation into identity by transpositions only?

Yes. How ???

- How to find a shortest series of transpositions ???

- The example $(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11)$:

Breakpoint:

There is one breakpoint if $\pi_{i+1} - \pi_i \neq 1$, for $0 \leq i \leq n$.

Note: different to the definition for reversal only

216

Genome rearrangements by transpositions (2):

- Idea in designing an algorithm:

Try to find a segment and a position so that, by transposing the segment to that position, we can reduce the number of breakpoints!!!

- a **strip** is a *maximal* segment containing no breakpoints inside (therefore, increasing)
- the example (13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11) contains 7 strips: (13), (1, 2, 3, 4, 5, 6, 7), (10), (9), (8), (12), (11)
- transposition segments should contain a full strip and the inserting position should be outside of any strip
 - otherwise will be creating new breakpoints (can be shown true precisely — can you ???)

Lemma Every transposition reduces the number of breakpoints $b(\pi)$ by at most 3.

So, one lower bound is $\frac{b(\pi)}{3}$.

- The example permutation (13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11) can be transformed into the identity via 4 transpositions ...

Its lower bound is $\lceil \frac{7}{3} \rceil = 3$.

Question: is it optimal ???

217

Genome rearrangements by transpositions (3):

- Breakpoint graph $G(\pi)$ of π the same as in the reversals only case:

$\pi = (0, 13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11, 14)$ and $n = 13$:
has $c(\pi) = 10$.

Lemma. For every permutation π , apply one transposition to transform it into π' . Then, $|c(\pi) - c(\pi')| = 0$ or 2.

Proof. By distinguishing the cases where the two black edges involved are in one cycle or two cycles in a maximum alternating cycle decomposition.

- In other words: a transposition can increase $c(\pi)$ by at most 2.
- Observation: $c(I) = n + 1$.

Theorem. Let $d(\pi)$ denote the minimum number of transpositions to transform π into I . Then,

$$d(\pi) \geq \frac{(n+1) - c(\pi)}{2}.$$

- For $\pi = (0, 13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11, 14)$, $c(\pi) = 10$.
Therefore, $d(\pi) \geq \frac{14-10}{2} = 2$. So,

Question: Is 4 optimal?

Sorry, we are unable to answer ...

218

Further notes on genome rearrangement by transpositions:

- Not known yet if it is NP-hard ...
- Approximation algorithms
 - using the above theorem on the lower bound, there is a 1.75-approximation
 - improved to 1.5
- There is **NO** signed genome rearrangement by transpositions ...
Why ???

You can never flip the sign of a gene by transpositions only.

219

Unsigned genome rearrangement by reversals and transpositions:

- For example permutation:
 $(13, 1, 2, 3, 4, 5, 6, 7, 10, 9, 8, 12, 11) + t(9, 12, 14) \rightarrow$
 $(13, 1, 2, 3, 4, 5, 6, 7, 12, 11, 10, 9, 8) + t(1, 2, 9) \rightarrow$
 $(1, 2, 3, 4, 5, 6, 7, 13, 12, 11, 10, 9, 8) + r(8, 13) \rightarrow$
 $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$
- It's hard to have precise definition *breakpoint*, although it is still a breakpoint if

$$|\pi_i - \pi_{i+1}| \neq 1.$$

So, $b(\pi) \geq 5$.

- Can still have the *breakpoint graph*:

$$c(\pi)$$

- Lower bounds:

- every operation decreases the number of the above special breakpoints, $b_0(\pi)$, by ≤ 3
- every operation increases $c(\pi)$ by ≤ 2
- every operation increases $c_o(\pi)$ by ≤ 2 , where $c_o(\pi)$ is the maximum number of odd cycles in any alternating cycle decomposition

Therefore,

$$d(\pi) \geq \left\{ \frac{b_0(\pi)}{3}, \frac{n+1 - c_o(\pi)}{2} \right\}.$$

- So, at least we have a 1.5-approximation algorithm ...
- Is it hard ???

220

Signed genome rearrangement by reversals and transpositions:

- Transforming signed into unsigned by:

$$+i \rightarrow (2i - 1, 2i)$$

$$-i \rightarrow (2i, 2i - 1)$$

and each of the pairs is not allowed broken

- There is a breakpoint if

$$|\pi_i - \pi_{i+1}| \neq 1.$$

In breakpoint graph, there is a black edge iff there is a breakpoint; there is a gray edge iff the numbers differ by 1 and their positions are not adjacent.

Note: different from the previous definitions

- Alternating cycles, and $c(\pi)$

- New goal: eliminating breakpoints and cycles

$$b(\mathcal{I}) = 0 \text{ and } c(\mathcal{I}) = 0$$

- Lower bounds:

- every operation decreases $b(\pi)$ by ≤ 3
- every operation decreasing $b(\pi)$ by 3 also decreases $c(\pi)$ by 1
those 3 black edges belong to a 3-cycle
- can show: every operation decreases $(b(\pi) - c(\pi))$ by ≤ 2

Therefore,

$$d(\pi) \geq \frac{b(\pi) - c(\pi)}{2}.$$

221

Signed genome rearrangement by reversals and transpositions (2):

- By carefully examination, we will be able to find a series of at most $(b(\pi) - c(\pi))$ operations which eliminate all breakpoints (and thus all cycles) in π .

Theorem. SBRT admits a 2-approximation algorithm.

- Further notes:

- believed to be NP-hard, no proof yet ...
- 2-approximation is the best guarantee so far

- One more operation — reversal transposition:

— $\pi = (\pi_1, \dots, \pi_{i-1}, \pi_i, \dots, \pi_{j-1}, \pi_j, \dots, \pi_{k-1}, \pi_k, \dots, \pi_n)$ a permutation on $(1, 2, \dots, n)$

— a reversal transposition $rt(i, j, k)$, $1 \leq i < j < k \leq n + 1$, on π makes π into

$$\pi' = (\pi_1, \dots, \pi_{i-1}, \boxed{\pi_j, \dots, \pi_{k-1}}, \boxed{\pi_{j-1}, \dots, \pi_i}, \pi_k, \dots, \pi_n)$$

- Genome rearrangement by reversals, transpositions, and reversal transpositions

- hardness open
- the best-of-the-art: 1.75-approximation algorithm

- Transposition regarded as an exchange

Another operation: double reversal $rr(i, j, k)$:

$$\pi = (\pi_1, \dots, \pi_{i-1}, \pi_i, \dots, \pi_{j-1}, \pi_j, \dots, \pi_{k-1}, \pi_k, \dots, \pi_n) \rightarrow$$

$$\pi' = (\pi_1, \dots, \pi_{i-1}, \boxed{\pi_{j-1}, \dots, \pi_i}, \boxed{\pi_k-1, \dots, \pi_j}, \pi_k, \dots, \pi_n)$$

222

Lecture 19: Genome Rearrangement

Agenda:

- Database & algorithms

Genetic/Genomic data (here genes & gene orders) \rightarrow

evolutionary distance \rightarrow

evolutionary tree (phylogeny)

Links:

by Alex Strilets

<http://alpha.hiqsoft.com/genomerr/>

by Ayman Ammoura

<http://www.cs.ualberta.ca/~ayman/bio/>

Ensembl Genome Browser

<http://www.ensembl.org/>

Arthropoda mitochondrial genomes

<http://www.ncbi.nlm.nih.gov/PMGifs/Genomes/6656.html>

Sauropsida mitochondrial genomes

<http://www.ncbi.nlm.nih.gov/PMGifs/Genomes/8457.html>

Tree of life

<http://tolweb.org/tree/phylogeny.html>

Phylogeny : Phylip programs

<http://bioweb.pasteur.fr/seqanal/phylogeny/phylip-uk.html>

223

Lecture 20: Protein Structure Prediction

Agenda:

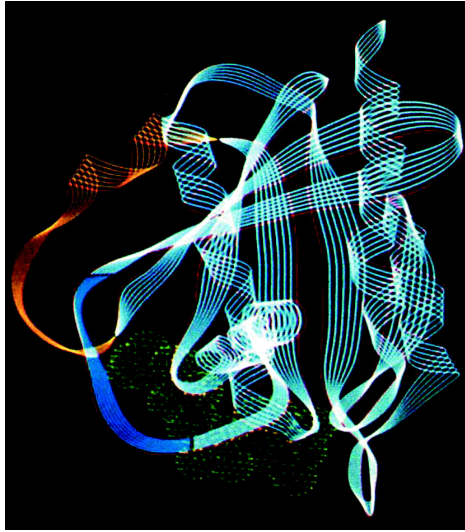
- Overview of protein structure/function prediction
- Sequence homology (BLAST, PsiBLAST)
- Structure homology (fold recognition by threading)

Reading:

- Ref[JXZ, 2002]: chapters 16-18.
- D. Baker and A. Sali. Protein structure prediction and structural genomics. *Science*. (2001) 294:5, 93–96.
- S. F. Altschul *et al.* Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*. (1997) 25, 3389 – 3402.
- Y. Xu *et al.* An efficient computational method for globally optimal threading. *Journal of Computational Biology*. (1998) 5, 597–614.

