

**BIO/CS 423 Computational Biology**  
**Spring 2018**  
**Course Handouts**  
**Dr. Tammy VanDeGrift**

**Name:** \_\_\_\_\_

**If found, call/email:** \_\_\_\_\_



## CS423/BIO423 Calendar Spring 2018

Calendar subject to change – see Moodle for latest updates. Complete assigned reading prior to lecture.

Updated 12/18/2017.

Week	Monday	Wednesday	Friday
Jan 15 - 19	Course Introduction Intro to Python	Python  <b>Reading:</b> pre-lab 1	Lab 1: Python  <b>DUE:</b> Pre-lab 1
Jan 22 - 26	Cell Biology  <b>Reading:</b> pp. 1 – 18, 26 -27	Lab 2: Databases and Genomes <b>DUE:</b> Pre-lab 2	Algorithms  <b>DUE:</b> Lab 1
Jan 29 – Feb 2	Locating Motifs  <b>Reading:</b> pp. 247 – 254	Lab 3: Finding Genes and Motifs <b>DUE:</b> Lab 2, Pre-lab 3	Sequence Evolution and Distance Metrics
Feb 5 - 9	Intro to Sequence Alignment	Lab 4: Mutations and Sequence Evolution <b>DUE:</b> Lab 3, Pre-lab 4	Dynamic Programming and Sequence Alignment
Feb 12 - 16	Global Sequence Alignment  <b>Reading:</b> pp. 29 - 39	Local Sequence Alignment Longest Subsequence <b>Reading:</b> pp. 39 – 53 <b>DUE:</b> Lab 4	Lab 5: Sequence Alignment  <b>DUE:</b> Pre-lab 5
Feb 19 - 23	Lab 5: Sequence Alignment	BLAST Review for exam <b>Reading:</b> pp. 109 - 124	<b>EXAM 1</b>
Feb 26 – Mar 2	BLAST  <b>DUE:</b> Lab 5	Lab 6: BLAST  <b>DUE:</b> Pre-lab 6	Multiple Sequence Alignment  <b>Reading:</b> pp. 139 – 152
Mar 5 - 9	Phylogenetic Trees  <b>Reading:</b> pp. 155 – 193	Lab 7: Multiple Sequence Alignment <b>DUE:</b> Lab 6, Pre-lab 7	Phylogenetic Trees  <b>DUE:</b> Project Proposal
Mar 12 - 16	<b>SPRING BREAK</b>		
Mar 19 - 23	DNA Microarrays  <b>Reading:</b> pp. 18 - 25	Lab 8: Research  <b>DUE:</b> Lab 7, Pre-lab 8	Clustering
Mar 26 - 30	Clustering	Lab 9: Microarrays and Clustering <b>DUE:</b> Lab 8, Pre-lab 9	<b>EASTER BREAK</b>
Apr 2 - 6	<b>EASTER BREAK</b>	Regulatory Motifs, EM <b>Reading:</b> pp. 254 – 260	Regulatory Motifs
Apr 9 - 13	Lab 10: Regulatory Motifs <b>DUE:</b> Lab 9, Pre-lab 10	HMMs Review	<b>EXAM 2</b>
Apr 16 - 20	Project or Guest  <b>DUE:</b> Lab 10	Project or Guest  <b>DUE:</b> Interim Code Demos	Project or Guest
Apr 23 - 27	Project or Guest	Course Evaluations and Project	Final Code Review <b>DUE:</b> Project Report / Code
TUES, May 1 8 – 10 am	Presentations <b>DUE:</b> Project Presentations		



# CS/BIO 423: Computer Networks and Internetworking Spring 2018

## Course Information

**Instructor:** Dr. Tammy VanDeGrift

Email: vandegri@up.edu  
Office Phone: 503-943-7256  
Office: Shiley 223  
Website: Course information on Moodle, learning.up.edu

**Meetings:** MWF 9:15 – 10:10 am

**Classroom:** Shiley 249

**Office Hours:** M 2:30 – 4pm, T 9 – 11:30 am; W 8:30 – 9:10am; F 11:25 – 12:20pm

**Bulletin Description:** Algorithmic and analysis techniques for biological data such as DNA, RNA, proteins, and gene expression. Topics include molecular biology, alignment and searching algorithms, sequence evolution algorithms, genetic trees, and analysis of microarray data. This course is interdisciplinary and assumes programming skills. (Pre-requisites: MTH 201, CS 203 with a grade of C- or better; one of the following: BIO 205, BIO 207, CS 305 with a grade of C- or better)

## Student Outcomes

At the end of the course, students should be able to:

- Create Python programs to analyze biological data
- Use databases to find biological data
- Describe roles of DNA, RNA, genes, transcription factors, and proteins
- Design and conduct a self-directed project
- Analyze the running time of computational algorithms

These outcomes will be accomplished by:

- Completing labs, a project, exams, a paper and a presentation
- Participating in class discussion, labs, and activities through regular attendance
- Seeking help of the professor and classmates when necessary
- Communicating ideas orally and in writing
- Providing help rather than giving answers to classmates seeking help

This is an elective course, so it does not serve as a benchmark course for ABET student outcomes.

## Course Philosophy

**General:** This cross-listed course is designed to introduce concepts related to algorithms applied to biological data. ***Because this course covers topics that build on one another, it is critical that you keep up with the material by completing assignments on time and preparing for each class session by reading assigned material.*** It's okay to struggle with the concepts, but it is your responsibility to seek help when you are confused. You will **not** succeed in this course if you start labs the night before they are due.

**Code of Academic Integrity:** Academic integrity is openness and honesty in all scholarly endeavors. The

University of Portland is a scholarly community dedicated to the discovery, investigation, and dissemination of truth, and to the development of the whole person. Membership in this community is a privilege, requiring each person to practice academic integrity at its highest level, while expecting and promoting the same in others. Breaches of academic integrity will not be tolerated and will be addressed by the community with all due gravity. See University Bulletin for policy. (from *UP Bulletin*)

**Seeking Help:** I expect you to have questions as you learn the course material. You may receive help from classmates (see below about Collaborative Learning) and seek help from the instructor. I encourage you to ask questions during lecture meetings and attend office hours.

**Collaborative Learning:** Your classmates are a huge resource available to you. Because we understand material in different ways, I encourage you to discuss concepts from the course, but **any work that you turn in must be your own**. In some cases, you will work with another student in the class and both members will be responsible for understanding team-authored assignments. Unacknowledged copying or using parts of someone else's work, the Internet, or other resources, even if it has been modified by you, is plagiarism and is not acceptable. When you work with others not on your team, please acknowledge places where you received help. An acceptable way to collaborate is to discuss problems and potential solutions and then writing the solutions on your own. When giving help to other classmates, please do not give them the answer. Instead, ask questions to learn of their understanding and give conceptual explanations - this practice will help you master the material yourself. Remember: you must turn in work that is your own, you must acknowledge the people who helped you, and you are encouraged to seek help when you are confused.

**Instructor's and Students' Responsibilities:** In this course, the instructor's job is to guide you in learning about computational biology. In addition to traditional lecturing, Tammy will have regular discussions and activities during lectures. You will generally work on labs during class time on Wednesdays. Tammy expects your full participation and readiness to learn at all class meetings. I will do my best to offer suggestions, activities, and explanations to help you learn the material.

**Classroom Conduct:** The Shiley School of Engineering is committed to developing and actively protecting a classroom environment in which respect must be shown to everyone in order to facilitate and encourage the expression, testing, understanding and creation of a variety of ideas and opinions. Failure to meet these standards will result in removal from the class session.

In order to maintain a positive learning environment, students should avoid disruptive behaviors such as: receiving cell phone calls during class, leaving class early or coming to class habitually late, talking out of turn, doing assignments for other classes, reading the newspaper, sleeping, and engaging in other activities that detract from the classroom learning experience.

Because this course takes place in a computer classroom, students are expected to use computers for activities related to the course when instructed by the professor. Computers are not to be used for email, web surfing, social media, video playing, and other personal entertainment unless such activity is related to course activities. Do not unplug computer cables. Respect the computer equipment (no food and drink spills).

**The Learning Commons:** The Learning Commons, located in the Shepard Academic Resource Center, houses the Writing Center, Math Resource Lab, Speech Resource Center, and the International Language Lab.

Writing: Go to <https://www1.up.edu/learningcommons/writing-center/index.html>. You will need to register as a user the first time you go there. If you cannot make any of the posted office hours, you can arrange an appointment by emailing [writing@up.edu](mailto:writing@up.edu).

Math: Math assistants are available on a walk-in basis. Please go to <https://www1.up.edu/learningcommons/math-resource-center/index.html> for a current schedule of hours math assistants are available.

Speech: Speech assistants are available to help with public presentations on a walk-in basis. Please go to <https://www1.up.edu/learningcommons/speech-and-presentation-lab/index.html> for a current schedule of hours speech assistants are available.

Group Process: Group process tutors are available on a walk-in basis. Please go to <https://www1.up.edu/learningcommons/group-work-lab/index.html> for a current schedule of hours group process assistants are available.

The Learning Assistance Counselor: He or she teaches learning strategies and skills that enable students to become more successful in their studies and future professions. The Counselor provides strategies to assist students with reading and comprehension, note-taking and study, time management, test-taking, and learning and remembering.

## Assessment of Learning

I will assess your learning based on your submitted work, including pre-labs, labs, a project, a presentation, exams, and in-class activities/quizzes. Generally, the labs, project, and in-class activities are your chance to learn, while the exams are the main way I will assess what you have learned. I expect you to submit your labs and projects by the due date and time. If circumstances arise (e.g. you are ill for an extended period of time, you are out of town for a university-related activity) that prevent you from submitting your work on time, please discuss the reason with me *before* the due date. Note that you have two free late days to use on labs.

## Grading Scheme

Course grades will be assigned based on the total points you earn during the semester, weighted accordingly to the categories shown below. The minimum cutoffs for grades will not change. I do reserve the right to raise your grade, but the following minimum percentages are guaranteed. (For example, if you earn 90% of the points, you will get an A-. If you earn 89% of the points, you earn a B+ but I reserve the right to raise your grade to an A-.)

- $\geq 93\%$  A
- $\geq 90\%$  A-
- $\geq 87\%$  B+
- $\geq 83\%$  B
- $\geq 80\%$  B-
- $\geq 77\%$  C+
- $\geq 73\%$  C
- $\geq 70\%$  C-
- $\geq 67\%$  D+
- $\geq 63\%$  D
- $\geq 60\%$  D-
- $< 60\%$  F

### Weights of Graded Work in the Course

Percent	Category	Description
30	Exams	Two midterm exams @15% each, completed <b>individually</b> .
45	Labs	Pre-labs should be completed <b>individually</b> . Labs should be completed in <b>pairs</b> (multi-disciplinary, if possible). You should work with a different partner for each lab.
20	Project	A <b>team</b> programming project related to computational biology, including the proposal, beta test, code review, paper, and presentation. The presentation will take place during the final exam slot.
5	Professionalism + Quizzes	Attendance and active engagement in class sessions; professional behavior demonstrated toward peers and classroom environment; if a student misses class, it is the student's responsibility to get notes from a classmate; some sessions may include quizzes; this grade is <b>individual</b>

**Late Assignments:** You are granted **two full days** to use at your discretion for submitting late pre-labs or labs without penalty. For example, you may choose to turn in two different assignments 24 hours after their due dates and times. You could choose to use the two days (48 hours) on a single assignment. Weekends count as regular days. If you use a late day, indicate the use of late day(s) with a note on the assignment when you submit your work. For labs, both partners will be charged late days. If you need to submit an assignment late (due to illness, death in the family, out of town for a university event) and you have already used your two late days, please contact me before the due date and time and we can discuss your options. Late days may **not** be used for exams, in-class activities, quizzes, or project deliverables.

### Logistics

**Textbook and Readings:** The required textbook for the course is *Algorithms in Bioinformatics: A Practical Introduction* by Wing-Kin Sung [ISBN: 978-1-4200-7033-0]. The text is available for purchase at the campus bookstore. I ask that you read certain sections *before* attending the accompanying class session (see the online course calendar for the latest updates to the readings). Lectures are intended to supplement the textbook. In some cases, I will ask you to read journal articles related to the topics; these will be posted on the website or distributed in class.

**Course Calendar:** The latest version of the calendar is on Moodle. The course calendar lists the lecture topics, assigned readings, exams, and due dates for assignments. The calendar is subject to change as the semester progresses.

**Accessibility Statement:** The University of Portland endeavors to make its courses and services fully accessible to all students. Students are encouraged to discuss with their instructors what might be most helpful in enabling them to meet the learning goals of the course. Students who experience a disability are also encouraged to use the services of the Office for Accessible Education Services [AES], located in the Shepherd Academic Resource Center (503-943-8985). If you have an AES Accommodation Plan, you should make an appointment to meet with your faculty member to discuss how to implement your plan in this class. Requests for alternate location for exams and/or extended exam time should, where possible, be made two weeks in advance of an exam, and must be made at least one week in advance of

an exam. Also, you should meet with your faculty member to discuss emergency medical information or how best to ensure your safe evacuation from the building in case of fire or other emergency.

**Non-Violent Community Statement:** University of Portland Faculty, Staff, and Students are committed to creating a community free from interpersonal violence, in which all members feel safe and respected. Each of us has a personal responsibility to reject violence or intimidation of any kind. Resources for those experiencing or wishing to report violence can be found on UP's Community Against Violence website.

**Assessment Disclosure Statement:** Student work products for this course may be used by the University for educational quality assurance purposes.

**Academic Regulation Statement:** Policies governing your coursework at the University of Portland can be found in the University Bulletin.

**Mental Health Statement:** As a college student, you may sometimes experience problems with our mental health that interferes with academic experiences and negatively impact daily life. If you or someone you know experiences mental health challenges at UP, please contact the University of Portland Health and Counseling Center in Orrico Hall at [www.up.edu/healthcenter/](http://www.up.edu/healthcenter/) or 503-943-7134. Their services are free and confidential, and if necessary they can provide same day appointments. Also know that the University of Portland Public Safety Department (503-943-4444) has personnel trained to respond sensitively to mental health emergencies at all hours. Remember that getting help is a smart and courageous thing to do – for yourself, for those you care about, and for those who care about you.

**Improving the Course:** I welcome your feedback about the course at any time. I may ask for your feedback periodically and you will have the opportunity to evaluate the course at the end of the semester.

# HANDOUTS

## CS423: First Look at python

```
#!/usr/bin/python

# demo code for first day of CS 423

sequence = "ATCGATTCG"
firstC = sequence.find("C")
print(firstC)

# look at iteration

count = 0
while count < 10:
    print("I like programming")
    count = count + 1

for c in sequence:
    print(c)

for i in range(10): #iterate 10 times
    print(i)

for i in range(2,10):
    print(i)

for j in range(2,len(sequence)):
    print(sequence[j])

# In general, when processing strings, try to use built-in string functions
# rather than writing loops. String functions are faster.
```

Top line shows path to the python executable, so this can be run as a script in addition to IDLE

## BIO/CS 423 Lecture 1 Activities

**Activity 1:** With a partner, write a python script called `percentA.py` that defines a DNA sequence (you can choose the DNA sequence bases) and prints the overall percentage of adenine (A) nucleotides.

Hint: check out the `len` and `count` functions

**Activity 2:** With a partner, write a python script called `third.py` that defines a function called **everyThird** that takes in a string called `sequence` as a parameter. The function should print out every third character in the string, starting with the 0<sup>th</sup> character. For example, if the input string is “computers”, then it should print the letters c, p, and e.

```
def everyThird(sequence):
```

```

#!/usr/bin/python

# BIO/CS 423 Lab 1
#
# NAMES:

#####
# Exercise 0
# convertFileToSequence - takes a FASTA file and returns the
# sequence as a string
#####
def convertFileToSequence(filename):
    # read in file
    file = open(filename)

    # read in first line
    header = file.readline()
    if (header[0] == '>'):
        print("in FASTA format")
    else:
        print("invalid format")
        file.close()
        return

    # read in rest of file
    sequence = file.read()

    # close file
    file.close()

    # remove all return and newline characters
    sequence = sequence.replace("\r", "")
    sequence = sequence.replace("\n", "")
    return sequence #returns the string

#####
seq = convertFileToSequence("sequence.txt")
print(seq)

```



## Quick reference: files, randomness, lists, math, keyboard input

### BIO/CS 423

Python standard library information: <https://docs.python.org/3/library/>

All documentation: <https://docs.python.org/3/>

#### Files

```
file = open(filename)           ## opens file (not append mode)
file = open(filename, 'a')      ## opens file in append mode
file = open(filename, 'w')      ## opens file in write mode
file.read()                     ## reads entire file and returns the string
file.write(value)               ## writes value to file
file.readline()                 ## reads one line of the file and returns it as a
string
file.close()                    ## closes file - be sure to do this when you are
finished
                                ## with it
```

Note: You may need to put “\r\n” in a string to write to a file to get the newline on Windows

#### Random Numbers

```
import random                   # to load the methods for generating random numbers
random.random()                 # generates floating point number in [0.0, 1.0)
random.randint(start, stop)     # generates random integer in [start, stop]
random.randrange(int)           # generates random integer in [0, int-1]
random.gauss(mu, sigma)         # generates random number from Gaussian distribution
                                # with mean mu and standard deviation sigma
random.shuffle(list)            # returns a shuffled version of the list
```

#### Lists

This is a collection in python; similar to an array in Java, but with the following differences:

- Items need not be the same type
- Can use + to combine lists

Example:

```
aList = ["hi", 4, 7.5, "bye"]
```

```
alist[0]    # returns "hi"
```

Unlike strings, lists are mutable objects in python:

```
aList[2] = 8.5          # now, aList is ["hi", 4, 8.5, "bye"]
```

#### Operators on Lists

```
+           combines two lists into a single list
*           concatenates list # of times given by second argument
```

```

in          just like with strings, returns True if item is in list
            "hi" in aList # returns True
not in      returns opposite of in
len(list)   returns size of list
[start:end] returns sublist starting at position start and up to but not including
            end

```

### Methods on Lists:

```

list.append(item)          # adds item to end of list
list.insert(i, item)      # puts item in ith position and shifts others
list.pop()                # removes and returns last item
list.pop(i)               # removes and returns ith item
list.reverse()            # reverses list
list.index(item)          # returns index of first occurrence of item
list.count(item)          # returns number of times item appears in list
list.remove(item)         # removes first occurrence of item
list.sort()                # sorts list (as long as items in list are comparable)
max(list)                  # returns maximum element in list
min(list)                  # returns minimum element in list

```

### Input from keyboard:

```

userInput = input("Enter your name: ") # will prompt user and store what they type
                                         # into userInput as a string

```

### Math

There is a math module (see documentation for all math functions)

Need to import module if you need to use these functions:

```
import math
```

Similar to what you find on a calculator:

```

math.cos(val)
math.sin(val)
math.sqrt(val)
math.fabs(val)          # absolute value
math.pi                 # constant
math.e                  # constant
math.floor(val)
math.ceil(val)          .... many more

```

### Program Organization, Compilation, and Execution

By the way, python can be written in object-oriented style, but we'll stick with the functional approach for now. But, as programs get bigger and more complex, it might make sense to write OO code.

