Midterm Notes

Material

Molecular Biology

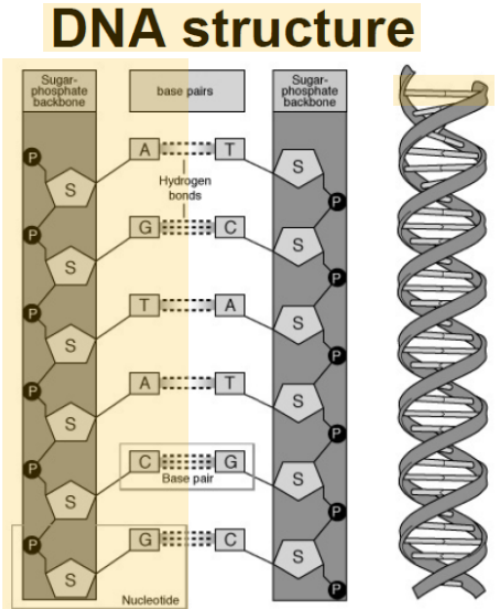
Cells:

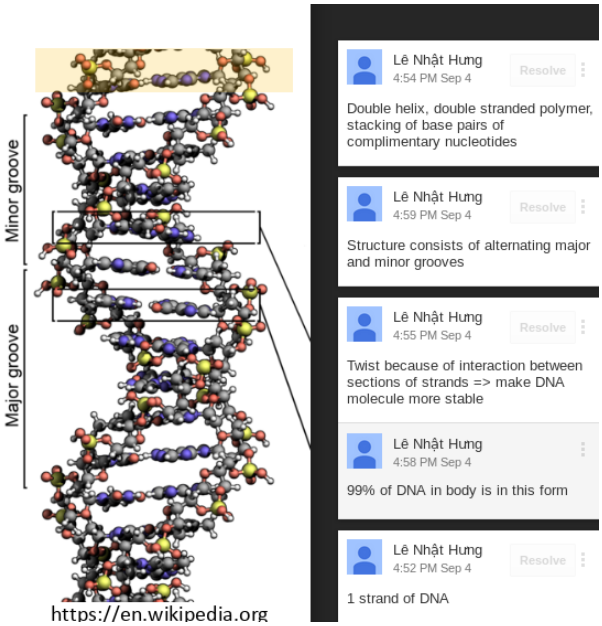
- bags, fundamental working units of all living organisms
- each contains **genome** i.e. blueprint of all cellular structures and activities
- are of diff types (blood, skin, nerves) but all originate from **one cell: the fertilized egg**

Metazoa: mult cell organisms e.g. humans w trillions of cells

Protozoa: unicellular orgs **with no nucleus** e.g. bacteria

Eucaryote (e.g. humans) vs prokaryote: eukaryotic genes have introns, prokaryotic genes don't





DNA structure

- A **deoxyribonucleic acid** or **DNA** molecule is a double-stranded polymer composed of four basic molecular units called nucleotides.
- Each **nucleotide** comprises
  - a phosphate group;
  - a deoxyribose sugar;
  - one of four nitrogen bases:
    - purines: **adenine (A)** and **guanine (G)**,
    - pyrimidines: **cytosine (C)** and **thymine (T)**.

DNA structure

- Polynucleotide chains are **directional** molecules, with slightly different structures marking the two ends of the chains, the so-called **3' end** and **5' end**.
- The 3' and 5' notation refers to the numbering of carbon atoms in the sugar ring.
- The 3' end carries a sugar group and the 5' end carries a phosphate group.
- The two complementary strands of DNA are **antiparallel** (i.e, 5' end to 3' end directions for each strand are opposite)

Genomes

The *genome* of a cell is the entirety of its DNA content.

A genome is made of one or more *chromosomes*: contiguous piece of double-stranded DNA

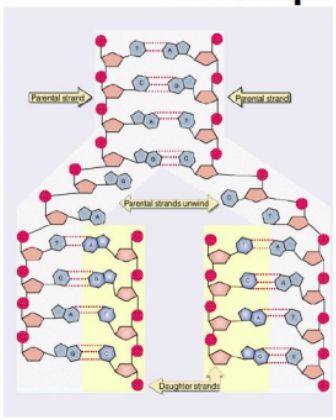
In bacteria (prokaryotes):

- One circular circular chromosome (2-10 Mb)
- Some small chromosomes called plasmids

In human (eukaryote):

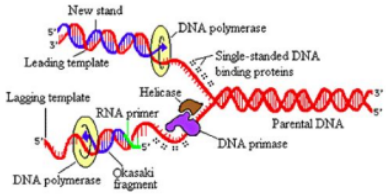
- 23 pairs of chromosomes = 22 autosomes pairs + 1 pair of sex chromosome (XX or XY)
- Each chromosome is 50 – 250 Mb
- Total genome size: 3,000,000,000 bp
- Total length of DNA in one nucleus: 2 meters!

DNA replication



Base pairing provides the mechanism for DNA replication.