224

A protein (3D or tertiary) structure:



See www.cmbi.kun.nl/gvteach/alg/infopages/proteins.shtml for a detailed atomic decomposition.

Secondary structure involves α -helix, β -sheet, and β -turn.

225

Structure prediction:

- Big reasons for prediction:
 - a huge number of proteins (e.g., 300,000 human proteins)
 - a small number of structures been experimentally determined accurate (high resolution)
 - experimental structure determination is very expensive and slow
 - not all protein structure necessarily determined experimentally
- Roughly, 3 different approaches for prediction:
 - NMR spectroscopy and X-ray crystallography
 - comparative modeling
 - De novo structure prediction
- 4 steps in structure modeling (including comparative modeling and threading):
 - find related known structures — templates
 - compute sequence-template alignment
 - build structure model
 - assess the structure model

226

NMR and X-ray crystallography:

- They are both experimental methods using knowledge from biology, physics, chemistry, biochemistry, and mathematics
- NMR spectroscopy
 - protein purification
 - spectra generation
 - peak picking
 - peak assignment
 - structural information extraction
 - structure calculation

Requires nearly complete and accurate peak assignment, which is hard

- X-ray crystallography
 - crystal preparation
 - X-ray diffraction collection
 - structure calculation

Requires high quality crystal, which is hard too

- None of them is a high-throughput technology
- NMR can provide the structure in near physiological condition

227

Structure modeling:

- 4 steps:
 - find related known structures — templates
 - compute sequence-template alignment
 - build structure model
 - assess the structure model
- Comparative modeling:
 - templates can be found by
 1. sequence comparison methods such as PSI-BLAST
 2. structure comparison methods
 - compute a sequence-structure alignment minimizing the 'energy':
 1. using sequence alignment (homology search)
 2. threading (next lecture)
 - produce all-atom model
 1. cores, loops, side-chains
 2. approximate positions of the conserved atoms

228

De novo prediction:

- No templates, predict from scratch
- Assumption: native structure is at the global free energy minimum
- Therefore, search limited to the conformational space with low global free energy

Accuracy:

approach	req. seq. identity	model accuracy
NMR, X-ray	-	1.0Å
comparative	sequence	> 50%
	threading	> 30%
	threading	< 30%
de novo	insignificant	high error
		4-8Å

Applications:

- understand the functions/structure
- function repair, drug design
- etc.

229

Protein function prediction:

- Through structure prediction
- Indirectly through structure prediction
 - protein backbone structure prediction via NMR, X-ray
 - threading to predict function
 - sequence similarity — homology search
 - key functional region identification
 1. de novo structure prediction
 2. function motif recognition / learning

E.g., the basic EF-hand consists of two perpendicular 10 to 12 residue alpha helices with a 12-residue loop region between, forming a single calcium-binding site (helix-loop-helix). Calcium ions interact with residues contained within the loop region. Each of the 12 residues in the loop region is important for calcium coordination.

3. structure motif recognition / learning

230

Sequence homology search tool — Psi-BLAST:

- Position-Specific-Iterated BLAST
- “<http://www.ncbi.nlm.nih.gov/BLAST/>”

- One protein sequence:

>tetrahymena01: 378aa

```
MILFKLLIQ KKVNYLSRL MIARHILKQ NLAKTTPFFR FSEFNETESS
FNHNENRPQR VQKPRFKVA IILSGGVYD GSEVTEVVS MVHLNKS HVS
FQQEINTFYS FLELIKNFKT QFQNIPLNQ RCFAPNQQL HVVNHITGET
TTETRNVLVE SARIARGEVK DITQLKGEDY QAVLLPGGFG AAKNLSYAV
NGTNFTVNSE VERVLREFHS QKKPIGAMCI SPLILAKVLQ NDNLNHHGRV
DNQDTPNKCW PHSEAINAAE TLGAQHFQRN ADRFQVDFEN KIVTAPAMMF
NGFTKFSTTF DGAGHLVNI DQIVQGNSIK LDTIGKKHE NFDGERPRG
QYNRRNNRRR PRENNNENHE QHHHEQQ
```

- Items need to check:
 - Psi-BLAST vs. BLASTp
 - input protein sequence format: FASTA
 - database selection
 - score scheme
 - # iterations
 - score, Z-score, E-value
 - sequence identity
 - PSSM generating and using

231

Protein threading:

- Knowledge-based force field
 - statistics on a database of known proteins
 - building a force field
 - collecting a set of possible conformations for peptides of length L
 - measuring the energy for query protein (of length L) in each conformation
 - ranking the energy (the lowest on the top)

- Details:

- s — discrete distance between two atoms c and d
- c and d — atoms from two amino acids a and b , at distance k
- the energy associated is

$$E_k^{ac,bd}(s) = -kT \ln \left[\frac{1}{1 + \sigma m_{ab}} + \frac{\sigma m_{ab}}{1 + \sigma m_{ab}} \frac{g_k^{ac,bd}(s)}{g_k^{c,d}(s)} \right], \text{ where}$$

m_{ab} — relative frequency of amino acid a and b at distance k

$g_k^{ac,bd}(s)$ — relative frequency of atoms c and d at distance s

$g_k^{c,d}(s)$ — relative frequency of atoms c and d at distance s averaged over all amino acid pairs

σ — weight of observation pair a and b

k — Boltzmann's constant

$T = 293$

- energy in one conformation is:

$$E(S, C) = \sum_{ij} E_k^{ac,bd}(s_{ij}), \text{ where}$$

S — query sequence

C — conformation

i, j — indices of atoms c and d , respectively

232

Protein threading (2):

- Lattice Monte Carlo model
 - global minimum (native structure) state is known in the model
 - global minimum state on the potential surface
 - folding starts from random-coil state to a random semi-compact globule
 - folding from semi-globule state to the native fold
 - complexity: $10^{16} \rightarrow 10^{10} \rightarrow 10^3$
 - calculate the energy for every possible state and output the minimum

233

Protein threading (3):

- Comparative protein modeling (Modeller):
 - compose a database of proteins with known 3D structure
 - align the query sequence with “related” known 3D structures
 - derive the distance and dihedral angle constraints on the query sequence
 - combine these constraints and energy terms into an objective function
 - optimize the objective function, for example by
 1. non-linear optimization
 2. molecular dynamics + simulated annealing
 - evaluate the fold
- Two steps of details
 - constraint deriving:
 1. conditional probability summarized from the database
 2. using least-square to fit into the histogram
 - optimization:
 1. typically there are thousands of constraints
 2. can model atoms from the sidechains as well
- Note: alignment is critical to the success of prediction

234

Protein threading (4):