You can define your functions in a python file and also execute functions below those (so define at the top and use at the bottom).

You can run your python program outside the interpreter (that's what the first line helps you do). In a shell, just type the name of the python code:

```
$ lab1.py
```

```
$ ./lab1.py          if . is not in your path
```

Python is a hybrid interpreted/compiled language. When you first import a module, python creates a .pyc file that is compiled python byte code (to help speed up execution). If the .pyc has the same timestamp as the .py file, the .pyc is used. If they are different, the .py file is reread/recompiled.



## **Molecular Biology**

See Moodle posting and textbook, pages 1 – 18



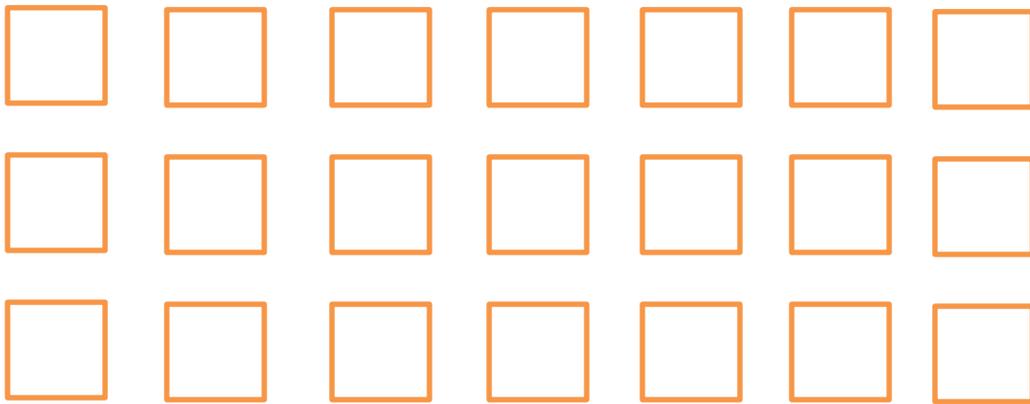
## CS423: Algorithms

### Single Pile Nim

Two-player game

Rule: On the player's turn, he/she can cross out 1, 2, or 3 boxes.

Winner: The player who crosses out the final box.



What is the winning “strategy”?



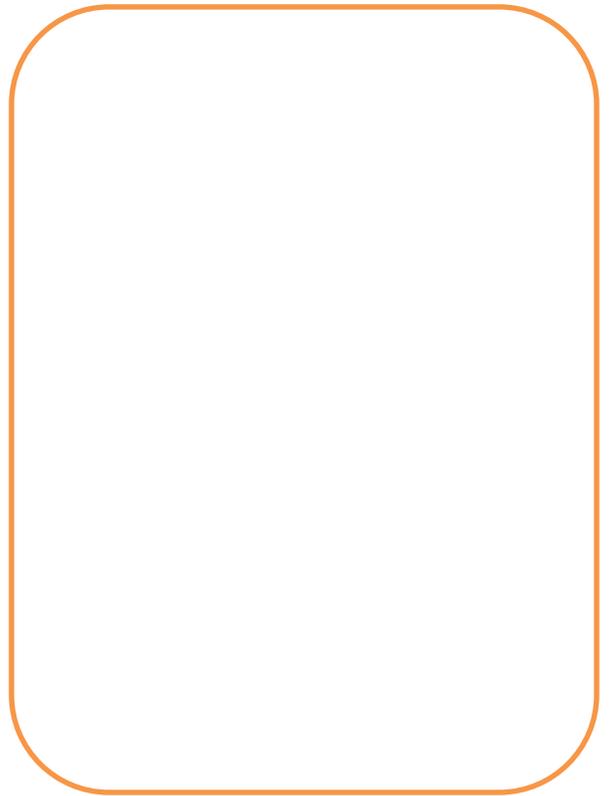
## Two Pile Nim

Two-player game

Rule: On the player's turn, he/she can cross out any number of boxes from 1 to the number that remain open in a **single** pile.

Winner: The person who crosses out the final box.

To play: Draw as many boxes as you would like in each pile. Then play the game.



What is the “winning” strategy?

```

#!/usr/bin/python

#Code to demonstrate running time of two different implementations
#Lecture 6
#CS 423

#Activity: Assume sequence has N nucleotides. Assume there are P instances
#of substring TATAAA in sequence.
#How many nucleotides are examined or copied in each function below?
#Give your answer in terms of N and P.

# idea: keep searching from previously found substring
def TATAPosition(sequence):
    foundPos = sequence.find("TATAAA", 0) #start searching from position 0
    while(foundPos >= 0): #stop once find returns -1
        print(foundPos)
        foundPos = sequence.find("TATAAA", foundPos + 1)
        # start search from string index (foundPos + 1)

# idea: every time the substring is found, remove the first part of the sequence
def TATAPosition2(sequence):
    offset = 0
    position = sequence.find("TATAAA") #find first occurrence of TATAAA
    while(position >= 0): #keep going if found an occurrence
        print(position + offset)
        offset = position + offset + 1
        sequence = sequence[position + 1:len(sequence)]
        position = sequence.find("TATAAA")

# code to execute
sequence = "AAAATATAAAGGGCCTATAAATATAAAAAATATAATATAAATATAAACCCCTATAAA"
#sequence = "GGCCTTAAGG"
#sequence = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATATAAAGGGG"
TATAPosition(sequence)
TATAPosition2(sequence)

```



## CS423: Motifs and Profiles

**Table 1: Positions 3 – 9 of CRP site in E. coli**

TTGTGGC  
 TTTTGAT  
 AAGTGTC  
 ATTTGCA  
 CTGTGAG  
 ATGCAAA  
 GTGTTAA  
 ATTTGAA  
 TTGTGAT  
 ATTTATT  
 ACGTGAT  
 ATGTGAG  
 TTGTGAG  
 CTGTAAC  
 CTGTGAA  
 TTGTGAC  
 GCCTGAC  
 TTGTGAT  
 TTGTGAT  
 GTGTGAA  
 CTGTGAC  
 ATGCGAC  
 TTGTGAG

Do you see a pattern?

**Table 2: Profile of CRP site**

	3	4	5	6	7	8	9
A	.35	.043	0	.043	.13	.83	.26
C	.17	.087	.043	.043	0	.043	.3
G	.13	0	.78	0	.83	.043	.17
T	.35	.87	.17	.91	.043	.087	.26

**Table 3: LLR weight matrix of CRP site**

	3	4	5	6	7	8	9
A	.48	-2.5	-inf	-2.5	-.94	1.7	.061
C	-.52	-1.5	-2.5	-2.5	-inf	-2.5	.28
G	-.94	-inf	1.6	-inf	1.7	-2.5	-.52
T	.48	1.8	-.52	1.9	-2.5	-1.5	.061

**Your turn to practice – motif for start codon:**

Hypothetical translation start sites (In general “ATG” is the start codon for a gene, but it turns out that it could be “GTG” or “TTG”). Here’s an example **profile** for the start codon. Answer the questions below.

	1	2	3
A	.625	0	0
C	0	0	0
G	.25	0	1
T	.125	1	0

- $P(\text{ATG is a start codon}) =$
- $P(\text{GTG is a start codon}) =$
- $P(\text{CTG is a start codon}) =$
- Likelihood Ratio(ATG is a start codon given uniform background distribution of nucleotides) =
- Likelihood Ratio(GTG is a start codon given uniform distribution) =
- Likelihood Ratio(CTG is a start codon given uniform distribution) =
- Complete the weight matrix of LLRs below for the start codon site [you will likely need a calculator]. Assume a uniform background distribution for nucleotides.

	1	2	3
A			
C			
G			
T			

- $\text{LLR}(\text{ATG}) =$
- $\text{LLR}(\text{GTG}) =$

j.  $\text{LLR}(\text{CTG}) =$



# CS423: Mutations

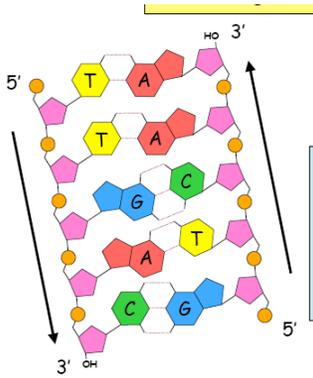


Figure 1: DNA nucleotide bonds

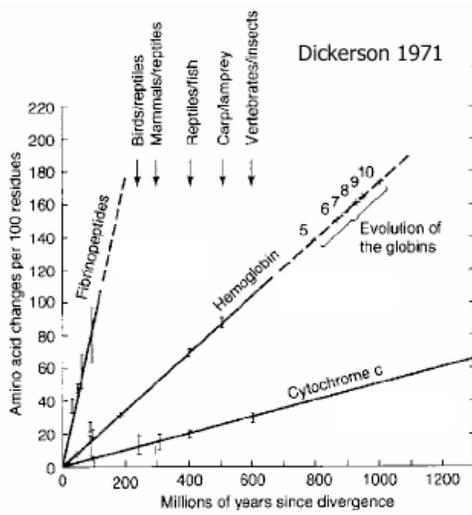


Figure 2: Amino acid substitutions over time for three proteins

Table 1: Sequence Evolution (bold are changes, underscore are final changes)

```

ACCATGGAATTTTATACCCT
AGCATGGTATTTTATACGCT
AGCATCGAAATTTTATACGAT
ATCATGGAAATTAATACCAT
ATCATGCAATATAATACCG
    
```

## CS 423: Jukes Cantor Correction

**Table 2: Substitution rates for DNA nucleotides**

	A	C	G	T
A	(1-3a)	a	a	a
C	a	(1-3a)	a	a
G	a	a	(1-3a)	a
T	a	a	a	(1-3a)

For any nucleotide, the probability of a substitution at a given time is 3a (add up the probabilities in a single column). Over a period of time t, the number of substitutions is 3at.

Since we are looking at a model such that the sequences diverge from a common sequence, there is a rate of 3at on each branch. Thus, the actual rate of substitutions (K) that take place over time t is 3at + 3at (one for each branch).  $K = 6at$ .

But, we don't get to observe all the substitutions, since some may change and change back to the original. We need an estimate for the "actual" rate of substitutions, but we don't know the value for 'a' and we don't know the value for 't'. Hmmm...

What we want:

$$K = \text{actual rate of substitutions} = (\text{correction factor}) * (\text{p distance})$$

### Derivation of Jukes Cantor Correction Factor:

Probability that nucleotide will be the same at time (t+1) given time (t):

$$= (\text{Probability that it is residue } r)(\text{Probability that it was residue } r \text{ at time } t) + (\text{Probability that it changes to } r)(\text{Probability that it was a different nucleotide at time } t)$$

$$P_{\text{same}(t+1)} = (1-3a)P_{\text{same}(t)} + a(1-P_{\text{same}(t)})$$

Solving with differential equations, we get:

$$P_{ii}(t) = \text{Prob}(\text{nucleotide stays the same at time } t) = \frac{1}{4} + \frac{3}{4}(e^{-4at})$$

$$P_{ij}(t) = \text{Prob}(\text{nucleotide goes from } i \text{ to } j \text{ at time } t) = \frac{1}{4} - \frac{1}{4}(e^{-4at})$$

Now, we can compute the probability that two sequences differ at a given position:

$$p = 1 - (\text{probability they are identical})$$

$$p = 1 - (\text{prob of both staying the same} + \text{prob of both changing to same base})$$

$$p = 1 - [(P_{AA(t)})(P_{AA(t)}) + (P_{AT(t)})(P_{AT(t)}) + (P_{AC(t)})(P_{AC(t)}) + (P_{AG(t)})(P_{AG(t)})]$$

Substituting the values from above, we get:

$$p = 1 - [(\frac{1}{4} + \frac{3}{4}(e^{-4at}))(\frac{1}{4} + \frac{3}{4}(e^{-4at})) + 3[(\frac{1}{4} - \frac{1}{4}(e^{-4at}))(\frac{1}{4} - \frac{1}{4}(e^{-4at}))]]$$

$$p = 1 - [\frac{1}{4} + \frac{3}{4}(e^{-8at})]$$

$$p = \frac{3}{4} - \frac{3}{4}(e^{-8at})$$

$$p = \frac{3}{4} (1 - e^{-8at})$$

Now, rearranging to isolate the term 8at:

$$(4/3)p = (1 - e^{-8at})$$

$$e^{-8at} = 1 - (4/3)p$$

$$-8at = \ln(1 - (4/3)p)$$

$$8at = -\ln(1 - (4/3)p)$$

Since, we know the number of substitutions from a single source is  $K = 6at$ , we can determine  $K$  by taking  $\frac{3}{4}$  of the value for  $8at$ :

$$K = -3/4 \ln(1 - (4/3)p)$$

This is the Jukes Cantor correction.

### Example

If we find that that the proportion distance  $p$  is: .2 for two DNA sequences, then

$$K = -3/4 \ln(1 - (4/3)*.2)$$

$$K = .2326$$

# so this gives us an estimate of the actual rate of change

# not just the observed substitution rate

# makes sense, since there are "hidden" changes, so  $K$  should be bigger than .2



## CS423: Practice With Distance Calculations

Individually or in small groups, complete the following questions:

### 1. Calculating sequence differences

Assume these are two DNA sequences that evolved from a common ancestral sequence:

AATTGCTAGACTACTTAGGG  
ATATGCGAGACTTGTTCGG

What is the Hamming distance?      H = \_\_\_\_\_

What is the p distance?              p = \_\_\_\_\_

What is the Jukes Cantor estimate?      K = \_\_\_\_\_  
 $K = -3/4 \ln(1 - (4/3) p)$

Given that the sequence has 20 nucleotides, the estimate K is for the rate of change at a single position. Using K, how many nucleotide substitutions are estimated? \_\_\_\_\_

## CS 423: Intro to Sequence Alignment

A. Why would we want to align DNA sequences to see if they are similar?

Example: multiple sclerosis

Hypothesis: T cells (normally identify foreign agents in the body) mistakenly identify the proteins in the nerve's myelin sheaths as foreign

B. How would you test this hypothesis?

C. Consider the following DNA sequences:

ATATATATA

TATATATAT

What is the Hamming distance for these sequences? \_\_\_\_\_

D. But, how really different are these sequences?

E. Assume you have the following operations: delete one nucleotide, insert one nucleotide, and substitute one nucleotide for another (D, I, S). How many operations does it take to make the sequences in part C have a Hamming distance of 0? \_\_\_\_\_

F. How many operations (D, I, S) does it take to get from the following top sequence to the bottom sequence?

TGCATAT

ATCCGAT

\_\_\_\_\_ What are those operations?

G. Given two strings (sequences of characters from the same alphabet), what is the optimal alignment of those sequences?

Example:

s1 = ACGCTG

s2 = CATGT

How would you align these to have the most matches?

Example alignment:

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

Another example alignment:

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

When aligning, the widest this will be is  $N + M$  where  $N$  and  $M$  are the lengths of the strings.

**Vocabulary:**

Match: column with same letter

Mismatch: column with differing letters

Insertion: column with space on top

Deletion: column with space on bottom

Indel: insertion or deletion

In immediate above example, the C/A column is a mismatch, the C/C column is a match, the A/- column is a deletion, the -/T column is an insertion.

H. Also need a way to “score” the alignment, so we can determine the optimal alignment. For simplicity, let’s assume every match is worth 2 points and every mismatch is worth -1. Then, in the example in part G, the score is  $-1 + 2 + -1 + -1 + 2 + -1 + 2 + -1 = 1$ .

Let  $d(x,y)$  be the score of aligning symbols  $x$  and  $y$ .

The length of a string  $S$ , denoted by  $|S|$  is the number of symbols in  $S$ . We’ll denote the symbols of  $S$  as  $S[0], S[1], S[2]...$ etc.

An alignment A between strings S and T maps them into strings S' and T' where S' and T' may contain space characters, where:

$$|S'| = |T'|$$

The removal of spaces in S' gives the string S and the removal of spaces in T' gives the string T

The alignment score of A, aligning strings S and T is:

$\text{SUM}_{i=0 \text{ to } n} (d(S'[i], T'[i]))$                       # add the distance for each character in the alignment  
# n is the total width of the aligned sequences

I. String alignment problem (formalized):

Given two strings S and T (sequences of characters from the same alphabet), what is the optimal (maximum scoring) alignment of S and T?

Can you think of an obvious algorithm to solve the string alignment problem?

How long does your algorithm take (assume sequence S has N characters and assume sequence T has N characters)?



## CS 423: String Alignment and Intro to Dynamic Programming

### String Alignment Overview

Given two strings S and T (sequences of characters from the same alphabet), what is the optimal (maximum scoring) alignment of S and T?

A. Can you think of an obvious algorithm for the string alignment problem (from last lecture)?

Idea: look at all possible “aligned” subsequences of each string where the other parts match up with dashes on the other string

$$|S| = n$$

For all  $i$ ,  $0 \leq i \leq n$  do

    For all subsequences A of S with  $|A| = i$  do

        For all subsequences B of T with  $|B| = i$  do

            Form an alignment that matches  $A[k]$  with  $B[k]$   $0 \leq k < i$

            And matches all other characters with dashes

            Determine the score of this alignment

            Keep maximum alignment

Return maximum alignment

B. But, what is the running time of this algorithm?

Outer loop: runs  $n$  times

    For all subsequences A of S:  $\binom{n}{i}$  subsequences with length  $i$

        For all subsequences B of T:  $\binom{n}{i}$  subsequences with length  $i$

            Forming alignment:  $(2n - i)$  characters long [since there are  $n$  characters in S and  $(n - i)$  unmatched characters with T]

So, we have a running time of:

$$\sum_{i=0}^n \binom{n}{i} \binom{n}{i} * (2n - i) >$$

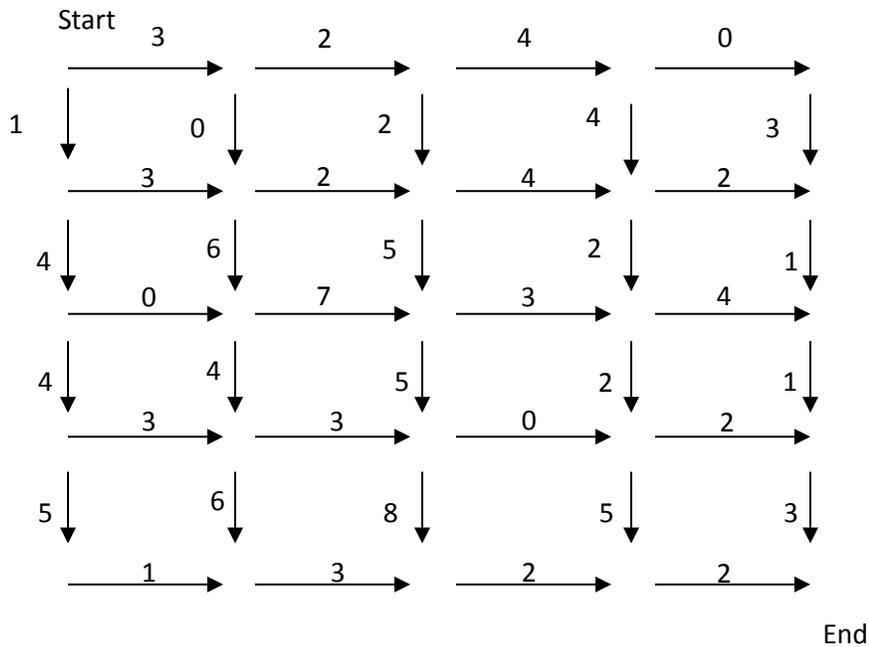
$$n * \sum_{i=0}^n \binom{n}{i} \binom{n}{i} = n * \binom{2n}{n} > 2^{2n} \text{ for } n > 3 \quad // \text{ Stirling's approximation}$$

**This running time is not feasible in practice, so we need to do better. We will use a computing technique called dynamic programming.**

## CS 423: Manhattan Problem (Example of Dynamic Programming)

Manhattan Problem: Find the highest weight path in a weighted grid (graph) from a start node to an end node.

Graph is like blocks in the city – can travel east and south. Each edge on the block has a weight (cost).



C. Can you find the path with the highest weight?

Let's label each vertex as  $[i][j]$  where Start is at position  $[0][0]$  and the one to the right of start is position  $[0][1]$ . The max total cost from the start to vertex at  $[i][j]$  is stored as  $v[i][j]$ .

We'll create a table that contains the highest scoring path to each of the vertices:

**Table of Costs from Start to each Vertex (First row completed):**

0	3	5	9	9

The top row is simply the cost of the path along the top line.

**Table of Costs from Start to each Vertex (Left column and first row completed):**

0	3	5	9	9
1				
5				
9				
14				

The left column is simply the cost of the path along the left line of the graph.

Now, to fill in the rest of the table:

$$v[i][j] = \max \begin{cases} v[i-1][j] + \text{weight of edge from } v[i-1][j] \text{ to } v[i][j] \\ v[i][j-1] + \text{weight of edge from } v[i][j-1] \text{ to } v[i][j] \end{cases}$$

**D. Final Table (complete the entries above in the table)**

**Manhattan Algorithm:**

Input: G is a graph n nodes wide and m nodes tall with weights along edges with start node at [0,0] and end node at [n-1,m-1]

```

v[0][0] = 0
for j = 1 to (n-1): // fill in top row
    v[0][j] = v[0][j-1] + weight([0][j-1] to [0][j])
for i = 1 to (m-1): // fill in left column
    v[i][0] = v[i-1][0] + weight([i-1][0] to [i][0])
for i = 1 to (m-1): // fill in middle cells
    for j = 1 to (n-1):
        v[i][j] = max {
            v[i-1][j] + weight([i-1][j] to [i][j])
            v[i][j-1] + weight([i][j-1] to [i][j])
        }

return v[n-1][m-1] // highest scoring path

```

E. What is the running time of the Manhattan algorithm assuming the graph is n nodes wide and m nodes tall?

O( )



## CS423: Global String Alignment

**Problem:** Given two strings, S and T, what is the alignment with the maximum score and what is the score?

**Example Input:** What alignment is the optimal alignment for the following DNA sequences:

> DNA Sequence 1

ACTGCGATTGACGTACGATCATCGTACGATCATCATGCTGAGCTATCATCATCGTACTGA  
TCGTAGACTACGTAGCTAGCATGCAGTCTGATGACGTCATGCTGACGTAGCATGC

> DNA Sequence 2

GACTAGCAGCGAGAGATCTCTCGAGTATGCGAGAGCTGATGCATCTACGTATGCAGTCGT  
GCTAATGCGAGCGTATACGCGGGCATGTAGAGACTTCCTAGTAC

**Small Example:** Suppose we want to align:

AGCGTTA

ACGTGA

**Table 1: Dynamic programming table for global string alignment**

		A	G	C	G	T	T	A
A								
C								
G					x			
T								
G			y					
A								

x represents the best score for aligning AGCG and ACG

y represents the best score for aligning AG and ACGTG

What does each table entry mean?

The north cell represents: aligning the left sequence's last character with a gap

The west cell represents: aligning the top sequence's last character with a gap

The northwest cell represents: aligning the last character of each sequence together

Let  $c[i,j]$  be the cell entry in the  $i$ th row and  $j$ th column.

$g$  = gap penalty (insertion or deletion)

$m$  = match/mismatch score

**Global Alignment Algorithm:**

```

c[0][0] = 0 // no alignment
for i from 1 to p // fill in gap penalties in left column
    c[i][0] = g*i
for j from 1 to q // fill in gap penalties in top row
    c[0][j] = g*j
for i from 1 to p // fill in table left to right, top to bottom
    for j from 1 to q
        c[i][j] = max(c[i][j-1] + g, c[i-1][j] + g, c[i-1][j-1] + m)

return c[p][q]

```

**Example:**

Let's assume these costs:

$g = -6$ , match = 5, mismatch = -4

We want to align:

AGCGTTA

ACGTGA

**Complete the table (left column done for you):**

		A	G	C	G	T	T	A
	0							
A	-6							
C	-12							
G	-18							
T	-24							
G	-30							
A	-36							

So, the optimal score is 15 for these two sequences. Now, how do we actually get the alignment? We figure out from which direction the maximum value came from in the table.

15 came from  $10 + 5$  at the end, so:

A  
|  
A

Then, 10 came from  $14 - 4$ : (diagonal), so:

TA  
||  
GA

Then, 14 came from  $9 + 5$  (diagonal), so:

TTA

|||  
TGA

Then, 9 came from 4 + 5 (diagonal), so:

GTTA  
||||  
GTGA

Then, 4 came from -1 + 5 (diagonally), so:

CGTTA  
|||||  
CGTGA

Then, -1 came from 5 - 6 (left), so:

GCGTTA  
||||||  
-CGTGA

Then, 5 came from 0 + 5 (diagonal), so:

AGCGTTA  
A-CGTGA

There, we have it! So, not only should we keep track of the table values, we should keep another table that holds from which direction the maximum value came from.

What is the running time for this global alignment algorithm, if S has length N and T has length M?

O( )

Now, you try aligning these with scoring system:  $g = -6$ , match = 5, mismatch = -4

		T	A	T	G	C	T
	0						
T							
G							
A							
C							
A							
G							
T							

What is the optimal alignment score?

What about aligning these two sequences?

```
---TGGTAGATTC--CACGAGATCTACCGAG-TATGAGTAGGGGGAC-GTTCGCT-C-GG
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
GCCT-CTA-ACACACTGCACGAGATCAACCGAGATATGAGTA---ATACAG-CGGTACGGG
```

---

The score for this alignment is 60. But, what if we were really looking for the best substring of the one sequence aligned with the best substring of the second sequence?

Global alignment: align complete strings

Local alignment: align best matching substrings

Best local alignment of sequences above has a score of 105:

```
CACGAGATCTACCGAG-TATGAGTA
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
CACGAGATCAACCGAGATATGAGTA
```



## CS423: Local String Alignment

Given two strings, S and T, what is the maximum alignment among any substrings S' of S and any substrings T' of T?

c – table storing alignment costs

d – table storing alignment directions (F, L, T, D)

Algorithm:

```
c[0][0] = 0           // no alignment
d[0][0] = "F"        // start location of alignment
maxValue = 0         // max value of alignment found
maxRowPos = 0        // to keep track of the location of the maximum alignment
maxColPos = 0
for i from 1 to p    // fill in gap penalties in left column
    c[i][0] = 0
    d[i][0] = "F"
for j from 1 to q    // fill in gap penalties in top row
    c[0][j] = 0
    d[0][j] = "F"
for i from 1 to p    // fill in table left to right, top to bottom
    for j from 1 to q
        c[i][j] = max(0, c[i][j-1] + g, c[i-1][j] + g, c[i-1][j-1] + m) //add 0 for choice to start new
        alignment
        d[i][j] = "F", "L", "T", or "D" – wherever max value came from // "F" is if max is 0
        if(c[i][j] > maxValue)
            maxValue = c[i][j]
            maxRowPos = i
            maxColPos = j
return c[maxRowPos][maxColPos]
```

*Note: The only major differences between global and local alignment is that we can restart the optimal alignment at any time (hence, the 0 as part of the max). Also, the optimal alignment may end at any position in the table, so we find the maximum value of all table entries as the maximum local alignment score. Recovering the alignment follows the same process as global alignment.*

Use the scores:  $g = -6$  match = 5 mismatch = -4

**Table 1: Complete the cost table below following the local alignment algorithm**

		A	G	A	T	C	A	C
	0	0	0	0	0	0	0	0
C	0							
G	0							
A	0							
C	0							
A	0							
G	0							

**Table 2: Complete the direction table below**

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

What is the optimal alignment and cost?

## CS 423: Longest Common Subsequence

Given two strings S and T, what is the longest common subsequence between the two strings?

*(In biology, this is important if we want to find the longest sequence of DNA that is exactly the same and in the same order in two strings.)*

Assume  $S = S[1]S[2]S[3]\dots S[q]$  //note: in python strings start with index 0

Assume  $T = T[1]T[2]T[3]\dots T[p]$

### Algorithm

```
c[0][0] = 0           // no alignment
d[0][0] = "F"        // start location of LCS
for i from 1 to p    // fill in left column
    c[i][0] = 0
    d[i][0] = "F"
for j from 1 to q    // fill in right column
    c[0][j] = 0
    d[0][j] = "F"
for i from 1 to p    // fill in table left to right, top to bottom
    for j from 1 to q
        if T[i] == S[j] // have a match
            c[i][j] = max(c[i][j-1], c[i-1][j], c[i-1][j-1] + 1)
        else
            c[i][j] = max(c[i][j-1], c[i-1][j])
        d[i][j] = "L", "T", or "D" – wherever max value came from
return c[p][q]
```

*Note: this is the global alignment algorithm except we do not penalize gaps and we add one for a match.*

**Table 3: Complete the cost table for LCS**

		A	G	A	T	C	A	C
	0	0	0	0	0	0	0	0
C	0							
G	0							
A	0							
C	0							
A	0							
G	0							

**Table 4: Complete the direction table**

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

What is the longest common subsequence and its length?



## CS423: Global alignment with affine gaps

In biology, a long gap in a string alignment might be due to a single copying error. Instead of penalizing each gap with score  $g$ , the affine gap model has a gap opening score and a gap extension score.

```
ATC--GA
ATCCCGA
```

For the alignment above in the regular model, each gap has an alignment score of  $-6$ , but the  $CC$  may have resulted from a single insertion or a single deletion.

Suppose the gap opening (GO) score is  $-7$  and the gap extension (GE) score is  $-2$ . With the affine gap model, a gap with length 3 has an alignment score of  $(-7 + -2 + -2 + -2) = -13$ .

Input: strings  $S$  and  $T$ , where  $|S| = n$ ,  $|T| = m$

### Algorithm Idea:

Keep track of four tables ( $S$  across top,  $T$  down left side):

$c$  is the best alignment cost (like our cost table earlier)

$e$  is the best alignment cost where the best score is matching char in  $S$  with “-“

$f$  is the best alignment cost where the best score is matching “-“ with char in  $T$

$g$  is the best alignment cost where the best score matches char in  $S$  with char in  $T$

Note: we need the extra tables to extend the gap score correctly.

### Global Alignment Algorithm (affine gap penalty)

Initialize tables:

$c[0][0] = 0$

$c[i][0] = GO + GE * i$  // fill left column,  $i$  from 1 to  $m$

$c[0][j] = GO + GE * j$  // fill top row,  $j$  from 1 to  $n$

$e[i][0] = -inf$  // fill left column,  $i$  from 1 to  $m$

$f[0][j] = -inf$  // fill top row,  $j$  from 1 to  $n$

Fill in tables,  $i$  from 1 to  $m$ ,  $j$  from 1 to  $n$ :

$g[i][j] = c[i-1][j-1] + m$  // score for last char in  $S$  aligned with last char in  $T$

$e[i][j] = \max(e[i][j-1] + GE, c[i][j-1] + GO + GE)$  // max of new gap or extending gap with char in  $S$

$f[i][j] = \max(f[i-1][j] + GE, c[i-1][j] + GO + GE)$  // max of new gap or extending gap with char in  $T$

$c[i][j] = \max(g[i][j], e[i][j], f[i][j])$  // best cost for alignment

Note: reconstructing the alignment -- know from which table max came from.

Note: can use this model with local alignment (just need to look for max value in table)

GO = -7, GE = -2, match = 5, mismatch = -4

c table

		A	G	A	T	T	T	C
	0	-9	-11	-13	-15	-17	-19	-21
C	-9 (top)	-4	-13	-15	-17	-19	-21	-14
C	-11 (top)	-13	-8	-17	-19	-21	-23	-16
A	-13	-6 (g)	-15	-3	-12	-14	-16	-18
G	-15	-15	-1 (g)	-10	-7	-14	-16	-18
A	-17	-10	-10	4 (g)	-5 (e) *	-7 (e)	-9 (e)	-11
C	-19	-19	-12	-5	0	-9	-11	-4 (g)

\* comes from  $\max(-5, -16, -14)$

e table

		A	G	A	T	T	T	C
	-inf							
C	-inf	-18	-13	-15	-17	-19	-21	-23
C	-inf	-20	-22	-17	-19	-21	-23	-25
A	-inf	-22	-15	-17	-12	-14	-16	-18
G	-inf	-24	-24	-10	-12	-14	-16	-18
A	-inf	-26	-19	-19	-5 *	-7	-9	-11
C	-inf	-28	-28	-21	-14	-9	-11	-13

\* comes from  $\max(-19 + (-2), 4 + (-7) + (-2)) = -5$

f table

		A	G	A	T	T	T	C
	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
C		-18	-20	-22	-24	-26	-28	-30
C		-13	-22	-24	-26	-28	-30	-23
A		-15	-17	-26	-28	-30	-32	-25
G		-15	-19	-12	-21	-23	-25	-27
A		-17	-10	-14	-16 *	-23	-25	-27
C		-19	-12	-5	-14	-16	-18	-20

\* comes from  $\max(-21 + (-2), -7 + (-7) + (-2)) = -16$

g table

		A	G	A	T	T	T	C
C		-4	-13	-15	-17	-19	-21	-14
C		-13	-8	-17	-19	-21	-23	-16
A		-6	-17	-3	-21	-23	-25	-27
G		-17	-1	-19	-7	-16	-18	-20
A		-10	-19	4	-14 *	-11	-18	-20
C		-21	-14	-14	0	-9	-11	-4

\* comes from  $-10 + (-4)$  [cost in NW cell in c table + cost of mismatch]

--AGATTTTC  
CCAGA---C  
Matches: 20 GapTop: -11 GapBottom: -13  
Total score: -4



## CS 423: BLAST (Basic Local Alignment Search Tool)

1. Suppose you have just discovered a new gene, whose associated protein is 1000 amino acids in length. You are curious to see if this protein sequence is similar to proteins in other organisms. The database includes 1000 organisms with average genome length of 1M (1,000,000) amino acids.

How long will it take to do the local alignment algorithm that we saw for aligning two DNA sequences? (Note that the local alignment algorithm could be extended to align protein sequences, with 20 characters instead of 4 DNA nucleotides.)

Hmmm, this seems like it would take too long. BLAST to the rescue. BLAST does local alignment heuristically (faster, but not necessarily optimal).

Here is the URL for BLAST: <http://blast.ncbi.nlm.nih.gov/Blast.cgi>

2. BLAST includes nucleotide matching, protein matching, and translated nucleotide matching. We will focus first on protein blast (search protein database using a protein query).

Here's how BLASTp works with protein sequences:

1. Database sequences are pre-tagged and scored
2. Your query string is broken into words of length 6 (by default, can set word length to 2 or 3 in query in algorithm parameters)
3. Similar words to query words meeting a certain threshold are found in the database sequences
4. Extensions to pairs of matches are found (high-scoring segment pairs HSPs)
5. The local alignment between high scoring matches is reported.

Example of BLASTp:

Your query protein:     LWGFAHAYHESKWAAHNQEILTPLV

### Breaking query string into words

The query string is split into words of length 3:

LWG

WGF

GFA

.....

### Scores of words

Each of these words (note: there are 20\*20\*20 distinct words) has a score with other words. Similar words with scores greater than some threshold T (default is 11) are retained for querying. BLOSUM62 matrix is used for alignment scores by default. Look at the row/column of the aligned amino acids in the words to get score. Each number in BLOSUM62 is added together for the total score.

Assume the query sequence has word **LWG**. Calculate similar word scores to **LWG** below.

- a. What is the score of the word **LWG** using BLOSUM62? \_\_\_\_\_
- b. What is the score of the word **IWG** using BLOSUM62? \_\_\_\_\_
- c. What is the score of the word **MWG** using BLOSUM62? \_\_\_\_\_
- d. What is the score of the word **VWG** using BLOSUM62? \_\_\_\_\_
- e. What is the score of the word **LYG** using BLOSUM62? \_\_\_\_\_
- f. What is the score of the word **FWS** using BLOSUM62? \_\_\_\_\_

### Finding similar words

Words scoring  $\geq$  threshold (default 11) are located in the database (note: database is indexed for quick retrieval of locations of each word).