- Template/Sequence profile (GenThreader):
 - database consists of unique protein chains (templates) from PDB
 - run each template on a much larger database to construct a profile
 - pairwise align query sequence with a profile
 - evaluate the alignment using potentials:
 1. pairwise potential:

$$E_k^{acbd}(s) = -kT \ln \left[\frac{1}{1 + \sigma m_{ab}} + \frac{\sigma m_{ab}}{1 + \sigma m_{ab}} \frac{g_k^{acbd}(s)}{g_k^{cd}(s)} \right]$$
 (you have seen this before)
 2. solvation potential:

$$E_{sd}^a(r) = -RT \ln \left(\frac{f^a(r)}{f(r)} \right), \text{ where}$$
 - r — degree of residue a burial
 - $f^a(r)$ — frequency of occurrence amino acid a with burial r
 - $f(r)$ — frequency of occurrence all amino acids with burial r
 - evaluation through a neural network:
 1. input layer: pairwise energy, solvation energy, alignment score, alignment length, template length, query sequence length
 2. hidden layer
 3. output layer: templates related, templates unrelated

235

Lecture 21: Protein Fold Recognition

Agenda:

- Protein threading — PROSPECT

Reading:

- Ref[JXZ, 2002]: chapters 17-18.
- Y. Xu *et al.* An efficient computational method for globally optimal threading. *Journal of Computational Biology*. (1998) 5, 597–614.
- Y. Xu & D. Xu. Protein threading using PROSPECT: design and evaluation. *Proteins: Structure, Function, and Genetics*. (2000) 40, 343–354.

236

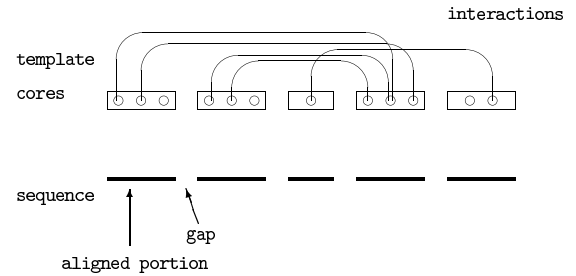
Protein threading (5) — PROSPECT:

- Main idea: divide-and-conquer
- Details:
 - again, thread query protein into template folds
 - find the fold achieving minimum free energy
 - computational recognition of native fold
 - can be used to construct the structure
 - mainly focus on backbone structure for function prediction
- Factors: — what are counted into the free energy ?
 - singleton fitness (into the local environment)
 - pairwise interaction (non-adjacent close residues)
 - gaps
- How to compute:
 - sequence-to-template alignment
 - assume additive sum of energy terms
 - no gaps inside cores
 - divide-and-conquer

237

Protein threading — PROSPECT (2):

- A schematic view:



Here, (secondary) core = α -helix or β -sheet.

- The goal:
To compute an optimal partition of the query sequence such that the corresponding sequence-to-template alignment achieves the minimum free energy.

238

Protein threading — PROSPECT (3):

- Singleton fitness:
Represents a residue's local preference of the **secondary structure** (helix, sheet, loop) and the **solvent environment** (the extent of exposure to solvent).

- Energy term:

$$e_{\text{single}}(aa, ss, sol) = -kT \log \left(\frac{N(aa, ss, sol)}{N_E(aa, ss, sol)} \right), \text{ where}$$

aa — amino acid type

ss — secondary structure type

sol — extent of exposure to solvent

k — Boltzmann's constant

T — temperature 295

$N(aa, ss, sol)$ — # amino acid of type aa in secondary structure ss with extent sol of exposure to solvent, counted from the database

$N_E(aa, ss, sol)$ — expected # amino acid of type aa in secondary structure ss with extent sol of exposure to solvent, counted from the database:

$$N_E(aa, ss, sol) = \frac{N(aa) \times N(ss) \times N(sol)}{N^2}$$

There are 3 types of solvent accessibility: buried ($< 9.5\%$), intermediate ($< 48.6\%$), and exposed ($\geq 48.6\%$) — calculated by ACCESS

239

Protein threading — PROSPECT (4):

- Pairwise interaction:
Measures the pairwise potential between spatially close non-adjacent residues.

- Energy term:

$$e_{\text{pair}}(aa_1, aa_2) = -kT \log \left(\frac{M(aa_1, aa_2)}{M_E(aa_1, aa_2)} \right), \text{ where}$$

aa_1, aa_2 — amino acid types

$M(aa_1, aa_2)$ — # of pairs of amino acid types aa_1 and aa_2 in the database that interact

$M_E(aa_1, aa_2)$ — expected # of pairs of amino acid types aa_1 and aa_2 in the database that interact:

$$M_E(aa_1, aa_2) = \frac{M(aa_1) \times M(aa_2)}{M}$$

- Spatially close:

- for two non-adjacent residues
- the distance between their β carbon (C_β) atoms is $\leq D$
- D usually ranges from 7 Å to 15 Å

240

Protein threading — PROSPECT (5):

- Mutation energy:
Describes the compatibility of substituting one amino acid type by another during the evolution.
- Energy term:
Use PAM250

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

241

Protein threading — PROSPECT (6):

- Gap penalty
Gap length is the length difference between a loop in the template and its aligned (loop) region in the query sequence.

- Affine gap penalty:

$$e_{\text{gap}} = 10.8 + 0.6 \times (\text{gap.length} - 1).$$

- Total energy:

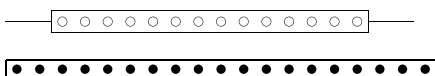
$$E_{\text{total}} = c_1 \times E_{\text{single}} + c_2 \times E_{\text{pair}} + c_3 \times E_{\text{mutation}} + c_4 \times E_{\text{gap}}$$

Need to learn coefficients c_i : How ???

242

Protein threading — PROSPECT (7):

- The threading algorithm:
The base step — how to align a subsequence to a core
- Cutting a core out of the template creates a set of half-links
- Links represent the pairwise interaction
- For one half-link, need to get the information:
the amino acid that the half-link should connect to
- When computing the alignment, impose a constraint that the half-link and its the other part can be connected together — meaning that both half-links should be *assigned* a same amino acid
- The reason we only need to record amino acid type (not its residual position) is:
Positional constraints are counted into the gap penalty ...
- Gaps are restricted to the loop area (and the tails of cores)
- Try all (in total linear number of) the possibilities and get the optimal



243

Protein threading — PROSPECT (8):

- The threading algorithm:
Analysis of the base step, suppose
 - core (t_p, t_q)
 - half-link set X
 - subsequence $(s_i, s_j) \rightarrow j - i \geq q - p$
 For every feasible assignment A for half-link set X , try all the possible cutting point $k : i \leq k \leq j - (q - p)$

$$\Theta((j - i) - (q - p))(q - p)$$
- The general recurrence:
When cutting a link, one half is open
— can be assigned with any amino acid and the pairwise interaction energy is calculated;
The other half is closed
— again can be assigned any amino acid, but the energy is $+\infty$ if it doesn't agree with the residue that the link links to
Therefore, these force the link to link two residues with pairwise energy counted exactly once.
- Complexity:
 - need to examine every possible cutting point — divide
 - need to create the assignments for links cut during the divide stage
 - C — maximum # links being cut over all possible positions
 $\Theta(Mn20^{1.5C} + mn20^C)$

244

Protein threading — PROSPECT (9):

- Overall running complexity $\Theta(Mn20^{1.5C} + mn20^C)$, so far ...
- Careful dividing strategy to minimize C — preprocessing templates
- Another observation for speed up:
No gaps inside the cores
- If two links linking to t_p and t_r within a same core, then they may produce up to $\Theta((j-i))$, instead of $\Theta(20^2)$, different pairs of half-links.
→ merge them into one (half-)link, if both are open (or closed).
- Need careful dividing strategy to minimize this new parameter C' for every template ...
Can be done by Dynamic Programming (exponential time in the template length)
 - preprocessing
 - once get the dividing strategy, use it forever in the future threading
 - for most of the templates, $C' \leq 4$ while some might be as high as 8
 - C' increases with distance threshold D increases
- Overall running time now: $\Theta(Mn \times n^{1.5C'} + mn n^{C'})$

245

Protein threading — PROSPECT (10):

- Database construction:
 - collect more than 2,000 templates from FSSP/PDB (also SCOP library)
 - secondary structure information
 - solvent accessibility information
 - preprocessing to compute the optimal dividing strategy for every template
- Accepts additional information to assist threading
 - PSSM
 - secondary structure preference
 - solvent accessibility preference
- Execution details:
 - excluding pairwise interaction to speed up first-round threading
then enter the second round detail threading using pairwise interaction
 - output sequence-to-template alignment
 - assess the alignment through a neural network (confidence)
 - alignment score, fitness score for every energy term, Z-score, compact factor, structure construction using template structure(s)

246

Lecture 22: Protein Function Prediction

Agenda:

- More function prediction approach introduction

Reading:

- Ref[JXZ, 2002]: chapters 17-18.
- R. Lathrop *et al.* Global optimum protein threading with gapped alignment and empirical pair score functions. *Journal of Molecular Biology*. (1996) 255, 641–665.
- M. Deng *et al.* Prediction of protein function using protein-protein interaction data. *IEEE CSB'02*. Pages 197–206.