If two word pairs from the query string are found in the database within a certain distance A (A = 40 for protein blasts) from each other, then the pair is determined a "hit".

### Hit extensions and alignment

The hits are then extended in each direction (left and right), aligning it with the query sequence until the alignment score starts decreasing (current version blast also does gaps with penalties, opening penalty of 11 and extension penalty of 1 by default).

Sequences with high alignment scores are reported along with the alignment.

3. Try it: Run BLASTp with an input sequence (example yeast protein YBL105C on Moodle). Or look at Tammy's results.

a. Look at the results. What is the best scoring sequence? \_\_\_\_\_

b. Look at the match with organism *Zygosaccharomyces bailii* CLIB 213. What do you think the + represents between F and Y in the first two aligned AAs?

\_\_\_\_\_

c. In the same match with *Zygosaccharomyces bailii* CLIB 213, how many of the AAs are identical? \_\_\_\_\_ How many are positive? \_\_\_\_\_ How many are gaps? \_\_\_\_\_

4. Now that you have seen how to run a BLASTp query, let's discuss the results. Remember, this algorithm is a heuristic – it finds high-scoring alignments, but sacrifices optimality for speed. So, how do you know if the similarity score is meaningful? What would you determine as a “high-scoring” match in BLAST?

### Need some statistics...

**E value** – expected number of alignments with optimal local alignment score of S or higher in a random database (basically gives us the number of **expected** false positives)

Lower E values are better (means the sequence that BLAST found is less likely to have occurred by chance).

$$E = Kmne^{(-LS)}$$

m = length of query string

n = size of database

K = constant based on extreme value distribution for alignment scores

L = constant based on extreme value distribution for alignment scores

S = alignment score

If S is bigger, what happens to E? \_\_\_\_\_

If m is bigger (query is longer), what happens to E? \_\_\_\_\_

If n is bigger (database is bigger), what happens to E? \_\_\_\_\_

### bit scores

A raw score is just the score computed given the alignment. A bit score is a **normalized** score of alignments (kind of like standard deviation). The bit scores can then be compared across searches.

Let S be the raw alignment score. The bit score S' is defined as follows:

$$S' = (LS - \ln K) / \ln(2)$$

There is a relationship between E and S', just based on mathematical substitution:

$$E = mn2^{(-S')}$$

There is also a relationship with p:

**p** = probability that an alignment occurs by chance against random database with score S or better

$$p = 1 - e^{(-E)} \quad // \text{ e is the natural number}$$

So, the bigger the E (# of alignments by chance), p is \_\_\_\_\_.

The smaller the E, p is \_\_\_\_\_.

BLAST uses the E value since it is easier the reason about the number of expected alignments scoring S or better that would occur by chance rather than the chance of getting an alignment score at least S.

The L and K values are given by the extreme value distribution. There are different distributions for different conditions: affine gap penalties, linear gap penalties, shorter sequences (alignment scores will be smaller), and longer sequences.

Derivation of E:

$$E = kmne^{(-LS)}$$
$$-LS = -lnk - ln2 * S' \quad (\text{from defn of } S')$$

$$E = kmne^{(-lnk - ln2 * S')}$$

$$E = kmne^{-(lnk)} e^{-ln2 * S'}$$

$$E = \frac{kmn}{k} * 2^{-S'}$$

$$E = mn * 2^{-S'}$$

```

# Matrix made by matblas from blosum62.iij
# * column uses minimum score
# BLOSUM Clustered Scoring Matrix in 1/2 Bit Units
# Blocks Database = /data/blocks_5.0/blocks.dat
# Cluster Percentage: >= 62
# Entropy = 0.6979, Expected = -0.5209
  A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V  B  Z  X  *
A  4 -1 -2 -2  0 -1 -1  0 -2 -1 -1 -1 -1 -2 -1  1  0 -3 -2  0 -2 -1  0 -4
R -1  5  0 -2 -3  1  0 -2  0 -3 -2  2 -1 -3 -2 -1 -1 -3 -2 -3 -1  0 -1 -4
N -2  0  6  1 -3  0  0  0  1 -3 -3  0 -2 -3 -2  1  0 -4 -2 -3  3  0 -1 -4
D -2 -2  1  6 -3  0  2 -1 -1 -3 -4 -1 -3 -3 -1  0 -1 -4 -3 -3  4  1 -1 -4
C  0 -3 -3 -3  9 -3 -4 -3 -3 -1 -1 -3 -1 -2 -3 -1 -1 -2 -2 -1 -3 -3 -2 -4
Q -1  1  0  0 -3  5  2 -2  0 -3 -2  1  0 -3 -1  0 -1 -2 -1 -2  0  3 -1 -4
E -1  0  0  2 -4  2  5 -2  0 -3 -3  1 -2 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
G  0 -2  0 -1 -3 -2 -2  6 -2 -4 -4 -2 -3 -3 -2  0 -2 -2 -3 -3 -1 -2 -1 -4
H -2  0  1 -1 -3  0  0 -2  8 -3 -3 -1 -2 -1 -2 -1 -2 -2  2 -3  0  0 -1 -4
I -1 -3 -3 -3 -1 -3 -3 -4 -3  4  2 -3  1  0 -3 -2 -1 -3 -1  3 -3 -3 -1 -4
L -1 -2 -3 -4 -1 -2 -3 -4 -3  2  4 -2  2  0 -3 -2 -1 -2 -1  1 -4 -3 -1 -4
K -1  2  0 -1 -3  1  1 -2 -1 -3 -2  5 -1 -3 -1  0 -1 -3 -2 -2  0  1 -1 -4
M -1 -1 -2 -3 -1  0 -2 -3 -2  1  2 -1  5  0 -2 -1 -1 -1 -1  1 -3 -1 -1 -4
F -2 -3 -3 -3 -2 -3 -3 -3 -1  0  0 -3  0  6 -4 -2 -2  1  3 -1 -3 -3 -1 -4
P -1 -2 -2 -1 -3 -1 -1 -2 -2 -3 -3 -1 -2 -4  7 -1 -1 -4 -3 -2 -2 -1 -2 -4
S  1 -1  1  0 -1  0  0  0 -1 -2 -2  0 -1 -2 -1  4  1 -3 -2 -2  0  0  0 -4
T  0 -1  0 -1 -1 -1 -1 -2 -2 -1 -1 -1 -1 -2 -1  1  5 -2 -2  0 -1 -1  0 -4
W -3 -3 -4 -4 -2 -2 -3 -2 -2 -3 -2 -3 -1  1 -4 -3 -2 11  2 -3 -4 -3 -2 -4
Y -2 -2 -2 -3 -2 -1 -2 -3  2 -1 -1 -2 -1  3 -3 -2 -2  2  7 -1 -3 -2 -1 -4
V  0 -3 -3 -3 -1 -2 -2 -3 -3  3  1 -2  1 -1 -2 -2  0 -3 -1  4 -3 -2 -1 -4
B -2 -1  3  4 -3  0  1 -1  0 -3 -4  0 -3 -3 -2  0 -1 -4 -3 -3  4  1 -1 -4
Z -1  0  0  1 -3  3  4 -2  0 -3 -3  1 -1 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
X  0 -1 -1 -1 -2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2  0  0 -2 -1 -1 -1 -1 -1 -4
* -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4  1

```

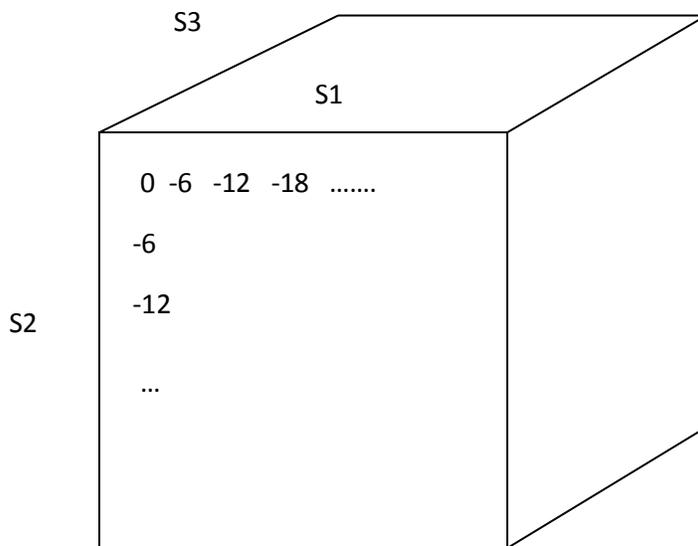


## CS423: Multiple Sequence Alignment

Problem: Given  $k$  strings (such as DNA or proteins), what is the best multiple global alignment and alignment cost?

Here's an example alignment of 3 strings:

```
--T--AC-C-AGT--TATGT-CAGGGGAC
AATTGCCGCC-GTCGT-T-TTCAG----C
-ATTGC-G--ATTCGTAT-----GGGAC
```



**Figure 1: Cost table  $\tau$  for aligning three strings**

Assume  $S1 = a_1a_2a_3\dots a_M$

Assume  $S2 = b_1b_2b_3\dots b_N$

Assume  $S3 = c_1c_2c_3\dots c_P$

We create a cost table  $\tau$  of size  $(M+1) \times (N+1) \times (P+1)$ . [Note: Also need to keep direction table to reconstruct alignment]

1. In the above 3-sequence alignment example, circle an aligned character set where they are all the same.

2. In the above 3-sequence alignment example, put a rectangle around an aligned character set where there are two sequences with a common character and the third sequence has a gap.

3. In the above 3-sequence alignment example, put an arrow toward an aligned character set where there are sequences with mismatched characters.

4. What are the possible character alignments for 3 sequences?

Algorithm for multiple global alignment for three strings (with no affine gaps):

$t[0][0][0] = 0$

Fill in top row with  $i * \text{cost}(a_i, -, -)$ .

The table above shows  $c((a_i, -, -)) = -6$ .

Fill in left column with  $j * \text{cost}(-, b_j, -)$ .

The table above shows  $c(-, b_j, -) = -6$ .

Fill in left column top row (depth of cube) with  $k * \text{cost}(-, -, c_k)$ .

The table above shows  $c(-, -, c_k) = -6$ .

Fill in table cells  $t[i][j][k]$  ( $i, j, k > 0$ ) as:

max of the following 7 values:

$t[i-1][j][k] + c((a_i, -, -))$

$t[i][j-1][k] + c(-, b_j, -)$

$t[i][j][k-1] + c(-, -, c_k)$

$t[i-1][j-1][k] + c((a_i, b_j, -))$

$t[i-1][j][k-1] + c((a_i, -, c_k))$

$t[i][j-1][k-1] + c(-, b_j, c_k)$

$t[i-1][j-1][k-1] + c((a_i, b_j, c_k))$

The best global alignment score for the three sequences is in  $t[M][N][P]$ . To recreate the alignments, use three-dimensional direction table (now each cell can be calculated from one of seven directions).

5. What is the running time of calculating the optimal alignment between 3 sequences of lengths M, N, and P, respectively?

6. Now, what would be the running time this multiple sequence alignment algorithm for 100 sequences, each with length  $\sim N$ ?

7. Is this a practical approach for aligning several sequences?

Go to heuristics...

8. What idea(s) do you have for aligning 100 DNA sequences together in a faster way than the algorithm described above?

Ideas:

Idea 1: Calculate pairwise alignment scores for the K choose 2 sequences, where K = the number of sequences to align. Then combine distances and alignments together.

Example:

Best between 1 and 2:

```
AAAATTTT----  
----TTTTGGGG
```

Best between 1 and 3:

```
----AAAATTTT  
GGGGAAAA----
```

Best between 2 and 3:

```
TTTTGGGG----  
----GGGGAAAA
```

9. What is the best overall alignment between the 3 sequences?

Idea 2:

- a) Calculate pairwise alignments between each pair of sequences.
- b) Perform alignment with most similar pair (highest global alignment score). Lock in this alignment. (So, if there is a gap or match/mismatch in that paired alignment, that will persist in the overall multiple sequence alignment).
- c) For the rest of the (K-1) sequences, align the sequence that is most similar to one of the already aligned sequences in the growing alignment.

This is what the original CLUSTAL algorithm does.

CLUSTAL OMEGA extends CLUSTAL to allow the user to specify an external profile for alignment.



## CS423: UPGMA Algorithm

Problem: Given  $n \times n$  matrix of pairwise relationships between organisms, construct an evolutionary tree.

Suppose you found global alignment scores of a common gene in 11 organisms:

> Yeast YOR020c

mstllksaksivplmdrvlvqrikaqaktasgylpe  
knveklnqaevvavpgftdangnkvvppqkvgdqvl  
ipqfggstiklgnnddevilfrdaeilakiakd

> Neurospora crassa

mattvsvksliplldrvlvqrvkaeaktasgiflpe  
ssvkdlnaekvlavpggaldkdgkrlpimgvnagdrvl  
ipqyggspvkvgeeeeytlfirdseilakiae

> Aspergillus nidulans

msllrnvknlaplldrvlvqrvkpeaktasgiflpe  
svkeqneakvlavpggavdrngqripmgvaagdrvl  
pqfggsplkigeeeyhlfrdseilakine

> Schizosaccharomyces pombe (fission yeast)

matkksaksivplldrvlvqrikadtktasgiflpe  
ksveklsegrvisvgkgynekegklaqpsvavgdrvl  
lpayggsnikvgeeeyslyrdhellaiike

> Mortierella alpina

masritkfsktivpmmdrvlvqrikpqqktasgiyip  
ekaqealnegyvavvgklttqegkvvpselaegdkv  
llppyggsvvkvdneelilfreseilakiq

> Cryptothecodium cohnii

matgiakrftplldrvlvqrikpeaktasgflpesa  
akapnyatvlavpggrtrdgdilpmnvkvgdkwvvp  
eyggmtlkfedeeefqfrdadimgilne

> Drosophila melanogaster

maaaikkiiplmdriliqraealtktkggivilpekav  
gkvlegtlavpggtrnastgnhipigvkegdrvlip  
efggtkvnlegdqkelffresdilakle

> Homo sapiens

agqafrkflplfdrvlversaetvtkggimlpeksq  
gkvlqatvvavsgsgkkggeiqpvsvkvgdvllpe  
yggtkvlddkdyffrdgxilgky

> Geobacillus stearothermophilus

vlkplgdrwievieteektasgivilpdtakekpqeg  
rvvavgkgrvldsgervapevevgdriifskyagtev  
kydgkeylilresdilavig

> Mycobacterium tuberculosis

makvnikpledkilvqaneaettasglvipdtakek  
pqegtvvavpggrwedgekripldaegdtviysky  
ggteikynggeeylilsardvlavvsk

> Mus musculus (house mouse)

magqafrkflldrvlversaetvtkggimlpeks  
qgkvlqatvvavsggkgsgeiepvsvkvgdvllp  
eyggtkvlddkdyffrdsdilgkyvn

**Table 1: Pairwise alignment scores between species**

Yeast	Neur	Asp	Schiz	Mort	Cryp	Dros	Human	Geo	Myco	Mus	
	49	46	78	45	55	54	44	38	37	42	Yeast
		52	41	40	43	46	44	41	39	43	Neur
			43	48	45	45	40	40	38	39	Asp
				42	53	55	41	41	40	40	Schiz
					43	46	40	43	38	39	Mort
						61	43	34	36	45	Cryp
							49	42	36	49	Dros
								37	32	93	Human
									59	38	Geo
										32	Myco
											Mus

Approach: **UPGMA** – **U**nweighted **P**air **G**roup **M**ethod with **A**rithmetic mean to build evolutionary tree

*UPGMA Algorithm:*

First, assume each organism is in its own group

Until there is one group,

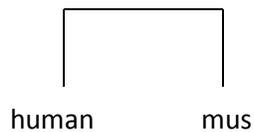
Merge together closest groups (using mean score)

Define closest as highest mean score between groups. Let S be the score between organism i and organism j.

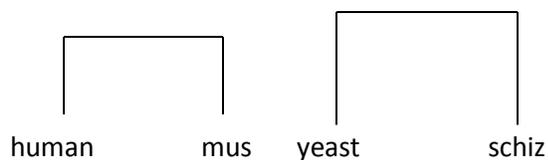
$$\text{mean}(G1, G2) = \frac{\sum_{i,j} S(i,j)}{|G1| * |G2|} \quad \text{where } i \text{ is in } G1 \text{ and } j \text{ is in } G2$$

Example of the algorithm applied to Table 1 data

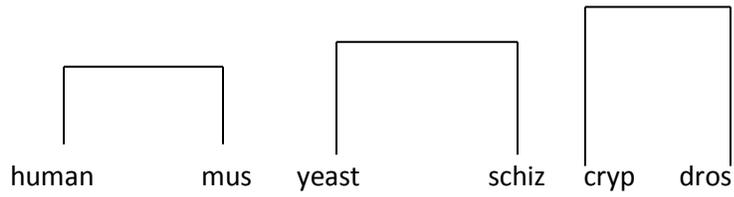
1. Merge human and mus (since maximum score is 93 in the table).