247

Lecture 22: Protein Function Prediction

Protein threading (6):

- Multiple sequence alignment:
 - a database of candidate structures (templates)
 - search for every template the similar structures in PDB (pairwise structure alignment)
 - compute a multiple structure alignment out of these pairwise alignments
 - use the multiple alignment as a profile, compute the sequence-structure alignment
- Some steps of details:
 - pairwise structure alignment
 - use the *center star* method to compose the multiple alignment (approximate)
 - threading (sequence-structure alignment):

$$\text{score}(a, c_i) = \sum_{k=1}^N s(a, c_i^k), \text{ where}$$

N — number of structures in the structure profile
 c_i — the i th column in the structure profile
 c_i^k — the amino acid in the k th structure in the i th column in the structure profile
 $s(\cdot, \cdot)$ — scoring scheme (PAM250)

248

Protein threading (7):

- Branch-and-bound:
 - a database of templates, each consists of
 1. core segments (secondary structures)
 2. pairwise interaction
 3. solvent accessibility
 - setting up for each core segment the region on the query sequence it may map to
 1. no gap in the core segments
 2. affine or whatever gap penalty
 3. score function
 - estimate for every core segment the minimal possible energy
 - sum the core segment energy up to be a lower bound of the search space
 - choose the mapping interval of the core segment reaching the minimal energy to split into 3 subsets
 - ending at a subset in which there is only one possible assignment
- The achieving assignment reaches the minimal energy
- Score function:
 - fitness energy for every amino acid: local environment
 - pairwise interaction
- Note: branch-and-bound algorithm nature — exponential in the worst case

249

Protein threading (8) — more models:

- Hierarchical threading
- Hidden Markov models for homology detection
- Machine learning approach(es)
by Brett Poulin
- Markov random field using pairwise/complex interaction data
by John Shillington

250

Lecture 23: Phylogeny Reconstruction

Agenda:

- Maximum parsimony
- Perfect phylogeny
- Neighbor-joining

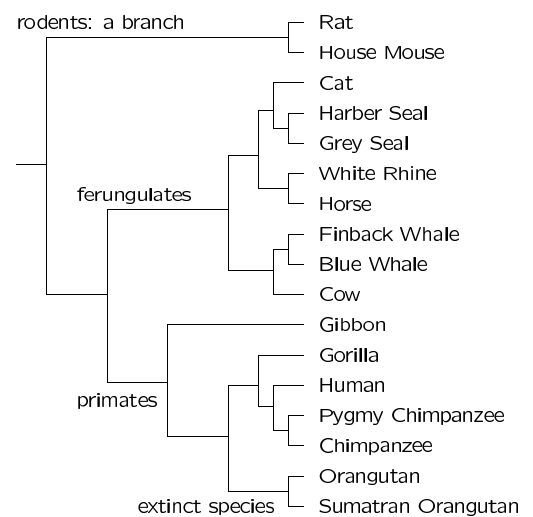
Reading:

- Ref[Gusfield, 1997]: pages 447–479
- Ref[JXZ, 2002]: chapter 5.

251

Lecture 23: Phylogeny Reconstruction

A phylogeny (also called evolutionary tree):



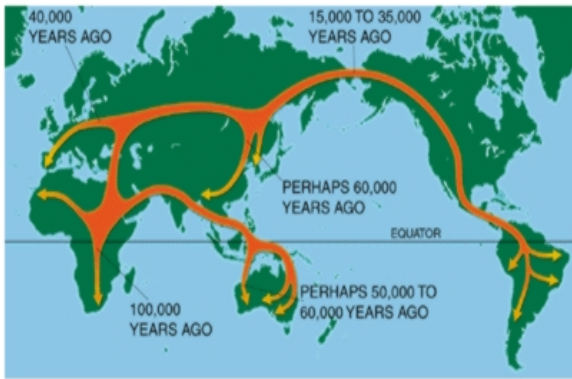
- What data?

Mitochondrial genomes

- Which method?

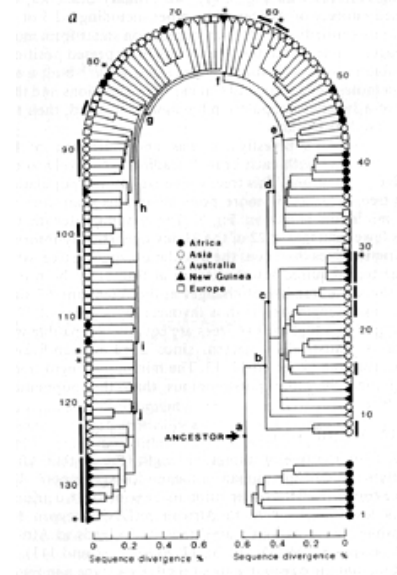
252

An example - Out of Africa Theory:



253

Phylogenetic analysis from Cann *et al.*, 1987, *Nature*, based on restriction mapping of mitochondrial DNAs (mtDNAs) of 147 people.

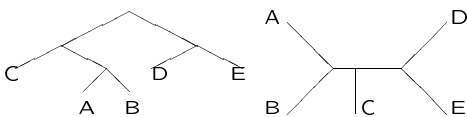


Similar results have been obtained using polymorphisms on Y chromosomes.

254

Factors in phylogeny reconstruction:

- Data:
 - character data — morphological features, aligned sequence characters, polymorphic genetic features
 - molecular sequence — DNA, RNA, protein
 - distance — estimated (evolutionary) distance between TAXA based on some model of evolution
- Methods:
 - parsimony, compatibility, distance-based, maximum likelihood, quartet, phylogenetic root
- Rooted vs unrooted phylogeny:



- Weighted vs unweighted phylogeny:
 - Weight gives the evolutionary distance between two TAXA

255

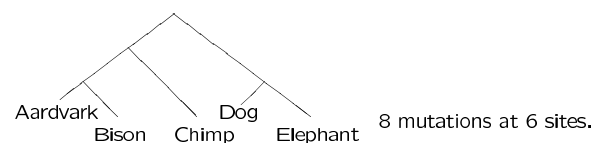
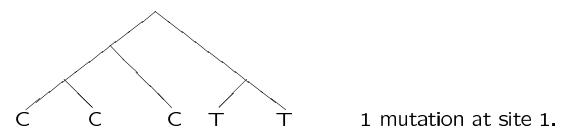
Parsimony:

Input usually character data:

species	site1	2	3	4	5	6
Aardvark	C	A	G	G	T	A
Bison	C	A	G	A	C	A
Chimp	C	G	G	G	T	A
Dog	T	G	C	A	C	T
Elephant	T	G	C	G	T	A

- How to evaluate a given tree? — minimum number of mutations
- How to search the tree space?

Small Parsimony Problem: given the phylogeny with TAXA arranged at the leaves, compute the internal nodes to minimize the total number of mutations.