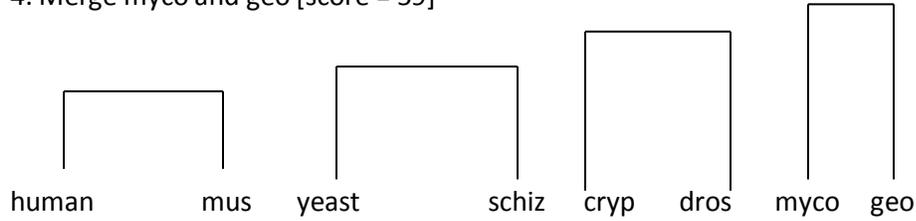
2. Merge yeast and schiz [score = 78]



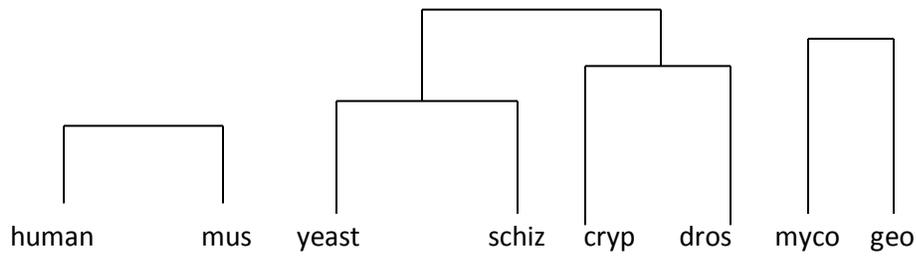
3. Merge cryp and dros [score = 61]



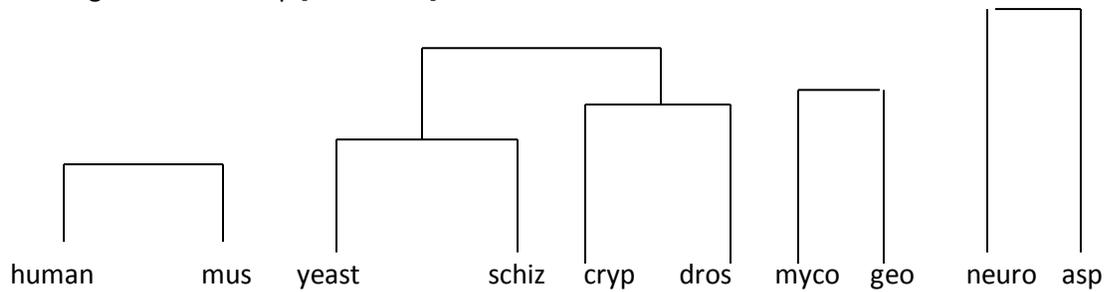
4. Merge myco and geo [score = 59]



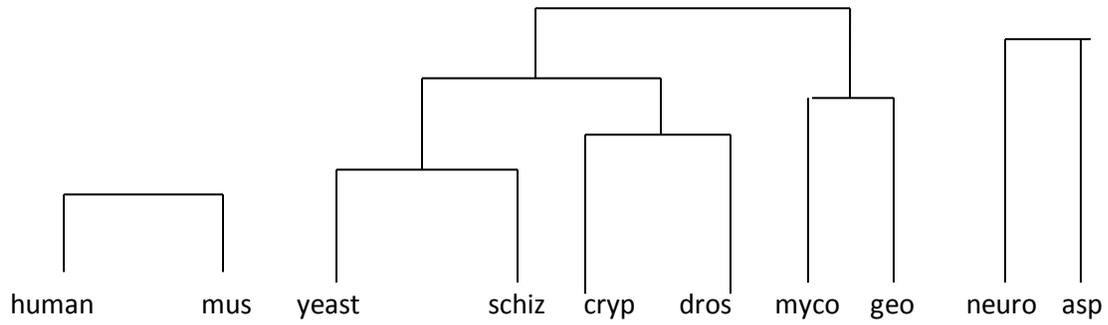
5. Next best score is {yeast, schiz} with {cryp, dros} [mean score =  $(55 + 54 + 53 + 55) / 4 = 54.25$ ]



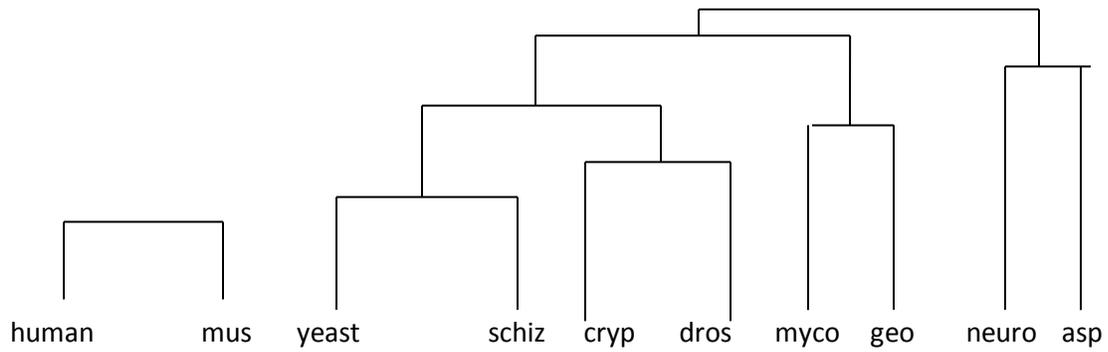
6. Merge nuero and asp [score = 52]



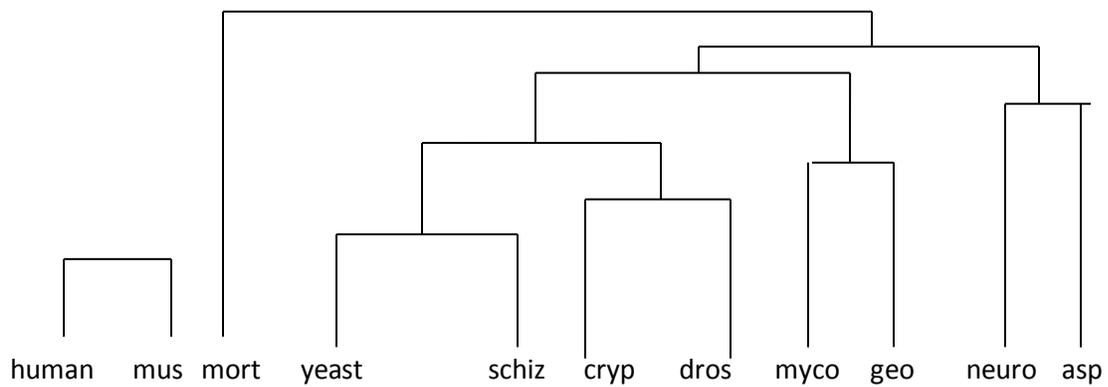
7. Next best is {myco, geo} with {yeast, schiz, cryp, dros} [mean score =  $(37 + 78 + 55 + 54 + 38 + 41 + 34 + 42)/8 = 47.3$ ]



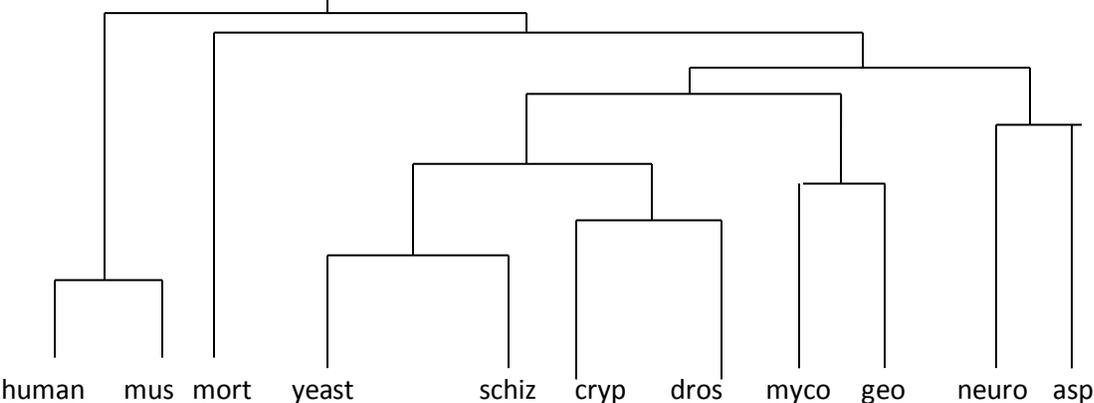
8. Next best is {neuro, asp} with {yeast, schiz, cryp, dros, myco, geo} [mean score = 43.4]



9. Next best is {yeast, schiz, cryp, dros, myco, geo, neuro, asp} with {mort} [mean score = 43.1]



10. Final merge is {human, mus} with {mort, yeast, schiz, cryp, dros, neuro, asp, myco, and geo}.







## CS 423: Phylogenetic Trees Overview

Question 1: Why are phylogenetic (evolutionary) trees helpful to construct?

Words: Phylogenetic trees have several names: guide trees, evolutionary trees, phylograms, cladograms

Draw an example phylogenetic tree here:

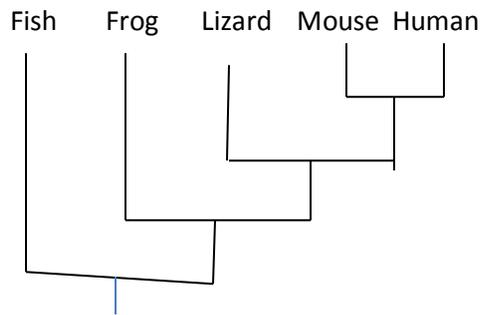
Interpretation:

- Some trees have a **root**; the root is the most recent common ancestor of the organisms in the tree.
- Some trees do not have a root. They just show relationships between organisms. These trees do not make assumptions about common ancestry.
- **Internal nodes** are hypothetical taxonomic units (HTUs).
- **Leaf nodes (terminal nodes)** are operational taxonomic units (OTUs).

Terminology:

- A **clade** is a single common ancestor and its descendants.
- A tree may be scaled, where the length of a branch denotes distance. This is called a **phylogram**.
- A tree may be unscaled, where the lengths of branches do not mean anything. The tree just shows relationships of ancestry. This is called a **cladogram**.

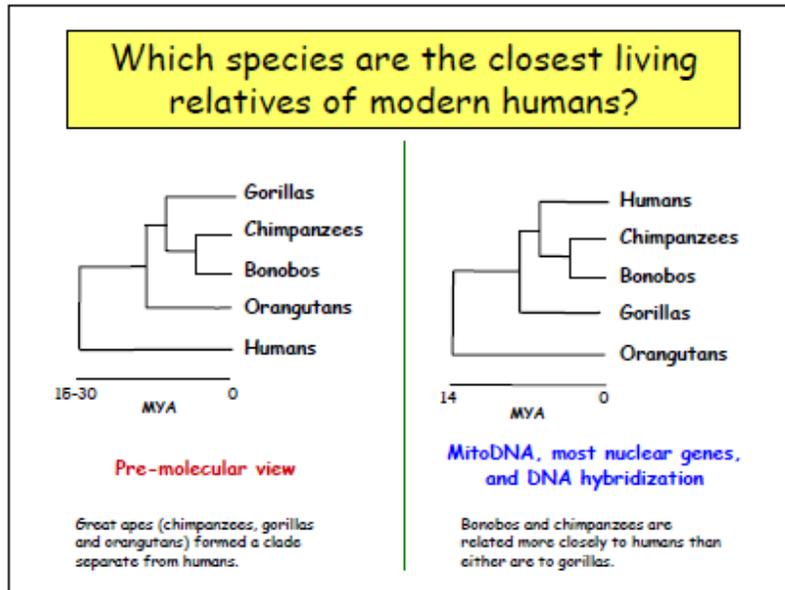
Question 2: Based on the tree below, is the frog more closely related to the fish or the human?



Question 3: Can you draw an equivalent tree to the one in question 2 where the tree looks different?

Question 4: Before bioinformatics, how were evolutionary trees constructed?

Biology example: before genetic data, humans were less related to chimpanzees.





## CS423: Neighbor Joining Algorithm

**Problem:** Given a set of pairwise distances between organisms (also called taxa), find an unrooted tree with the organisms as the leaves of the tree.

**Solution:** UPGMA is one way to solve this problem, but this algorithm creates rooted trees. Another method is called Neighbor Joining (NJ), where at each step an internal node of the tree is created based on minimizing the distance between the joined taxa (groups) and maximizing the distances to all other taxa (groups). Note that in UPGMA, the algorithm seeks to only minimize the distance between the joined groups.

### Algorithm:

Given: a distance matrix  $D$  (containing the pairwise distances between groups),  $N$  is the number of groups

Create a star tree  $T$  with a central hub and all groups connected via an edge to the hub

while  $N \geq 3$ :

Step 1: For each group  $i$ , calculate the net divergence  $R_i$  from  $i$  to all other groups

$$R_i = D_{i1} + D_{i2} + \dots + D_{iN}$$

Note: if  $D_{ij}$  is not in the matrix, look for  $D_{ji}$  instead

Step 2: Calculate a new score matrix  $M$  where  $M_{ij} = D_{ij} - [(R_i + R_j)/(N-2)]$

(Note: this  $M_{ij}$  value is the distance between  $i$  and  $j$  minus the distances to all other groups, so finding the minimum  $M_{ij}$  gives us close distance between  $i$  and  $j$  and far distances to all other groups.)

Step 3: Choose groups  $p$  and  $r$  where  $M_{pr}$  is the smallest. Create an internal node in  $T$  called  $u$  where new edges are added between  $u$  and  $p$  and  $u$  and  $r$ .

The edge distances can be calculated as follows:

$$E_{pu} = (D_{pr} / 2) + [(R_p - R_u)/(2(N-2))]$$

$E_{ru} = D_{pr} - E_{pu}$  // the remaining distance, if either should be negative  
// give the extra length to the other edge

Step 4: Define new distances from new node  $u$  to every other group  $i$

$$D_{iu} = (D_{pi} + D_{ri} - D_{pr}) / 2$$

Step 5:  $N = N-1$

**Example:**

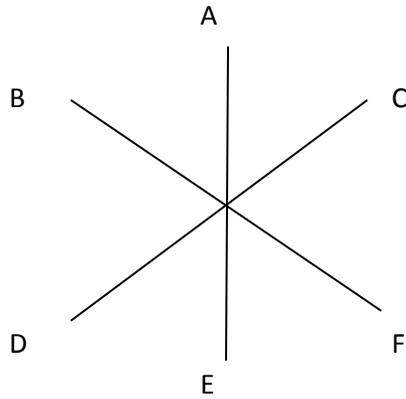
Suppose we have six organisms (or groups or OTUs – operational taxonomic unit) with the following distance matrix:

**Table 1: Distance matrix D for six organisms (smaller numbers mean closer-related organisms)**

A	B	C	D	E	F	
	5	4	7	6	8	A
		7	10	9	11	B
			7	6	8	C
				5	9	D
					8	E
						F

N = 6

T:



First iteration N= 6:

Step 1:

$$R_A = 5 + 4 + 7 + 6 + 8 = 30$$

$$R_B = 5 + 7 + 10 + 9 + 11 = 42$$

$$R_C = 4 + 7 + 7 + 6 + 8 = 32$$

$$R_D = 7 + 10 + 7 + 5 + 9 = 38$$

$$R_E = 6 + 9 + 6 + 5 + 8 = 34$$

$$R_F = 8 + 11 + 8 + 9 + 8 = 44$$

Step 2:

Matrix M. Let's look at the calculation for  $M_{AB}$ :

$$M_{AB} = D_{AB} - (R_A + R_B)/(N-2)$$

$$M_{AB} = 5 - (30 + 42)/4$$

$$M_{AB} = 5 - (72/4) = 5 - 18 = -13$$

**Table 2: Matrix M of scores between groups**

A	B	C	D	E	F	
	-13	-11.5	-10	-10	-10.5	A
		-11.5	-10	-10	-10.5	B
			-10.5	-10.5	-11	C
				-13	-11.5	D
					-11.5	E
						F

Step 3: Choose the smallest M value (-13). We could join groups A and B or D and E. We'll arbitrarily choose A and B to join.

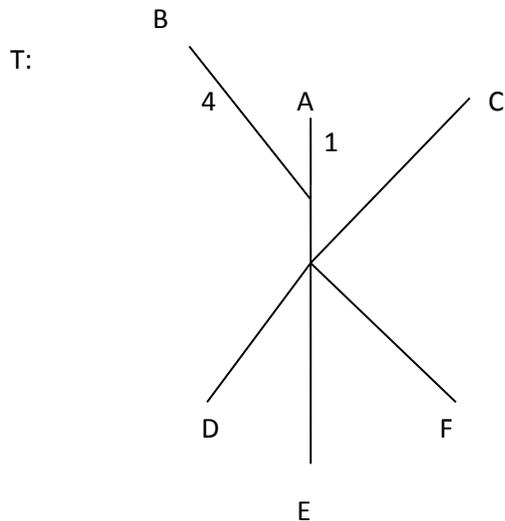
Calculate distance of tree branches:

$$E_{AU} = (D_{AB} / 2) + [(R_A - R_B)/(2(N-2))]$$

$$E_{AU} = (5/2) + (30-42)/(2*4) = 2.5 - 1.5 = 1$$

$$E_{BU} = D_{AB} - E_{AU}$$

$$E_{BU} = 5 - 1 = 4$$



Step 4:

Let U be the group of {A, B}.

General formula for calculating distances from U to other nodes:  $D_{iu} = (D_{pi} + D_{ri} - D_{pr}) / 2$

$$D_{CU} = (D_{AC} + D_{BC} - D_{AB}) / 2 = (4 + 7 - 5) / 2 = 3$$

$$D_{DU} = (D_{AD} + D_{BD} - D_{AB}) / 2 = 6$$

$$D_{EU} = (D_{AE} + D_{BE} - D_{AB}) / 2 = 5$$

$$D_{FU} = (D_{AF} + D_{BF} - D_{AB}) / 2 = 7$$

New matrix D:

**Table 3: Matrix D at the end of the first iteration**

U	C	D	E	F	
	3	6	5	7	U
		7	6	8	C
			5	9	D
				8	E
					F

Second Iteration: N = 5

Step 1:

$$R_U = 3 + 6 + 5 + 7 = 21$$

$$R_C = 3 + 7 + 6 + 8 = 24$$

$$R_D = 6 + 7 + 5 + 9 = 27$$

$$R_E = 5 + 6 + 5 + 8 = 24$$

$$R_F = 7 + 8 + 9 + 8 = 32$$

Step 2:

$$M_{ij} = D_{ij} - [(R_i + R_j) / (N - 2)]$$

**Table 4: Matrix M in second iteration**

U	C	D	E	F	
	-12	-10	-10	-10.67	U
		-10	-10	-10.67	C
			-12	-10.67	D
				-10.67	E
					F

Step 3: Choose the smallest M value (-12). We could join groups C and U or D and E. We'll arbitrarily choose C and U to join.