256

Fitch's algorithm for Small Parsimony:

1. Goal: to minimize the total number of mutations.
2. Consider sites separately.
3. For each internal node i and state s , $c(i, s)$ is the minimum number of changes required for the subtree rooted at i , if i is given the state s ;
4. Recurrence relation:
If j and k are the children of i ,

$$c(i, s) = \min_{p, q \in \{A, C, G, T\}} (c(j, p) + c(k, q) + s(s, p) + s(s, q))$$

$$= \min_{p \in \{A, C, G, T\}} (c(j, p) + s(s, p)) + \min_{q \in \{A, C, G, T\}} (c(k, q) + s(s, q))$$
5. Traverse the tree in post order or bottom up to compute $c(\cdot, \cdot)$.

Time: $O(|T| \cdot |\Sigma|)$

257

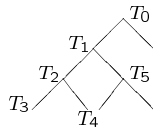
The (Large) Parsimony Problem:

- We know how to evaluate a given tree ...
- How to find a tree with the minimum cost?
For 20 leaves, there are 10^{21} binary trees, 10^{24} rooted trees ...
- NP-hard

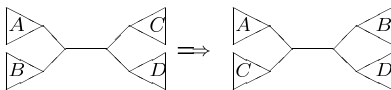
258

The (Large) Parsimony Problem (2):

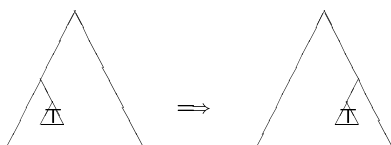
- Heuristics and approximations:
 1. branch-and-bound:
 - consider trees of increasing size;
 - abort an extension if cost already exceeds current best.



2. local search (neighborhoods)
 - nearest neighbor interchange (NNI)



- subtree transfer



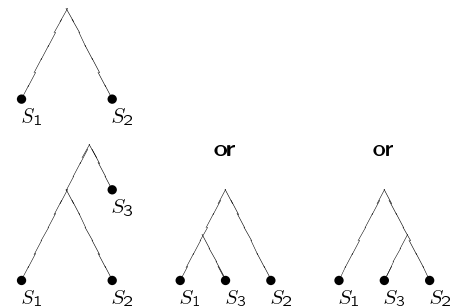
259

The (Large) Parsimony Problem (3): A greedy method

- Species are represented by DNA sequences
- Assume a given multiple alignment \rightarrow character data

PHYLP – DNAPARS

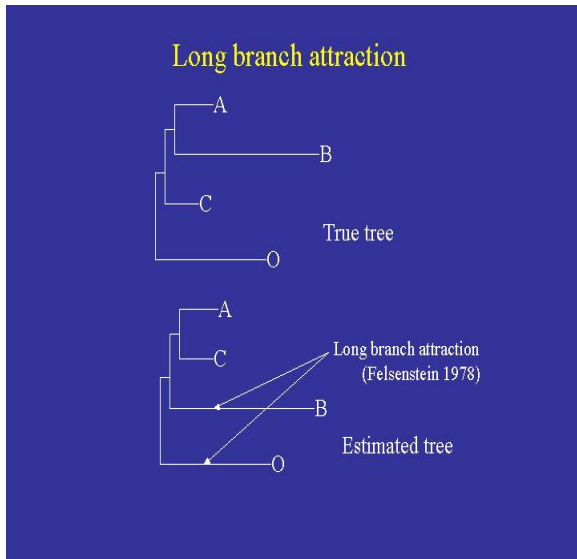
1. Arrange the sequences in some order.
2. Construct a phylogeny for the first two sequences.
3. **repeat** until all sequences are added:
Add the sequence at the head of the order to the current phylogeny to achieve parsimony.



...

260

Problem with the parsimony method: statistical inconsistency

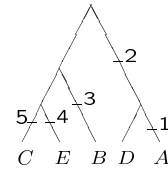


261

Perfect phylogeny:

- The parsimony problem taking **binary** state characters
- Requires at most 1 change for each character/site.

	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	0
C	0	0	0	0	1
D	0	1	0	0	0
E	0	0	0	1	0

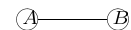


Definition: For any character i , let O_i be the set of taxa with state 1 at character i .

Theorem: A perfect phylogeny exists for the input, iff for any $i \neq j$, either $O_i \cap O_j = \emptyset$, or $O_i \subseteq O_j$, or $O_j \subseteq O_i$.

Proof. "only if": Trivial.

"if": For any character i incurring a state change, define a bipartition (A, B) , and draw an edge



Repeat this process for other characters. The property of the data guarantees that sets A and B will be subdivided by further bipartitions. (If there are any unresolved nodes, resolve them arbitrarily.)

Generalizations To more than 2 states, see [Gusfield, 1997].

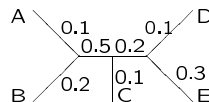
262

Distance-based phylogeny reconstruction (1):

- The problem: for every pair of taxa i and j , define a distance $d(i, j)$. Construct an edge weighted tree such that the path joining i and j has weight $d(i, j)$.

- An example,

	A	B	C	D	E
A	0				
B	0.3	0			
C	0.7	0.8	0		
D	1.1	1.2	0.6	0	
E	0.9	1.0	0.4	0.4	0



- Getting distance — Jukes/Cantor's distance model
 1. input data: ungapped alignment of DNA sequences
 2. how: treat each site as an independent random variable
 3. assumption: every nucleotide has equal probability of mutating one of the three other nucleotides
 4. parameter: common point mutation rate m — expected number of mutations per unit time. Hence, the expected number of mutations within time Δt is $m\Delta t$.
 5. Jukes/Cantor's model is Markovian:

$$1 - m\Delta t \quad \text{A} \quad \text{G} \quad 1 - m\Delta t$$

$$\frac{m}{3}\Delta t \quad \frac{m}{3}\Delta t$$

$$1 - m\Delta t \quad \text{C} \quad \text{T} \quad 1 - m\Delta t$$

263

Distance-based phylogeny reconstruction (2):

- Jukes/Cantor's distance model:
 - assume $m\Delta t \leq 0.75$, starting from state i , the probability of ending up in a state $j \neq i$ is:

$$\Pr(q_j|q_i, t) = \frac{1}{4} \left(1 - \left(1 - \frac{4m\Delta t}{3} \right)^{\frac{1}{4}} \right).$$

Letting $\Delta t \rightarrow 0$,

$$\Pr(q_j|q_i, t) = \frac{1}{4} \left(1 - e^{-\frac{4}{3}mt} \right).$$

- hence, the probability of at least 1 mutation is

$$\Pr(\text{mutation}|t) = \frac{3}{4} \left(1 - e^{-\frac{4}{3}mt} \right),$$

since the sites are independent and identical processes (iid).

- fraction of sites changed (letting $f = \frac{3}{4} \left(1 - e^{-\frac{4}{3}mt} \right)$):

$$mt = -\frac{3}{4} \log \left(1 - \frac{4}{3}f \right) \quad \text{--- mutation distance}$$

- Kimura 2-parameter:
Replace m with transition and transversion rates.
- HKY/Felsenstein:
Unequal probability for individual nucleotide.

264

Distance-based phylogeny reconstruction (3):

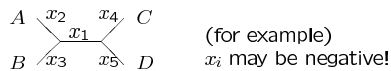
- Distance data $d(i, j)$ is **additive** if there is an edge weighted tree fully agreeing with the distances.
- (Buneman) Four Point Condition:
For any taxa A, B, C, D ,

$$\max \left\{ \begin{array}{l} d(A, B) + d(C, D) \\ d(A, C) + d(B, D) \\ d(A, D) + d(B, C) \end{array} \right\} \\ = \text{median} \left\{ \begin{array}{l} d(A, B) + d(C, D) \\ d(A, C) + d(B, D) \\ d(A, D) + d(B, C) \end{array} \right\}$$

Theorem A distance data is additive if and only if it satisfies the four point condition.

Proof.