Calculate distance of tree branches:

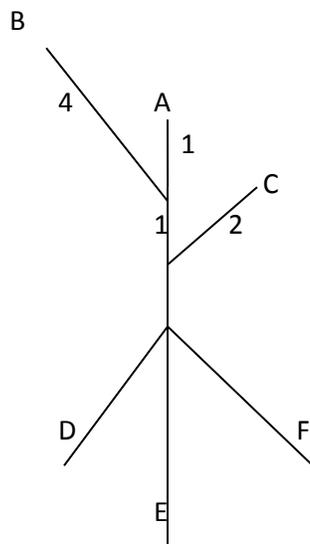
$$E_{CQ} = (D_{CU} / 2) + [(R_C - R_U) / (2(N-2))]$$

$$E_{CQ} = (3/2) + 3/(2*3) = 2$$

$$E_{UQ} = D_{CU} - E_{CQ}$$

$$E_{UQ} = 3 - 2 = 1$$

T:



Step 4:

Let Q be the group of {U, C}.

$$D_{iQ} = (D_{Ui} + D_{Ci} - D_{UC}) / 2$$

$$D_{DQ} = (D_{UD} + D_{CD} - D_{UC}) / 2 = 5$$

$$D_{EQ} = (D_{UE} + D_{CE} - D_{UC}) / 2 = 4$$

$$D_{FQ} = (D_{UF} + D_{CF} - D_{UC}) / 2 = 6$$

New matrix D:

**Table 5: Matrix D at the end of the second iteration**

Q	D	E	F	
	5	4	6	Q
		5	9	D
			8	E
				F

Third Iteration: N = 4

Step 1:

$$R_Q = 5 + 4 + 6 = 15$$

$$R_D = 5 + 5 + 9 = 19$$

$$R_E = 4 + 5 + 8 = 17$$

$$R_F = 6 + 9 + 8 = 23$$

Step 2:

$$M_{ij} = D_{ij} - [(R_i + R_j) / (N - 2)]$$

**Table 6: Matrix M in third iteration**

Q	D	E	F	
	-12	-12	-13	Q
		-13	-12	D
			-12	E
				F

Step 3: Choose the smallest M value (-13). We could join groups F and Q or D and E. We'll arbitrarily choose D and E to join.

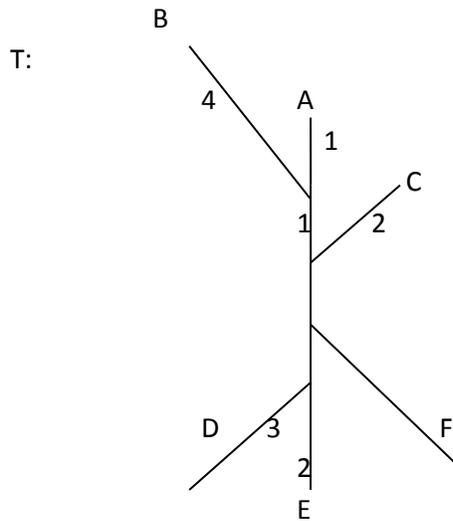
Calculate distance of tree branches:

$$E_{DP} = (D_{DE} / 2) + [(R_D - R_E) / (2(N-2))]$$

$$E_{DP} = (5/2) + (19-17)/4 = 2.5 + .5 = 3$$

$$E_{EP} = D_{DE} - E_{DP}$$

$$E_{EP} = 5 - 3 = 2$$



Step 4:

Let P be the group of {D, E}.

$$D_{iP} = (D_{Di} + D_{Ei} - D_{DE}) / 2$$

$$D_{QP} = (D_{DQ} + D_{EQ} - D_{DE}) / 2 = 2$$

$$D_{FP} = (D_{DF} + D_{EF} - D_{DE}) / 2 = 6$$

New matrix D:

**Table 7: Matrix D at the end of the third iteration**

Q	P	F	
	2	6	Q
		6	P
			F

Fourth Iteration: N = 3

Step 1:

$$R_Q = 2 + 6 = 8$$

$$R_P = 2 + 6 = 8$$

$$R_F = 6 + 6 = 12$$

Step 2:

$$M_{ij} = D_{ij} - [(R_i + R_j)/(N-2)]$$

**Table 8: Matrix M in fourth iteration**

Q	P	F	
	-14	-14	Q
		-14	P
			F

Step 3: Choose the smallest M value (-14). We could join any of the groups. We'll arbitrarily choose to join P and F.

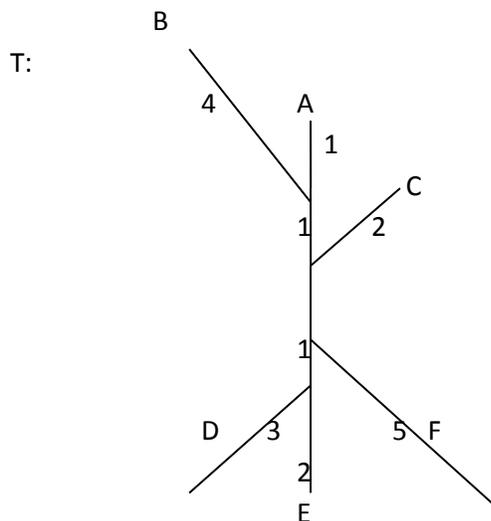
Calculate distance of tree branches:

$$E_{PR} = (D_{PF} / 2) + [(R_P - R_F)/(2(N-2))]$$

$$E_{PR} = (6/2) + (8-12)/2 = 3 - 2 = 1$$

$$E_{FR} = D_{PF} - E_{PR}$$

$$E_{FR} = 6 - 1 = 5$$



Step 4:

Let R be the group of {P, F}.

$$D_{iR} = (D_{Pi} + D_{Fi} - D_{PF}) / 2$$

$$D_{QR} = (D_{PQ} + D_{FQ} - D_{PF}) / 2 = (2 + 6 - 2) / 2 = 3$$

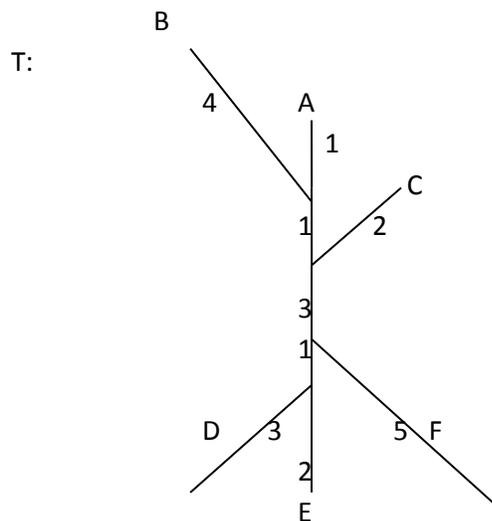
New matrix D:

**Table 9: Matrix D at the end of the fourth iteration**

Q	R	
	3	Q
		R

DONE

Final Tree:



**Table 10: Original Distances**

A	B	C	D	E	F	
	5	4	7	6	8	A
		7	10	9	11	B
			7	6	8	C
				5	9	D
					8	E
						F

Let's see how close the tree is to the original distances:

A and B is 5           //correct  
A and C is 4           //correct  
A and D is 9           //7  
A and E is 8           //6  
A and F is 11          //8  
C and B is 7           //correct  
B and D is 12          //10  
E and B is 11          //9  
F and B is 13          //11  
D and C is 9           //7  
E and C is 8           //6  
F and C is 10          //8  
E and D is 5           //correct  
F and D is 9           //correct  
F and E is 8           //correct

## CS423: Maximum Parsimony for Building Trees

How many different rooted trees are there for 2 organisms?

How many different rooted trees are there for 3 organisms?

How many different unrooted trees are there for 3 organisms?

How many different unrooted trees are there for 4 organisms?

How many possible trees are there if we want to search for the optimal tree showing relationships among organisms?

**Table 1: Number of unrooted and rooted trees given the number of OTUs**

# OTUs (organisms)	# Unrooted trees	# Rooted trees
2	1	1
3	1	3
4	3	15
5	15	105
6	105	945
7	954	10395
8	10395	135135
9	135135	34459425
10	34459425	$2.13 \times 10^{15}$

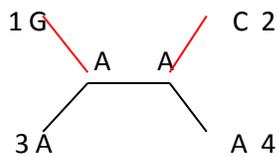
Maximum parsimony method: choose the tree that requires the fewest evolutionary events (substitutions) to explain the sequences at the tree leaves.

Example: Suppose we have a multiple sequence alignment of DNA from 4 organisms (OTUs) and we want to build the maximum parsimony tree.

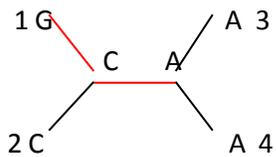
Let's look at the three different trees for site #3 below in the sequences.

Input Seq #	Site #								
	1	2	3	4	5	6	7	8	9
1	A	A	G	A	G	T	T	C	A
2	A	G	C	C	G	T	T	C	T
3	A	G	A	T	A	T	C	C	A
4	A	G	A	G	A	T	C	C	T

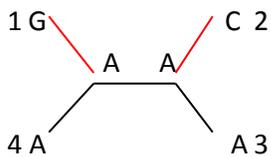
Let's look at site 3 and build the 3 possible trees showing the relationships among the four organisms.



This tree has 2 changes along edges



This tree has 2 changes along edges



This tree has 2 changes along edges

Thus, this site gives no useful information, since no tree is better than the others.

**Maximum Parsimony:**

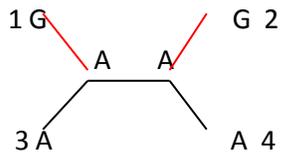
Step 1: Identify informative sites

An informative site is one in which there are at least two different characters at the site, each of which is represented by at least two different sequences.

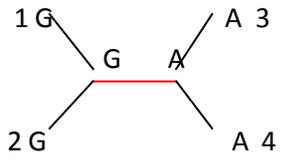
In the example above with the four sequences, which site(s) is/are informative?

Step 2: Build all possible trees for every informative site.

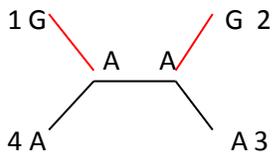
Site 5:



Tree 1 has 2 changes

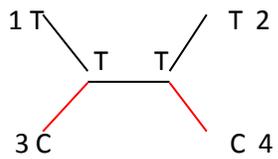


Tree 2 has 1 change

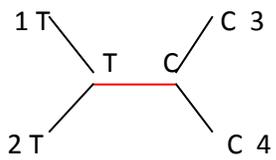


Tree 3 has 2 changes

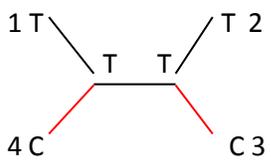
Site 7:



Tree 1 has 2 changes

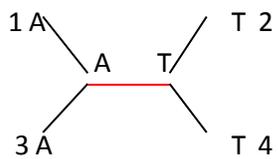


Tree 2 has 1 change

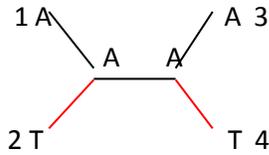


Tree 3 has 2 changes

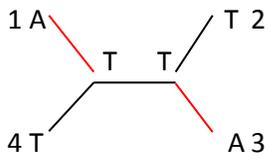
Site 9:



Tree 1 has 1 change



Tree 2 has 2 changes



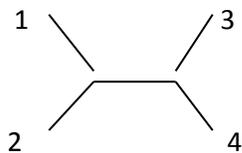
Tree 3 has 2 changes

Step 3: Calculate minimum scoring tree by summing the number of changes for each tree over all sites.

	Site			
	5	7	9	SUM
Tree1	2	2	1	5
Tree2	1	1	2	4
Tree3	2	2	2	6

Tree2 is the best tree for these four sequences.

Output of maximum parsimony algorithm:



Note: If we use maximum parsimony for 5 organisms (OTUs), we would generate the 15 possible trees for each informative site and choose the best tree. At some point, this exhaustive search technique can no longer be applied due to the huge number of possible trees. Then, we can resort to heuristics for throwing away (pruning) bad trees and extending/modifying the good trees.

How can we tell if the tree we get is robust?

Well, in computational biology, we cannot always verify the accuracy of the results of our heuristics. The trees that are created are just hypotheses. One way we can see if a tree is robust is to do the following:

1. Start with original data and original tree produced
2. For 1000 times
  - a. Randomly re-sample sites with replacement
  - b. Build tree with re-sampled data
  - c. Determine if each clade in original tree is observed in the tree from the random re-sampling

3. Calculate frequency with which each clade appears in the trees
4. If frequency > 70%, the clade designation is robust.

example:

Original sequence data:

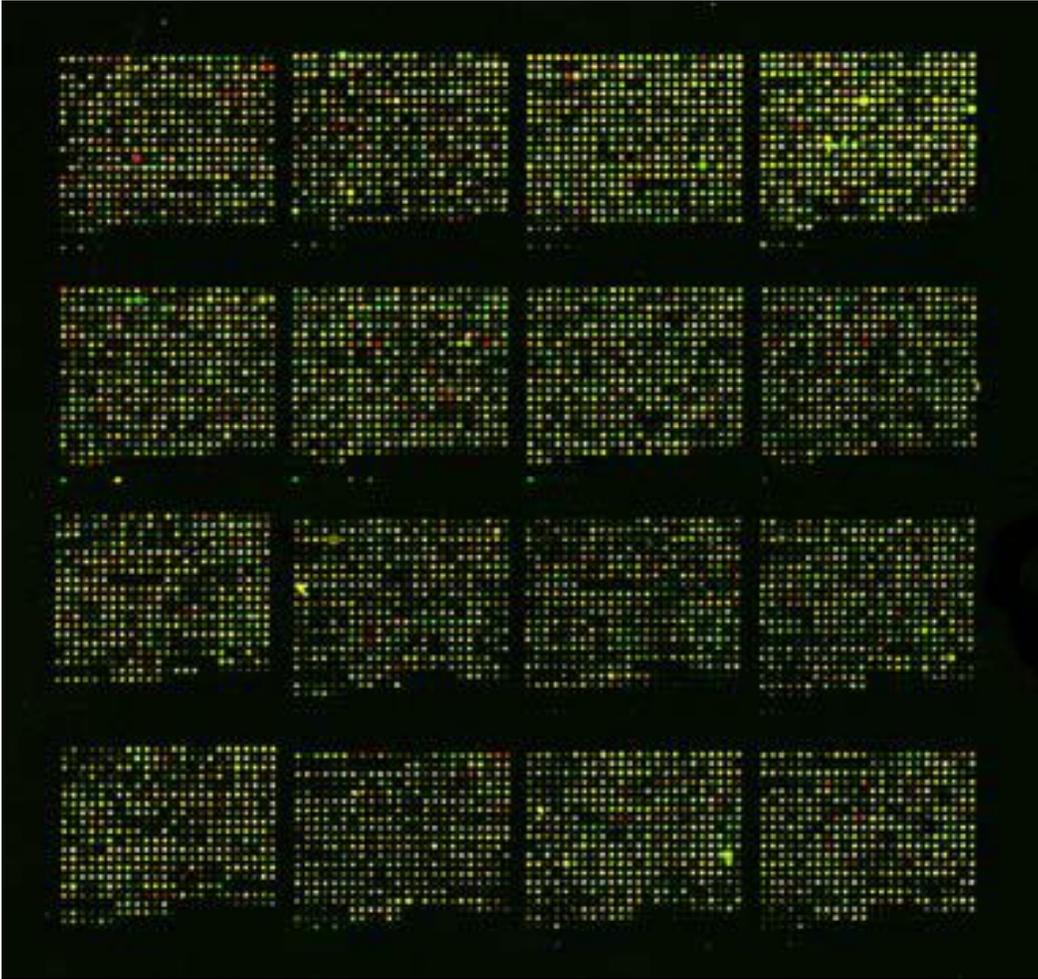
Seq #	Site #								
	1	2	3	4	5	6	7	8	9
1	A	A	G	A	G	T	T	C	A
2	A	G	C	C	G	T	T	C	T
3	A	G	A	T	A	T	C	C	A
4	A	G	A	G	A	T	C	C	T

Re-sampled data with replacement for one of the 1000 trials (note that sites 6 and 7 are sampled twice whereas sites 3 and 9 are not used at all – due to the randomness):

Seq #	Site #								
	5	4	7	6	7	6	2	1	8
1	G	A	T	T	T	T	A	A	C
2	G	C	T	T	T	T	G	A	C
3	A	T	C	T	C	T	G	A	C
4	A	G	C	T	C	T	G	A	C

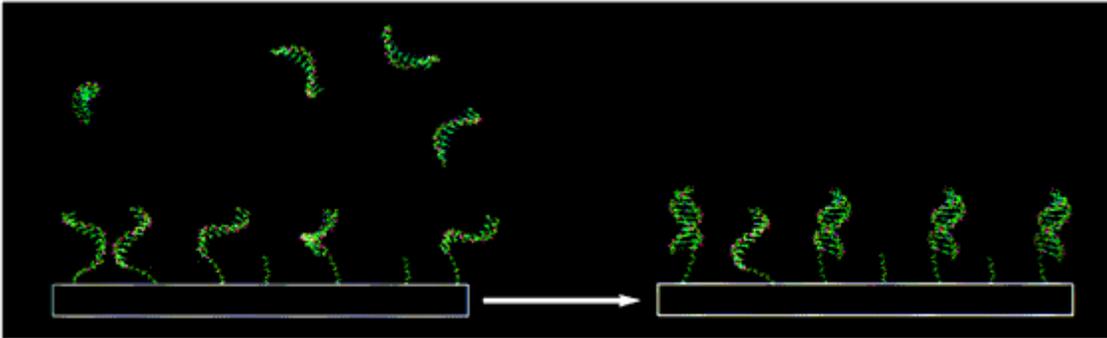


**CS423: DNA Microarrays**



**Figure 1: Yeast genome DNA microarray**

Colors show intensity of hybridization level of each probe (each spot above) to the target mRNA.



**Figure 2: mRNA attaching to probes**

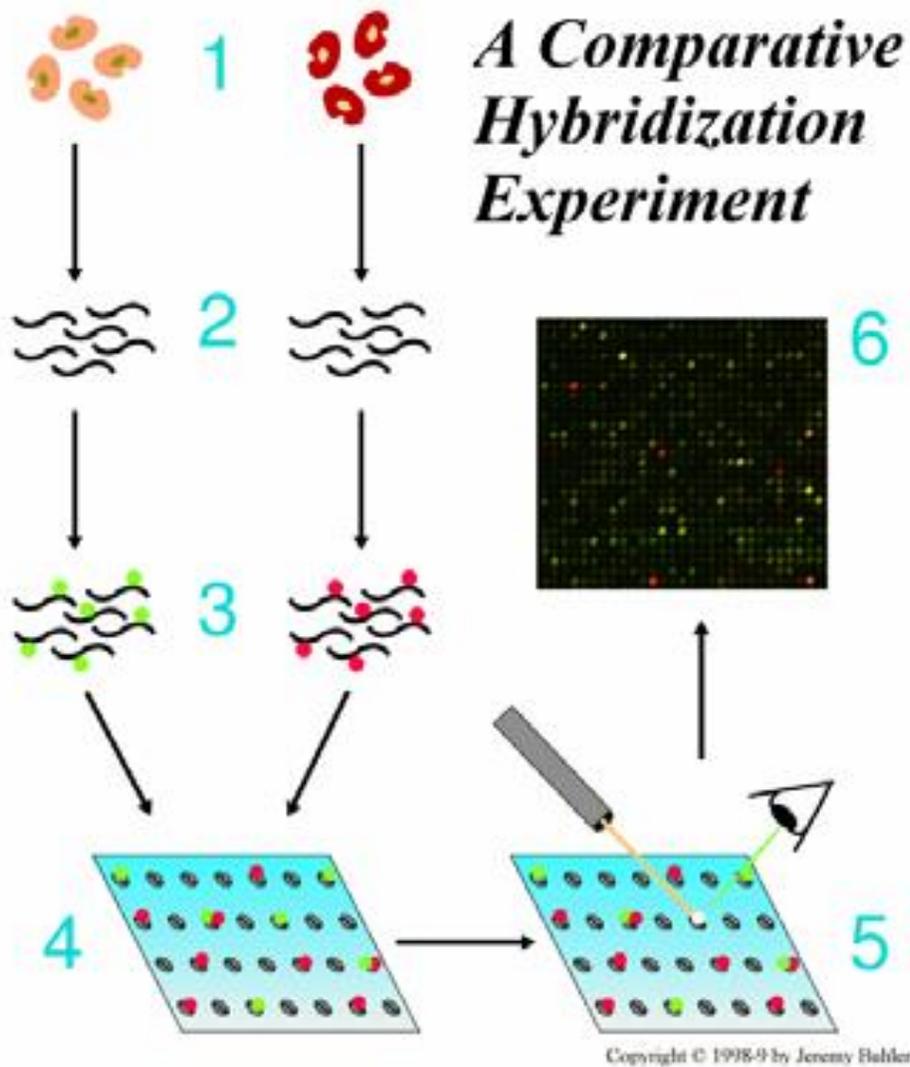


Figure 3: A two-channel experiment (figure by Jeremy Bulter)

Resulting data from M experiments can be organized into a matrix, as follows:

	exp1	exp2	exp3	exp4	... expM
gene1	1.3	0.4	3.2	1.2	2.2
gene2	2.5	2.2	1.1	3.1	2
gene3	.7	0.0	0.2	0.1	4
...					
...					
geneN	1.1	1.4	2.6	2.4	1.7



## CS 423 Activity: Microarray Hybridization

Suppose the following 20-mer target mRNA is washed over the DNA microarray.

5'      GUACACCGUUAACGAUGAAU      3'

Suppose the following cDNA 20-mers are on the microarray. (Remember 3' and 5' ends)

5'      ATTCATCGTTGGCGGTGTAC      3'

How well will the target hybridize to this DNA probe? \_\_\_\_\_

5'      TATCATTGATGGGCCTGTCC      3'

How well will the target hybridize to this DNA probe? \_\_\_\_\_

5'      TAATCCCCAACCCGCACATG      3'

How well will the target hybridize to this DNA probe? \_\_\_\_\_



## CS423: K-means Clustering (Lloyd's Heuristic)

Formal problem: Given N items with known distances between items and K clusters, assign the N items into one of K clusters such that the total distance from each item to its cluster center is minimized.

(Finding the optimal set of clusters takes exponential time, so k-means is a randomized, greedy heuristic to solve this problem more quickly.)

K-means clustering heuristic (Lloyd):

Input: n items, k = number of clusters to create

For each item, randomly assign it to one of the k clusters.

Calculate cluster centers for each cluster.

While there are new cluster assignments:

    For every item i:

        Assign i to the cluster with closest center.

    Re-compute cluster centers for each cluster.

-----

To calculate the square of Euclidean distance from item c to item t:

$$D(\langle c_1, c_2, \dots, c_m \rangle, \langle t_1, t_2, \dots, t_m \rangle) = \sum_i [(c_i - t_i)^2] \quad \begin{array}{l} // \text{ note that we do not need to take sqrt} \\ // \text{ since all distances are positive and this} \\ // \text{ reduces the number of computations} \end{array}$$

To find the center c from a set of items T:

$$\text{center}(T) = \frac{\text{vector sum of items in } T}{|T|}$$

Example k-means clustering:

Suppose our microarray data is the following with k = 2

	exp1	exp2
gene1	0	0
gene2	0	1
gene3	1	1
gene4	1	0
gene5	.5	.5
gene6	5	5
gene7	5	6
gene8	6	6
gene9	6	5
gene10	5.5	5.5

Step 1: Randomly assign each item to one of two clusters:

$$C_1 = \{\text{gene1, gene4, gene5, gene8}\}$$

$$C_2 = \{\text{gene2, gene3, gene6, gene7, gene9, gene10}\}$$

Step 2: Calculate cluster centers:

$$\begin{aligned} \text{Center}(C_1) &= \langle 0, 0 \rangle + \langle 1, 0 \rangle + \langle .5, .5 \rangle + \langle 6, 6 \rangle / 4 \\ &= \langle 7.5, 6.5 \rangle / 4 \\ &= \langle 1.875, 1.625 \rangle \end{aligned}$$

$$\begin{aligned} \text{Center}(C_2) &= \langle 0, 1 \rangle + \langle 1, 1 \rangle + \langle 5, 5 \rangle + \langle 5, 6 \rangle + \langle 6, 5 \rangle + \langle 5.5, 5.5 \rangle / 6 \\ &= \langle 22.5, 23.5 \rangle / 6 \\ &= \langle 3.75, 3.917 \rangle \end{aligned}$$

While no new cluster assignments:

Calculate distance from each point to cluster center to assign point to cluster:

Item	Sq Distance to center $C_1$	Sq Distance to center $C_2$	Cluster assignment
$\langle 0, 0 \rangle$	$1.875^2 + 1.625^2 = 6.15625$	$3.75^2 + 3.917^2 = 29.405$	$C_1$
$\langle 0, 1 \rangle$	3.91	22.57	$C_1$
$\langle 1, 1 \rangle$	1.16	16.07	$C_1$
$\langle 1, 0 \rangle$	3.41	22.91	$C_1$
$\langle .5, .5 \rangle$	3.15	22.24	$C_1$
$\langle 5, 5 \rangle$	21.16	2.74	$C_2$
$\langle 5, 6 \rangle$	28.91	5.90	$C_2$
$\langle 6, 6 \rangle$	36.16	9.40	$C_2$
$\langle 6, 5 \rangle$	28.41	6.24	$C_2$
$\langle 5.5, 5.5 \rangle$	28.16	5.57	$C_2$

Calculate cluster centers:

$$\begin{aligned} \text{Center}(C_1) &= \langle 0, 0 \rangle + \langle 0, 1 \rangle + \langle 1, 1 \rangle + \langle 1, 0 \rangle + \langle .5, .5 \rangle / 5 \\ &= \langle .5, .5 \rangle \end{aligned}$$

$$\begin{aligned} \text{Center}(C_2) &= \langle 5, 5 \rangle + \langle 5, 6 \rangle + \langle 6, 6 \rangle + \langle 6, 5 \rangle + \langle 5.5, 5.5 \rangle / 5 \\ &= \langle 5.5, 5.5 \rangle \end{aligned}$$

Calculate distance from each point to cluster center to assign point to cluster:

Item	Sq Distance to center $C_1$	Sq Distance to center $C_2$	Cluster assignment
<0, 0>	.5	60.5	$C_1$
<0, 1>	.5	50.5	$C_1$
<1, 1>	.5	40.5	$C_1$
<1, 0>	.5	50.5	$C_1$
<.5, .5>	0	50.0	$C_1$
<5, 5>	40.5	.5	$C_2$
<5, 6>	50.5	.5	$C_2$
<6, 6>	60.5	.5	$C_2$
<6, 5>	50.5	.5	$C_2$
<5.5, 5.5>	50.0	0	$C_2$

No new cluster assignments, so STOP. The final assignment is in the above table.



## CS423: QT Clustering

Problem: Given N items, find the best clusters.

Note: In k-means clustering, the value of k is given as input (so the # of clusters to find must be known before the clustering takes place).

QT (Quality Threshold) Clustering is a clustering solution that instead requires as input the maximum diameter (distance between furthest items) for any cluster and finds clusters that do not exceed the maximum diameter.

### QT Clustering Algorithm:

Input: N items, maximum diameter for clusters

S = all N items

While S has items:

    For each item i in set S of items to cluster:

        Build the largest cluster with item i by choosing the nearest item, the next nearest, etc  
        until cluster diameter is exceeded

    Save the cluster with the most items as a true cluster and remove the items from S

-----  
example:

Assume the dataset is the following:

	exp1	exp2
g1	3	4
g2	3	5
g3	6	7
g4	0	2
g5	0	1
g6	2	2

Assume the maximum diameter is 3

S = {g1, g2, g3, g4, g5, g6}

For each item, find the largest cluster:

g1: {(3, 4), (3, 5)}                   //note: (2,2) is too far from (3,5) to be included

g2: {(3, 5), (3, 4)}

g3: {(6, 7)}

g4: {(0, 2), (0, 1), (2, 2)}

g5: {(0, 1), (0, 2), (2, 2)}

g6: {(2, 2), (0, 2), (0, 1)}

Choose cluster with most items:  $\{(0, 2), (0, 1), (2, 2)\}$  and remove these from S.

$S = \{g1, g2, g3\}$

For each item, find the largest cluster:

$g1: \{(3, 4), (3, 5)\}$

$g2: \{(3, 5), (3, 4)\}$

$g3: \{(6, 7)\}$

Choose cluster  $\{(3, 4), (3, 5)\}$  and remove these from S.

$S = \{g3\}$

Choose cluster  $\{(6, 7)\}$ .

Final Clusters:  $\{(0, 2), (0, 1), (2, 2)\}, \{(3, 4), (3, 5)\}, \{(6, 7)\}$

## CS 423: Hierarchical Clustering

Hierarchical clustering uses the process that the UPGMA uses for creating trees. Instead of using alignment scores as distances, we can use any items for which we have pairwise distances (such as microarray data).

Input: distance matrix D of size  $n \times n$  (for  $n$  items to cluster)

Output: a tree where each leaf is an item (so the reader can then choose appropriate cluster membership and number of clusters by looking at the tree)

Algorithm:

1. Form  $n$  clusters, each with item as its own cluster.
2. Construct a tree  $T$  where each item is a leaf node.
3. While there is more than one cluster:
  - a. Find the two closest clusters  $C_i$  and  $C_j$
  - b. Merge  $C_i$  and  $C_j$  to  $C$  so it has all elements from each cluster
  - c. Compute distance from  $C$  to other clusters
  - d. Add new vertex in  $T$  to link clusters  $C_i$  and  $C_j$  [length of vertex to leaf is half the distance from  $C_1$  to  $C_2$ ]
  - e. Remove rows and columns associated with  $C_i$  and  $C_j$
  - f. Add new row and column for  $C$

The way distance is calculated can vary. Among many options are the following:

Option 1: use minimum distance between items in clusters

Option 2: use average distance between all points in each cluster (UPGMA)

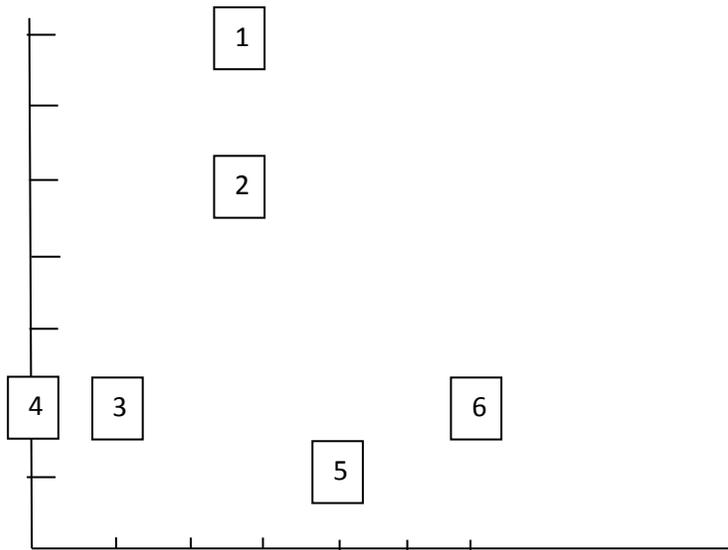
Option 3: can use distance that tries to separate clusters (similar to neighbor joining)

$$\frac{d(C^*, C_1) + d(C^*, C_2) - d(C_1, C_2)}{2} \quad // \text{ when merging } C_1 \text{ and } C_2, C^* \text{ is another cluster}$$

### EXAMPLE:

Suppose we have 6 genes:

	exp1	exp2
G1	3	7
G2	3	5
G3	1	2
G4	0	2
G5	4	1
G6	6	2



Need to calculate the distance matrix (input to hierarchical clustering):

G1	G2	G3	G4	G5	G6	
	2	5.39	5.83	6.08	5.83	G1
		3.61	4.24	3.61	4.25	G2
			1	3.16	5	G3
				4.12	6	G4
					2.24	G5
						G6

Use metric: closest distance between closest elements in each cluster metric

Merge closest: G3 and G4

Update distances

G1	G2	G5	G6	{G3, G4}	
	2	6.08	5.83	5.39	G1
		3.61	4.25	3.61	G2
			2.24	3.16	G5
				5	G6
					{G3, G4}

Merge closest: G1 and G2

Update distances

G5	G6	{G3, G4}	{G1, G2}	
	2.24	3.16	3.61	G5
		5	4.25	G6
			3.61	{G3, G4}
				{G1, G2}

Merge closest: G5 and G6

Update distances

{G3, G4}	{G1, G2}	{G5, G6}	
	3.61	3.16	{G3, G4}
		3.61	{G1, G2}
			{G5, G6}

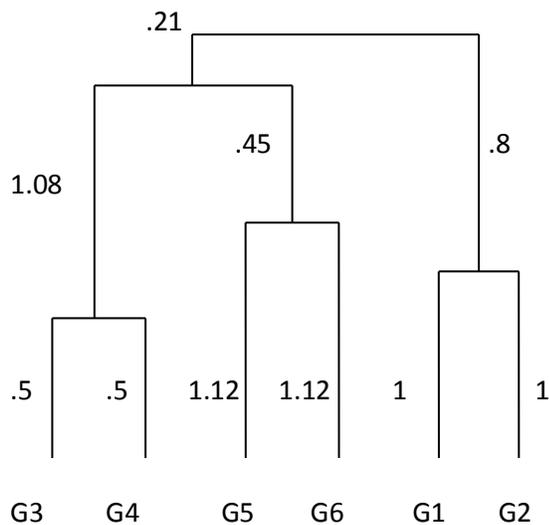
Merge closest: {G3, G4} with {G5, G6}

Update distances

{G1, G2}	{G3, G4, G5, G6}	
	3.61	{G1, G2}
		{G3, G4, G5, G6}

Merge closest: {G1, G2} with {G3, G4, G5, G6}

Overall hierarchical tree:



Note: we could have used other metrics to determine distance between clusters. Now, you can draw a line cutting through the tree to create 1 cluster, 2 clusters, 3 clusters, 4 clusters, 5 clusters, or 6 clusters.



## CS423: Expectation-Maximization to find motifs

Assume we have five sequences for which we believe there is a common 16-mer motif:

```
> Escherichia coli
TTGATTCCCTGAATGCCCGCTTAGTGTAACACTACTGTAACCGGCATTTTCTGCTTTTCC
TGCCGATATTTTTTCTTATCTACCTCACAAAGGTTAGCAATAACTGCTGGGAAAATCCG
AGTTAGTCGTTATATTCTAT
> Haemophilus influenzae
ATCTAACGGTACGGATTCTCCAAAGGCCTATGGAATCTTGTAGAATATGAAACGTTCTAA
TAAATCATAAAGTTGGAGCAAACGCTCGGCATAAGTAGTAAGTGCCGTGCCTCCGCCATT
AGTTACACTAGTGGGACACC
> Vibrio cholerae
ATTTGTGGCGGTTTTCAAATGCTTGGAGAATGGGTACATGATCCGCTTGGCATTGAAGGT
GAGGCTGGCAGCAGCGAAGGTCTGGGGCTGTTTGAACGTTACACGAGTGTAACCGCCGAA
CCATGTTGACACGAATTCTG
> Salmonella typhi
GGTCGGCTTAGACTAGTGTGACCAAAAAGCTTTTGTGCTGAAGTTTCAGGGTAAGAAGAACC
AGCTCCTAGTAAAAAGACTATTGTGACTGAAAAGCGCGTCAGCGCAAAGCCGACCGCACA
AAACGCACAAGGAGTTACAG
> Pseudomonas aeruginosa
ACGCGGCCAGGGTCTTCTCCTGCGAGATCATGCGCGGCGCGCCGCGCATGCCGGCGCCGC
TGCTGGAACGCCTCGACCCCAGGGCTACTACTAGTTTAAACCGGAACGCCGCCAGTGGATCG
GCCTGCCCCAGCTATTGCTC
```

How do we find the motif? If we had a profile, we could easily find the best scoring 16-mer in each sequence. But we also need the profile.

Solution: Expectation-Maximization (EM)

Input: Set of sequences  $S$  thought to have a common motif and  $L$ , the length of the motif

Output: The motif profile and the instances of each sequence that best fit that motif profile.

Algorithm:

1. Create motif instances by randomly guessing the possible locations of the  $L$ -mer motif for each item in  $S$
2. Repeat until convergence:
  - a. Calculate a motif model given motif instances
  - b. For each item in  $S$ , calculate new locations for the highest-scoring motif instances given the model generated in (a)

Example:

Let's run the algorithm on the sequences above.