- "only if": Trivial.
- "if": For any taxa A, B, C, D , the condition uniquely determines,



$$\left. \begin{array}{l} 2x_1 = d(A, C) + d(B, D) - d(A, B) - d(C, D), \\ x_2 + x_3 = d(A, B), \\ x_4 + x_5 = d(C, D), \\ x_1 + x_2 + x_4 = d(A, C), \\ x_1 + x_3 + x_5 = d(B, D), \\ x_1 + x_2 + x_5 = d(A, D), \\ x_1 + x_3 + x_4 = d(B, C). \end{array} \right\} \text{redundant by 4 point condition}$$

265

Distance-based phylogeny reconstruction (4):

- The Theorem implies an $O(n^4)$ time algorithm. (How?)
- But, there are $O(n^2)$ time ones, such as **Neighbor-Joining (NJ)** to be discussed.
- What if data is **not** additive / perfect?
 - Edwards & Cavalli-Sforza try to find a weighted tree to minimize

$$\sum_{i,j} (D_{ij} - d_{ij})^2,$$

D_{ij} — observed distance,
 d_{ij} — implied by tree.

- Fitch & Margoliash try to minimize

$$\sum_{i,j} \frac{(D_{ij} - d_{ij})^2}{D_{ij}^2}.$$

266

Distance-based phylogeny reconstruction (4)

— Neighbor-Joining (NJ) by Saitou & Nei, 1987:

Given $d(i, j), 1 \leq i, j \leq n$.

- For each taxa i , compute net divergence

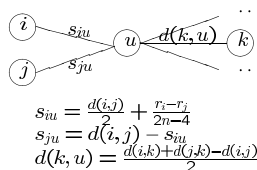
$$r_i = \sum_{j=1}^n d(i, j).$$

- Compute rate-corrected distance

$$M_{ij} = d(i, j) - \frac{r_i + r_j}{n-2}.$$

- Let i and j be the taxa minimizing M_{ij} .

Introduce a new taxa u , such that



- If more than 2 edges unweighted, go to STEP 2.
Otherwise let $s_{ku} = d(k, u)$.

Notes on NJ:

- Perfect on additive distance data.
- Very popular heuristic in practice.
- Theoretical analysis by K. Atteson.

267

Lecture 24: Phylogeny Reconstruction

Agenda:

- Maximum likelihood
- Quartet and quartet-based

Reading:

- Ref[JXZ, 2003]: pages 111 – 134

268

Maximum Likelihood Methods:

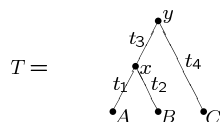
Find a tree with maximum probability of generating data (sequences).

- Model:

A branching process describing how taxa/species diverge over evolutionary time, and a sampling model (ignored).

$$\max_{M: \text{Jukes/Cantor, Kimura, HKY}} L(T|S, M) = \Pr(S|T, M) \quad (\text{Bayesian})$$

- For example,



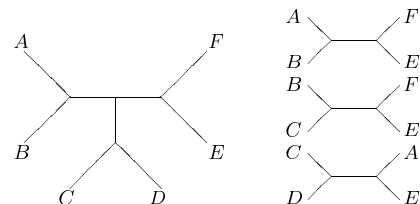
- t_i : mutation distance ($m\Delta t$)
- $L = \sum_x \sum_y P(y) \cdot P(x|y, t_3) \cdot P(A|x, t_1) \cdot P(B|x, t_2) \cdot P(C|y, t_4)$
- In a stochastic process, $P(y)$ is the equilibrium probability of y .
- The sum can be made nested.

- Maximum Likelihood gives confidence level ... but time consuming.

269

Quartet-based phylogeny reconstruction (1)

- A **quartet** q is a set of 4 distinct species in a tree T ; the subtree of T linking the species in q is the **quartet** topology for q relative to T .



Assuming bifurcating events ...

- Each tree has a unique set of associated quartet topologies (Buneman, 1971)
- Reconstruct tree by inferring topologies of **all** possible quartets and then combining these topologies.

270

Quartet-based phylogeny reconstruction (2)

Combining inferred quartet topologies is not easy in practice.

1. $O(n^4)$ quartets (huge data redundancy).
2. All existing phylogeny reconstruction methods sometimes infer incorrect quartet topologies (quartet errors).
3. Quartet recombination methods are usually sensitive to quartet errors.
4. It is NP-hard to find the largest set of compatible quartet topologies, even if the input set of quartet topologies is complete. [Steel, 1992; Berry *et al.*, 1999]
5. Heuristics such as quartet puzzling [SH96] and Q^* [BG97], etc. don't have guaranteed performance.

An alternative approach: Look for efficient algorithms that reconstruct the tree correctly in presence of bounded number of quartet errors in particular regions of the tree.

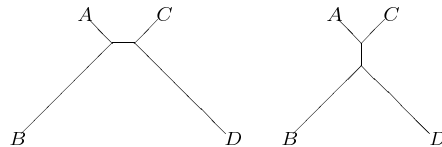
Such algorithms essentially “clean” errors in a given set of quartet topologies, and are hence called quartet cleaning algorithms.

271

Quartet-based phylogeny reconstruction (3)

A motivation: Felsenstein Zone

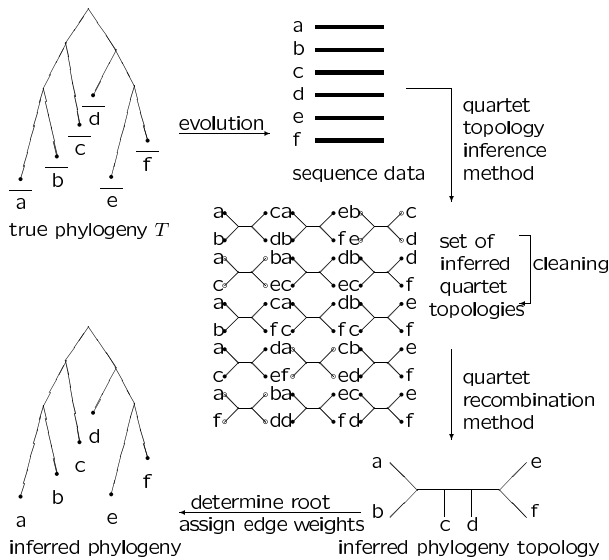
Maximum parsimony is one of the most popular methods in phylogeny reconstruction. But it is known to be statistically inconsistent.



However, when the tree to be reconstructed is large, we can afford to infer a few (strange shaped) quartet topologies incorrectly. Data redundancy allow us to clean these errors.

272

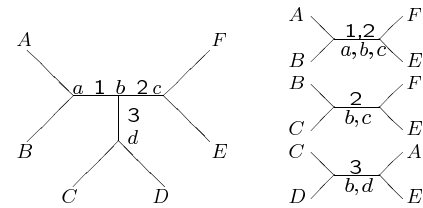
Quartet-based phylogeny reconstruction (4)



273

Quartet cleaning (1)

- Quartet error bounds may be stated relative to quartet errors across either vertices or edges:



Definition: A quartet error q is across a vertex (an edge) in a tree T if the joining path of the quartet topology associated with q in T includes that vertex (edge).

- Quartet error bounds may either be **local** (hold in individual parts of the tree) or **global** (hold in all parts of the tree).
- 4 types of cleaning algorithms: $\{\text{global, local}\} + \{\text{edge, vertex}\}$ cleaning
- Local quartet cleaning preferable —
 - all parts of tree may not satisfy a quartet error bound
 - can still reconstruct part of the tree

274

Quartet cleaning (2)

1. Global Edge Cleaning

- Every edge (A, B) has at most $\frac{(|A|-1)(|B|-1)}{2}$ quartet errors.
- $O(n^4)$ time.
- Error bound is optimal.

2. Local Edge Cleaning

- Recovers an edge (A, B) if the number of quartet errors across (A, B) is at most $\frac{(|A|-1)(|B|-1)}{2}$.
- $O(n^5)$ time.
- Unresolved tree.

3. Hyper-Cleaning(m)

- Returns all bipartition (A, B) with

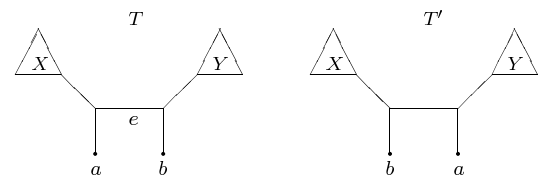
$$|Q_{(A,B)} - Q| \leq m \frac{(|A|-1)(|B|-1)}{2}.$$

- The bipartitions may not be all compatible.
- $O(n^7 \cdot f(m))$ time.

275

Quartet cleaning (3) — the limit

- Consider the following two trees T and T' .



- Observe that Q_T and $Q_{T'}$ differ by quartet topologies of the form $ax|by$, where $x \in X$ and $y \in Y$. Hence,

$$|Q_T - Q_{T'}| = (|X| - 1)(|Y| - 1).$$

- Therefore, no algorithm can detect more than $\frac{(|X|-1)(|Y|-1)}{2}$ errors across edge e .

276

Quartet cleaning (4) — local edge cleaning

Theorem There exists an $O(n^5)$ time algorithm that correctly recovers all edges e of the tree satisfying

$$|Q_{(A,B)} - Q| \leq \frac{(|A_e| - 1)(|B_e| - 1)}{2}.$$

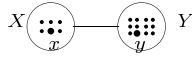
- Proof.

Let $S = \{s_1, s_2, \dots, s_n\}$, $S_k = \{s_1, s_2, \dots, s_k\}$, and Q_k the subset of Q induced by S_k . The idea is to recursively compute sets:

$BP_{xy}(Q_k) = \{(X, Y) | x \in X, y \in Y, \text{ and there is at most one quartet error across } (X, Y) \text{ involving both } x \text{ and } y\}$.

$BP_x(Q_k) = \{(X, Y) | x \in X \text{ and there are at most } \frac{(|Y|-1)}{2} \text{ quartet errors across } (X, Y) \text{ involving } x\}$.

$BP(Q_k) = \{(X, Y) | \text{there are at most } \frac{(|X|-1)(|Y|-1)}{2} \text{ quartet errors across } (X, Y)\}$.



Observe that the algorithm may leave some of edges of the tree unresolved.

277

Quartet cleaning (5) — hypercleaning

- The local edge cleaning algorithm returns the set of bipartitions (A, B) incurring at most $\frac{(|A|-1)(|B|-1)}{2}$ quartet errors in Q . These bipartitions are compatible with each other.
- What about bipartitions incurring more than $\frac{(|A|-1)(|B|-1)}{2}$ quartet errors?

Theorem There exists an $O(n^7 f(m))$ time algorithm that computes all bipartitions (A, B) satisfying:

$$|Q_{(A,B)} - Q| \leq m \frac{(|A| - 1)(|B| - 1)}{2}.$$

- These bipartitions may not be compatible with each other!
- The maximum subset of compatible quartets (hard).
- Greedy searching.

- $f(m) = 4m^2(1 + 2m)^{4m}$ (in practice, $m \leq 5$).

278

Quartet cleaning (6) — global edge cleaning

1. Quartet error bound must hold for every edge in the tree.
2. For each edge inducing a bipartition (A, B) of S , we state the bound relative to (A, B) .

Theorem There exists a polynomial time algorithm with bound $\alpha \cdot |A| \cdot |B|$ ($\alpha \leq \frac{1}{2}$).

Theorem The following algorithm computes the correct tree if every edge of the tree has at most $\frac{(|A|-1)(|B|-1)}{2}$ quartet errors in Q .

The algorithm can be implemented to run in $O(n^4)$ time, i.e., linear time.

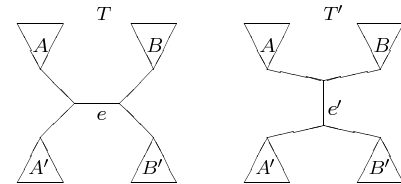
The error bound is optimal.

- Algorithm Global-Clean(S, Q):

1. Let $R := S$.
2. For every pair of rooted subtrees T_1 and T_2 in R
 - (a) Let A denote the leaf labels of T_1 and T_2 .
 - (b) If $|Q(A, S - A) - Q| < \frac{(|A|-1)(|S-A|-1)}{2}$ then
 - i. Create a new tree T' with T_1 and T_2 as its subtrees.
 - ii. Let $R := R - \{T_1, T_2\} \cup \{T'\}$.
3. Repeat step 2 until $|R| = 3$.
4. Connect the three subtrees in R and output the resulting (unrooted) tree.

279

Quartet cleaning (7) — global edge cleaning (cont'd)



- Suppose the algorithm incorrectly joins subtrees T_1 and T_2 with label sets A and B . This implies that in the true tree T , there is an edge e with bipartition $(A \cup A', B \cup B')$, $|A'| \geq 1$, $|B'| \geq 1$, which separates T_1 and T_2 in T . Similarly, there is an edge e' in the reconstructed tree with bipartition $(A \cup B, A' \cup B')$. Let $Q_1 = Q(A \cup A', B \cup B')$ and $Q_2 = Q(A \cup B, A' \cup B')$ denote the sets of quartets across e and e' , respectively.

- Consider the symmetric difference of Q_1 and Q_2 — $|Q_1 \oplus Q_2|$. Observe the following:

- $|Q_1 \oplus Q_2| = |A||B||A'||B'|$.
- $|Q_1 \oplus Q_2| \leq |Q_1 - Q| + |Q_2 - Q|$.

- As each edge in T satisfies the quartet-error bound,

$$|Q_1 - Q| < \frac{((|A| + |A'|) - 1)((|B| + |B'|) - 1)}{2}.$$

However, as T_1 and T_2 were joined by the algorithm,

$$|Q_2 - Q| < \frac{((|A| + |B|) - 1)((|A'| + |B'|) - 1)}{2}.$$

280

Quartet cleaning (8) — global edge cleaning (cont'd)

- The above two items imply

$$|A||B||A'||B'| < \frac{(|A|+|A'|)(|B|+|B'|)-1}{2} + \frac{(|A|+|B|)(|A'|+|B'|)-1}{2},$$

which is false for any $|A|, |B|, |A'|, |B'| \geq 1$.

- Error bound $\frac{(|A|-1)(|B|-1)}{2}$ is optimal.
- Intuitions behind the Proof:
 - Trees “separated” by minimum number of differences in associated sets of quartet topologies.
 - If quartet error bounds hold relative to true tree, cannot accumulate enough quartet topologies to reconstruct different tree under the cleaning algorithm.

281

Quartet cleaning (9) — simulation study

- Quartet cleaning is computationally expensive. Is it really worth? If so, under what assumption?
- Two questions:
 - How often do quartet errors occur?
 - How often can quartet errors be cleaned, *i.e.*, how often is the number of quartet errors less than the bounds required by known algorithms?
- Examine quartet errors in simulated datasets:
 - Approach: Generate phylogeny; generate DNA sequence dataset for phylogeny; infer quartet topologies relative to sequence dataset; determine quartet errors relative to phylogeny.
 - Vary evolutionary rates on edges of phylogenies and lengths of sequences in dataset.
 - Quartets inferred using 3 popular methods: Maximum Parsimony, Neighbor Joining, and Ordinal Quartet [Kearney, 1997].

282

Quartet cleaning (10) — open problems

- Improve the time efficiency of local edge cleaning and hypercleaning algorithms.
- Combine cleaning with maximum likelihood reconstruction of quartet topologies.
- Test the algorithms on real datasets.
- Various ways of extracting a tree from the bipartitions found by hypercleaning.
- Extend the work to *incomplete* set of quartet topologies.

283

Lecture 25: Phylogeny Reconstruction

Agenda:

- Phylogenetic root — a graph theory based approach

Reading:

- P.E. Kearney *et al.* Tree powers. *Journal of Algorithms*, 29:111–131, 1998.
- G.-H. Lin *et al.* Phylogenetic k -root and Steiner k -root. ISAAC 2000, LNCS 1969, pages 539–551, 2000.
- Z.-Z. Chen *et al.* Computing phylogenetic roots with bounded degrees and errors. WADS 2001, LNCS 2125, pages 377–388, 2001.