Step 1: Randomly guess positions of 16-mers in each sequence (these are bold and separated by spaces)

```
> Escherichia coli
TTGATTCCCTGAATGCCCGCTTAGTGTAACACTACTGTAACCGGCATTTTCTGCTTTTCC
TGCCGATATTTTTTCTTATCTACCTCACAAAGGTTAGCAATAACTGCTGGGAA AATTCCG
AGTTAGTCG TTATATTCTAT
> Haemophilus influenzae
A TCTAACGGTACGGATT CTCCAAAGGCCTATGGAATCTTGTAGAATATGAAACGTTCTAA
TAAATCATAAAGTTGGAGCAAACGCTCGGCATAAGTAGTAAGTGCCGTGCCTCCGCCATT
AGTTACACTAGTGGGACACC
> Vibrio cholerae
ATTTGTGGCGGTTTTCAAATGCTTGGAGAATGGGTACATGATCCGCTTGGCATTGAAGGT
GAGGCTGGCAGCAGCGAAGGTCTGGGGCTGTTTGAACGTTACACGAGTGTAACCGCCGAA
CC ATGTTGACACGAATTC TG
> Salmonella typhi
GGTCGGCTTAGACTAGTGTGACCAAAAAGCTTTTGTGCTGAAGTTTCAGGGTAAGAAGAACC
AGCTCCTAGTAAAAAGACTAT TGTGACTGAAAAGCGC GTCAGCGCAAAGCCGACCGCACA
AAACGCACAAGGAGTTACAG
> Pseudomonas aeruginosa
ACGCGGCCAGGGTCTTCTCCTGCGAGATCATGCGCGGCGCGCCGCGCATG CCGGCGCCGC
TGCTGG AACGCCTCGACCCAGGGCTACACTAGTTTAAACCGGAACGCCGCCAGTGGATCG
GCCTGCCCCAGCTATTGCTC
```

Step 2: (Repeat until convergence)

a. Calculate motif profile based on those substrings

<b>A</b>	.40	.20	0.0	.20	.40	0.0	.20	.20	.40	.40	.20	.60	.20	.20	0.0	0.0
<b>C</b>	.20	.40	0.0	0.0	.40	.60	.20	.40	0.0	.40	.20	0.0	.20	0.0	.20	.40
<b>G</b>	0.0	.20	.40	.40	0.0	.40	.40	.40	.40	0.0	.20	.40	.60	.20	.40	.40
<b>T</b>	.40	.20	.60	.40	.20	0.0	.20	0.0	.20	.20	.40	0.0	0.0	.60	.40	.20

b. Find highest scoring instance given this motif profile in each sequence

```
> Escherichia coli
TTGATTCCCTGAATGCCCGCTTAGTGTAACACTACTGTAACCGGCATTTTCTGCTTTTCC
TGCCGATATTTTTTCTTATCTACCTCAC AAAGGTTAGCAATAAC TGCTGGGAAAATTCCG
AGTTAGTCGTTATATTCTAT
> Haemophilus influenzae
ATCTAACGGTACGGATTCTCCAAAGGCCTATGGAATCTTGTAGAATATGAAACGTTCTAA
TAAATCATAAAGTTGGAGCAAACGCTCG GCATAAGTAGTAAGTG CCGTGCCTCCGCCATT
AGTTACACTAGTGGGACACC
```

```

> Vibrio cholerae
ATTTGTGGCGGTTTTCAAATGCTTGGAGAATGGGTACATGATCCGCTTGGCATTGAAGGT
GAGGCTGGCAGCAGCGAAGGTCTGGGGCTGTTTGAACGTTACACG AGTGTAACCGCCGAA
C CATGTTGACACGAATTCTG
> Salmonella typhi
GGTCGGCTTAGACTAGTGTGACCAAAAAGCTTTTGCTGAA GTTTCAGGGTAAGAAG AACC
AGCTCCTAGTAAAAAGACTATTGTGACTGAAAAGCGCGTCAGCGCAAAGCCGACCGCACA
AAACGCACAAGGAGTTACAG
> Pseudomonas aeruginosa
ACGCGGCCAGGGTCTTCTCCTGCGAGATCATGCGCGGCGCGCCGCGCATGCCGGCGCCGC
TGCTGGAACGCCTCGACCCCAGGGCTACACTA GTTTAACCGGAACGCC GCCAGTGGATCG
GCCTGCCCCAGCTATTGCTC

```

a. Calculate motif profile(you complete this)

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

b. Find highest scoring instance given this motif profile in each sequence

```

> Escherichia coli
TTGATTCCTGAATGCCCGCTTAGT GTAACACTACTGTAAC CGGCATTTTCTGCTTTTCC
TGCCGATATTTTTTCTTATCTACCTCACAAAGGTTAGCAATAACTGCTGGGAAAATTCG
AGTTAGTCGTTATATTCTAT
> Haemophilus influenzae
ATCTAACGGTACGGATTCTCAAAGGCCTATGGAATCTTGTAGAATATGAAACGTTCTAA
TAAATCATAAAGTTGGAGCAAACGCTCGGCATAAGTAGTAAGTGCCGTGCCTCCGCCATT
A GTTACACTAGTGGGAC ACC
> Vibrio cholerae
ATTTGTGGCGGTTTTCAAATGCTTGGAGAATGGGTACATGATCCGCTTGGCATTGAAGGT
GAGGCTGGCAGCAGCGAAGGTCTGGGGCTGTTTGAAC GTTACACGAGTGTAAAC CGCCGAA
CCATGTTGACACGAATTCTG
> Salmonella typhi
GGTCGG CTTAGACTAGTGTGAC CAAAAGCTTTTGCTGAAGTTTCAGGGTAAGAAGAACC
AGCTCCTAGTAAAAAGACTATTGTGACTGAAAAGCGCGTCAGCGCAAAGCCGACCGCACA
AAACGCACAAGGAGTTACAG
> Pseudomonas aeruginosa
ACGCGGCCAGGGTCTTCTCCTGCGAGATCATGCGCGGCGCGCCGCGCATGCCGGCGCCGC
TGCTGGAACGCCTCGACCCCAGG GCTACACTAGTTTAAAC CGGAACGCCCGCCAGTGGATCG
GCCTGCCCCAGCTATTGCTC

```

- a. Calculate motif profile  
... keep going until convergence (no different instances are found)



## CS423: Hertz and Stormo (Finding motif profiles and motifs)

Input: Set  $S$  of sequences with a likely common motif,  $L$  the motif length, and  $D$  the number of profiles to retain at each step of the algorithm

Algorithm:

1. Create a singleton profile from one of the  $L$ -mers in one of the sequences.
2. Until each profile has  $k$  members where the size of  $S$  is  $k$ :
  - a. For each of the  $D$  profiles retained, add each possible  $L$ -mer from sequences not already represented in the set of profiles
  - b. Compute the information content of each profile
  - c. Retain the  $D$  profiles with the highest information content

Example input:

Sequences  $S$ :

Seq1: ACTGA

Seq2: TAGCG

Seq3: CTTGC

$L = 4$

$D = 1$

Step 1:

Randomly choose ACTG as the singleton profile.

Profile:

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

Step 2:

- a and b. Add each possible 4-mer from sequences not represented in the profile

Profile with set {ACTG, TAGC}: //try adding substring TAGC from Seq2

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

Info content: (the 0's will make the log undefined, so we'll assume these are positive values close to 0)  
 (We'll assume a background distribution of 25% for each nucleotide.)

= 4

Profile with set {ACTG, AGCG}: // try adding substring AGCG from Seq2

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

Info content: 6

Profile with set {ACTG, CTTG}: //try adding substring CTTG from Seq3

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

Info content: 6

Profile with set {ACTG, TTGC}: //try adding substring TTGC from Seq3

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

Info content: 4

Step c: Keep the top 1 profile (arbitrarily choose set {ACTG, AGCG})

Step 2:

a and b: Add each possible 4-mer from sequences not represented in the profile

Profile with {ACTG, AGCG, CTTG}: //note: we add the first 4-mer from seq3

	1	2	3	4
A	.67	0	0	0
C	.33	.33	.33	0
G	0	.33	0	1
T	0	.33	.67	0

Info content: 4.57

Profile with {ACTG, AGCG, TTGC} //note: we add the second 4-mer from seq3

	1	2	3	4
A	.67	0	0	0
C	0	.33	.33	.33
G	0	.33	.33	.67
T	.33	.33	.33	0

Info content: 2.96

Done, since the profile includes 3 instances, one from each of the three sequences.

Thus, the algorithm found the best motif profile to be:

Profile with {ACTG, AGCG, CTTG}:

	1	2	3	4
A	.67	0	0	0
C	.33	.33	.33	0
G	0	.33	0	1
T	0	.33	.67	0

Info content: 4.57

Consensus sequence for this motif: ACTG or AGTG or ATTG

## CS 423: Markov Chains

Defn: Let  $S$  be a set of states (example: A, C, G, T). Let  $(X_0, X_1, \dots)$  be a sequence of random variables, each with sample space  $S$ . A  $k^{\text{th}}$  order Markov chain satisfies:

$$\Pr(X_t = x \mid X_0, X_1, X_2, \dots, X_{t-1}) = \Pr(X_t = x \mid X_{t-k}, X_{t-k+1}, \dots, X_{t-1}) \text{ for any } t \text{ and any } x \text{ in } S. \text{ (the last } k \text{ steps matter)}$$

0<sup>th</sup> order Markov chain: distribution of  $X_t$  is not dependent on previous state (just like frequencies in motif profiles)

1<sup>st</sup> order Markov chain: distribution of  $X_t$  is dependent on just one previous state  $X_{t-1}$

Etc...

Example:

Let's look at dinucleotide frequencies. Then the probability transition matrix  $M$  for the Markov chain is size  $4 \times 4$  (4 states going to each of the other states).

$$M_{r,s} = \Pr(X_t = s \mid X_{t-1} = r)$$

	A	C	G	T
A	.40	.30	.10	.20
C	.30	.25	.35	.10
G	.10	.30	.20	.40
T	.25	.25	.25	.25

row A, col A: means the probability that the next nucleotide is A given A

row A, col C: means the probability that the next nucleotide is C given A

etc...

So, if this is the Markov chain model  $M$ , then the probability of sequence  $s_1 s_2 s_3 \dots s_t$  being generated by this model is:

$$\Pr(s_1 s_2 s_3 \dots s_t) = \Pr(s_1) * M_{s_1, s_2} * M_{s_2, s_3} * M_{s_3, s_4} * \dots * M_{s_{t-1}, s_t}$$

What is the probability that "ACTCGGA" is generated by this model  $M$ ? Assume the probability of the first nucleotide is 25% for each of ACGT.

$$\Pr(\text{ACTCGGA}) = .25 * .30 * .10 * .25 * .35 * .20 * .10$$





## CS423/BIO423 Review Sheet – Midterm Exam #1

*Content:* Exam 1 will cover ch1, ch2, and ch5 of the textbook (or however far we get through the day before the exam). Material will be drawn from labs 1 - 5, prelabs, lectures, and the textbook.

*Procedure:* The exam will start promptly at 9:15 AM. Please arrive to class on time. You may use one sheet of 8/5" x 11" paper (both sides) during the exam. Other than your sheet of notes, the exam is closed-book, closed-calculator, closed-computer, and closed-notes. All computations will be simple enough for you to do by hand. Tammy will provide the RNA-> amino acid translation table for you during the exam.

*Topics:* This study guide is not a contract – in other words, the exam may not cover every topic listed below and there may be topics that we covered in class that are not explicitly listed. You may want to review the lecture notes, found on Moodle and review your lab assignments.

- Cellular and Molecular Biology (guest lecture, ch.1 of textbook, lab 2)
  - Genomes
  - Mutations
  - DNA (bases: ACTG)
  - RNA (bases: ACUG)
  - Amino Acids (20 different ones) and Proteins
  - Central Dogma
    - Transcription (DNA -> RNA)
      - If there are introns, these are spliced out before translation
    - Translation (RNA -> proteins)
  - Regulatory regions, transcription start/stop sites
  - DNA replication; DNA polymerase
  - Gene expression
  - Eukaryotes (genes have introns/exons, DNA is double helix), ex: yeast
  - Prokaryotes (genes are all exons, circular DNA loop), ex: E.coli
  - Alternative splicing
- Algorithms
  - Types: recursive, exhaustive, branch and bound, dynamic programming, divide and conquer
  - Example: Towers of Hanoi is recursive
  - Example: Merge sort is divide and conquer and recursive
  - Example: Linear search is exhaustive
  - Example: Binary search is branch and bound and recursive
  - Example: Global string alignment uses dynamic programming and is recursive
- Algorithm analysis
  - Big O
  - Determine running time of algorithms
    - Examples: insertion sort ( $O(n^2)$ ), merge sort ( $O(n \lg n)$ ), global string alignment ( $O(nm)$ )
- Python (lab 1)

- Variables, strings
- Printing
- Conditions (if, else, elif), loops (for, while), functions (def)
- Files (input/output)
- Random numbers
- Booleans and comparison, logical operators (or, and, not)
- Lists (like arrays)
- Motifs (lab 3)
  - DNA polymerase might bind to a motif, transcription start/stop sites, promoter signals, intron splice sites
  - Finding instances of known sites:
    - Use profiles (table indicating frequency of nucleotides at each position in motif)
    - Calculate likelihood ratios (Prob that it is a motif / Prob that it is background dist)
    - Log likelihood ratios (Take log of likelihood, so entries can be added in table)
- Mutations/Substitutions/Evolution (lab 4)
  - Molecular clock
  - Sequence similarity
    - Hamming distance
    - p distance
    - Jukes Cantor correction estimate (includes non-observed mutations)
  - PAM and BLOSUM matrices for amino acid substitution rates
- String alignment (lab 5)
  - Determine how similar two sequences are given deletions, insertions, and substitutions
  - Manhattan problem (example of dynamic programming)
  - optimal global alignment (dynamic programming) – find best alignment for two entire strings
  - optimal local alignment (dynamic programming) – find best alignment among any substring of each of two input strings
  - longest common subsequence (dynamic programming) – find longest common subsequence in two strings
  - Affine gap penalty (dynamic programming with four tables)
    - penalty for starting a gap, penalty for extending gap
  - All these run in time  $O(mn)$  where m and n are the lengths of the two input strings

Subject to change: refer to moodle for most up-to-date exam guide

## CS423/BIO423 Review Sheet – Midterm Exam #2

*Content:* Exam 2 will cover material from BLAST to the day before the exam. Material will be drawn from labs 6 - 10, lectures, and the textbook.

*Procedure:* The exam will start promptly at 9:15 AM. Please arrive to class on time. You may use one sheet of 8/5" x 11" paper (both sides) during the exam. You may also **use a calculator** on this exam, since so many of the algorithms require calculations. You may not use your textbook, a computer, or notes other than your notesheet.

*Topics:* This study guide is not a contract – in other words, the exam may not cover every topic listed below and there may be topics that we covered in class that are not explicitly listed. You may want to review the lecture handouts, found on Moodle, and review your lab assignments. You should still know basic molecular biology from the lecture prior to the first exam. However, the focus on this exam is on topics since exam 1.

- BLAST (Basic Local Alignment Search Tool)
  - For proteins: default word size is 3 amino acids
  - E-values
  - bit scores
  - types: blastp (protein to protein), blastn (DNA to DNA)
- Multiple sequence alignment
  - Aligning more than 2 sequences at once
  - One algorithm is extension of dynamic programming, but it gets too slow with several sequences
  - Heuristics: build up multiple alignment by aligning two strings, getting its alignment and then adding another string, etc
  - Clustal omega (tool for multiple sequence alignment and generating trees)
- Phylogenetic Trees (also known as Evolutionary Trees, Guide Trees)
  - Terminology
    - Clade (node and all descendants)
    - Rooted (tree has a root, shows ancestry)
    - Unrooted (tree has no root, just shows kinship among organisms)
    - Cladogram (tree is unscaled, branch lengths do not show distance)
    - Phylogram (tree is scaled, branch lengths show distance)
  - UPGMA – given distance matrix, creates rooted tree
    - Use mean distance as distance between clusters
    - Merge closest clusters at each step
    - If matrix is a distance matrix (where lower numbers are better), then branch lengths can be calculated as the distance between the clusters / 2 for the total length of the branch to the leaves
  - Neighbor Joining – given distance matrix, creates unrooted tree
    - Start with star topology
    - Minimizes intra-cluster distance and maximizes distance to other clusters
    - Join leaves by creating internal node in tree

- Both UPGMA and Neighbor Joining are greedy algorithms and may not produce the optimal tree in terms of minimizing tree branch lengths. But both are fast and run in time  $O(N^3)$ .
- The number of possible trees for N organisms grows really fast, so finding optimal tree is only realistic for  $N < 8$ .
- Maximum Parsimony – given sequences and their alignment, creates unrooted tree
  - Generates all possible trees for all informative sites, actually uses the alignment to construct tree (not just distance scores or alignment scores)
  - Choose tree that has the fewest changes (best explains the differences among the sequences) over all sites
  - Note that this method uses the sequence info and not the distance info (as in UPGMA and Neighbor Joining)
- DNA Microarrays
  - Probes (complementary DNA) on array / Targets (mRNA) washed over array
  - Uses:
    - sick / healthy cells
    - compare species
    - disease detection
    - cell life cycle
    - related cell functional pathways
- Clustering
  - k-means – given microarray data, produce k clusters
    - randomly select initial cluster assignments, calculate cluster center and reassign
    - may not converge to same set of clusters
    - usually fast running time
  - QT – given microarray data, produce clusters with diameter at most d
    - consider maximum clusters for each item, choose largest cluster and remove items from chosen cluster for next iteration
    - always produces same answer
    - greedy, so may not produce optimal clusters
    - usually fast running time
  - Hierarchical clustering – given microarray data, produce a tree
    - Pre-process step includes taking microarray data and generating  $N \times N$  distance matrix
    - Runs just like UPGMA and distance metric can be chosen (mean distance, closest points between clusters, furthest points between clusters, etc)
    - Greedy, so may not produce optimal tree
    - Up to experimenter to determine where to slice branches of tree to produce independent clusters
    - $O(N^3)$
  - Cluster 3.0 and Java TreeView (tools for clustering and viewing clusters)
- Motifs
  - Conserved short regions of DNA, generally thought to be involved in gene expression
  - Expectation Maximization
    - Given sequences, randomly sample sequences to create profile, then use profile to find best instances (until no changes are made)
  - Gibbs Motif Sampler (on-line tool for lab 10)
  - Hertz-Stormo Algorithm

- Keep adding best l-mer from sequences not already represented in profile
- Markov chains (if we get to this topic)
  - Example: dinucleotide frequencies (frequency of nucleotide at position N depends on nucleotide at position N-1)

Subject to change: refer to moodle for most up-to-date exam guide