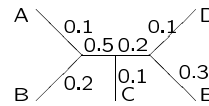
284

Impacts:

- The estimated distances are not accurate
- Rather than using exact distance, use thresholds
 - two species are *close* if their estimated distance is within some threshold t
 - in the phylogeny, their distance (the number of edges) is within some bound k

285

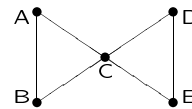
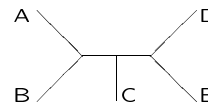
The supposed phylogeny:



One approach to reconstruct:

Step 1. distance estimation

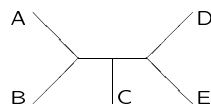
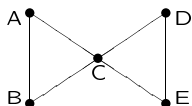
	A	B	C	D	E
A	0				
B	0.3	0			
C	0.7	0.8	0		
D	1.1	1.2	0.6	0	
E	0.9	1.0	0.4	0.4	0

Step 2. closeness graph (threshold $t = 0.8$)Step 3. phylogeny reconstruction (bound $k = 3$)

286

The phylogenetic k th root problem (PR_k):

- Given: a graph G (simple, undirected) and an integer k
 - vertices represent species
 - adjacency represent evolutionary closeness
- Output: a tree T (unrooted)
 - leaf set of $T =$ vertex set of G
 - vertices are adjacent in G **if and only if** they are connected by at most k edges in T
 - internal nodes in T have degree at least 3



287

State-of-the-art (PR_k):

- Solvable in polynomial time for $k = 2, 3, 4$ (2000)
- Remains unknown for $k \geq 5$
- Special cases: The PR_k with constraint that the maximum degree bounded by Δ is denoted by ΔPR_k
 - solvable in linear time (2001)
- For today:
 - an algorithm for PR_3
 - an algorithm for 3PR_3

288

Related Work:

- Graph power (easy) and its inverse (hard, 1994)
- Tree power (easy) and its inverse (easy, 1995, 1998)
- The optimization problems:

One graph might not have a phylogenetic (tree) root. Compute a tree such that its k th phylogenetic (tree) power achieving the minimum edge difference from the input graph.

Called *Closest Phylogenetic (tree) Root*, CPR_k (CTR_k)

 - CTR_k is hard for $k \geq 2$ (1998)
 - CPR_k is hard too (2001)
- For today:
 - key ideas in proving the hardness
- A variant of PR_k to have degree-2 internal nodes:
 1. solvable for $k = 2, 3, 4$ (1999)
 2. remains unknown for $k \geq 5$
 3. optimization version unknown

289

Facts:

- Notions:
 - chordal
 - maximal clique
 - critical clique (CC) K :
a maximal clique of vertices having a same set of neighbors
- If a graph has a k -root phylogeny (tree), then it is *chordal*.
- Checking if a graph is chordal can be done in linear time.
- Computing all maximal cliques in a chordal graph can be done in linear time.
- Critical cliques form a partition of the vertex set.
- Critical clique graph can be constructed in linear time.

290

An algorithm for PR_3 :

- More facts:
 - if graph G has a 3-root phylogeny, then it has a 3-root phylogeny where every CC of G is adjacent to exactly one internal node, which is called the representing node of the CC.
 - if graph G has a 3-root phylogeny T , then representing nodes for adjacent CCs are adjacent in T .
 - if a critical clique CC is a singleton, then its representative must be deep internal (adjacent to at least 2 other internals).
- Problem reduces to: Restricted 1-Root Steiner Phylogeny Problem (RSR1)

Given a graph $G = (V, E)$ and $S \subseteq V$, find a Steiner phylogeny T of V such that every vertex in S is internal in T , and for each pair of vertices u and v , u and v are adjacent in G iff they are adjacent in T .

291

An algorithm for RSR1:

- Notations (G must be a tree):
 1. S -vertex — vertices need to be internal in T (degree 0 or 1 in G)
 2. Z -tree — a component tree of G containing ≥ 2 S -vertices
 3. Y -vertex — an S -vertex inside some Z -tree
 4. W -vertex — a degree-0 S -vertex
 5. X -vertex — a vertex from each non- Z -tree (S -vertex priority)
- Algorithm:
 - if $|X| < 2|Y| - 3(|Z| - 1)$ or $|Z|$ even or $|W| > |X| - 2|Y| + 3|Z| - 3$, no such phylogeny
 - otherwise,
 1. introducing $(|Z| - 1)/2$ Steiner nodes to collapse Z -trees into one single Z -tree;
 2. using 2 X -vertices to make a Y -vertex degree-2;
 3. the remaining 1 Y -vertex and $|X| - 2|Y| + 3|Z| - 1$ X -vertices can be made into a full Steiner tree (each internal node has degree 3);
 4. inserting W -vertices onto edges in the above full Steiner tree.
- Special case: G is a non-trivial tree, has a root that is itself
 $\leftrightarrow S = \emptyset$

292

An algorithm for ΔPR_k :

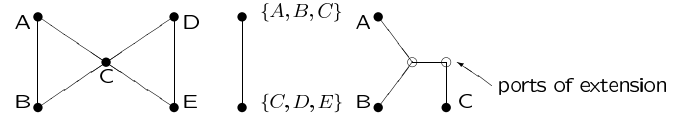
- Facts:
 - the maximum size of a clique in G is at most

$$\begin{cases} \Delta(\Delta - 1)^{k/2-1} & \text{if } k \text{ is even} \\ (\Delta - 1)^{(k+1)/2} - 1 & \text{if } k \text{ is odd} \end{cases}$$
 - graph G is chordal and thus has a clique-tree-decomposition D
- Algorithm — key ideas:
 - perform a dynamic programming on a rooted version of D
 - DP starts at the leaves of D and proceeds upwards
 - after the root of D is processed, a desired phylogeny of G will be found if there is one

293

The processing of a node α of D :

- U_α — the set of vertices contained in the maximal cliques associated with α and its descendants in D
- While processing α , compute a set of trees T_α such that T_α may possibly be a subtree of a desired phylogeny and each vertex of U_α is a leaf of T_α



- The essence
 - only those ports that close to the vertices of the maximal clique associated with α are necessary, where “close” means “at distance $< k$ ”
 $\#(\text{necessary ports in each } T_\alpha) \text{ is } O(k, \Delta)$
 - In each T_α , only the necessary ports and the vertices of the maximal clique associated with α are related to further extension of T_α .
 - Among all T_α with the same necessary ports and structure between ports, we only keep one of them in the dynamic programming table.

294

NP-hardness of CPR_k :

- Reduction from Fitting Ultrametric Tree problem:
 Given an integer b and an $n \times n$ symmetric matrix M such that $M(i, i) = 0$ for all $1 \leq i \leq n$ and $M(i, j) = 1$ or 2 for all $1 \leq i < j \leq n$.
 Is there a rooted tree T with 3 layers and the leaves are numbered $1, 2, \dots, n$ such that there are at most b pairs (i, j) with

$$M(i, j) \neq d_T(i, j)/2?$$

- Key ideas:
 - trying to associate the number of pairs violating matrix constraint with the number of edge difference between graph G and the phylogenetic power of T
 - 1st try: $V = \{1, 2, \dots, n\}$, $E = \{(i, j) \mid M(i, j) = 1\}$
 sounds good but T might have degree-2 internal nodes!
 - 2nd try: duplicating the vertices

$$M'_{2n \times 2n} = \begin{pmatrix} M_{n \times n} & M_{n \times n} + I_n \\ M_{n \times n} + I_n & M_{n \times n} \end{pmatrix}$$

$$V = \{1, 2, \dots, n, n+1, n+2, \dots, 2n\}$$

$$E = \{(i, j) \mid M'(i, j) = 1\}$$

295

Open problems:

- What is the complexity of PR_k for $k \geq 5$?
- Does CPR_k ($k \geq 2$) admit *constant-ratio* polynomial time approximation algorithm?
 Design such an algorithm.
- Same questions asked for ΔCPR_k for $k \geq 2$.
- Same questions asked for CTR_k and ΔCTR_k for $k \geq 2$.
- What is the complexity of Closest Chordal Graph problem? and its approximability?

296