DNA replication



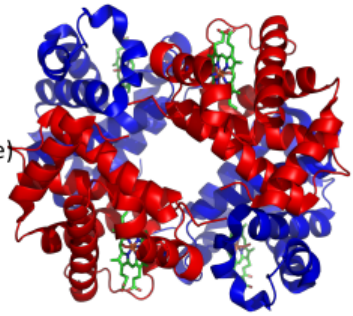
Collaboration of Proteins at the Replication Fork

Useful video: https://www.youtube.com/watch?v=TNK

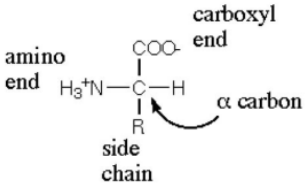
Proteins

Proteins are molecules that perform a huge diversity of functions in the cell:

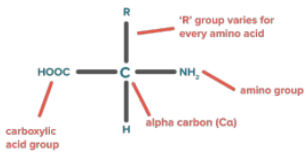
- Structure (actin, tubulin)
- DNA replication (DNA polymerase) + repairs
- DNA transcription (DNA transcriptase)
- Transport of small molecules (hemoglobin)
- Signaling (kinases)
- Regulation (transcription factors)
- Catalyze reactions (enzymes)
- Etc. etc.



Amino acids



Amino acids



b) Name and describe briefly the three main biochemical steps that lead to the production of a protein from a eukaryotic gene.

1)Transcription, which transcribes the DNA of a gene into pre-messenger RNA

2)Splicing, which removes introns, to obtained the mature messenger RNA.

3)Translation, which converts the mRNA to a protein sequence.

c) In protein-coding regions of genomes, why are insertions and deletions of 3 consecutive nucleotides more frequently observed than those of 1 or 2 nucleotides?

Because indels of size that is not a multiple of 3 result in frame shifts, so that all codons downstream are affected. This means that the amino acid encoded by the sequence downstream of the indel are likely to be completely changed. Indels of length 3 affect only one or two amino acids.

d) If only 16 amino acids existed (instead of 20), what would be the minimal length of codons? Why? Watch out: that's a trick question!

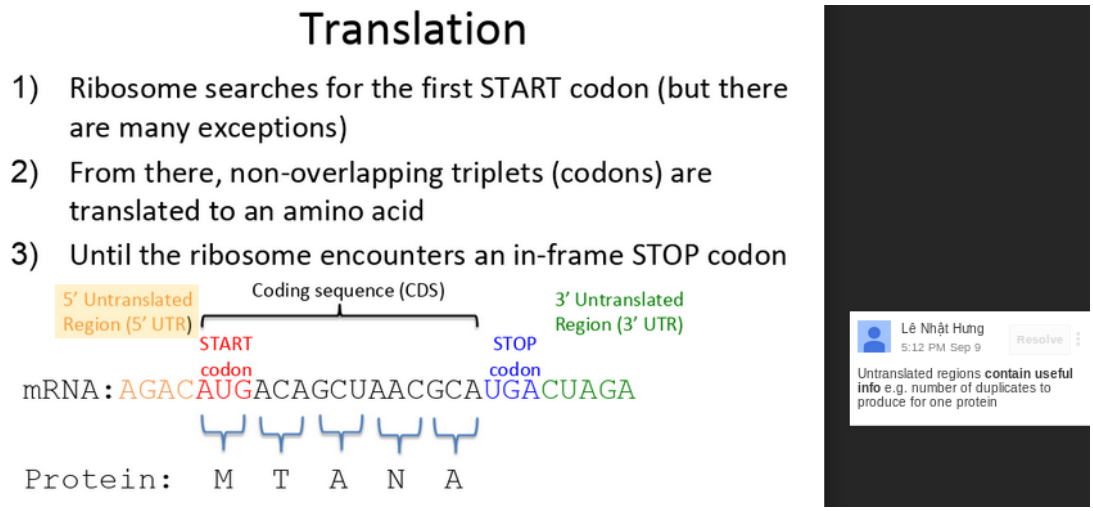
3. You'd need 16 codons to encode the 16 amino acids, plus one for the stop codon. So you'd need ceiling ($\log_4(16+1)$) = 3 nucleotides per codon.

e) Give one advantage and one disadvantage of using a Blast seed of size w=11 versus a blast seed of size w=12.

Advantage of w=11: Better sensitity.
Disadvantage of w=11: Higher running time.

f) What is the expected length of an open reading frame (ORF) in a random sequence where the nucleotides are chosen independently with probability $p_A = p_T = 1/3$, $p_C = p_G = 1/6$? Recall that the three stop codons are TAA, TAG, and TGA.

*Probability of a stop codon = 1/3*1/3*1/3 + 1/3*1/3*1/6 + 1/3*1/6*1/3 = 2/27. So the expected length of an ORF is 1 / (2/27) = 27/2 = 13.5 codons = 40.5 nucleotides.*



Sequence Alignment

Goal: compare 2 biological seqs, to

- measure similarity
- highlight corresponding regions
- infer evolutionary history of 2 seqs derived from same ancestor

Mutations are random. Types of **mutations:**

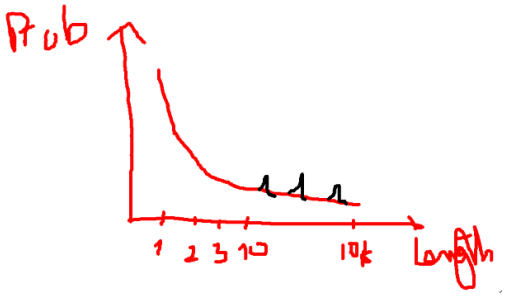
1. Substitutions
2. Insertion
3. Deletion

Types of nucleotide **substitutions**

1. **Transitions:** happen more often
   - A <-> G
   - C <-> T
2. **Transversions**
   - A → C,T
   - C → A,G
   - G → C,T
   - T → A,G

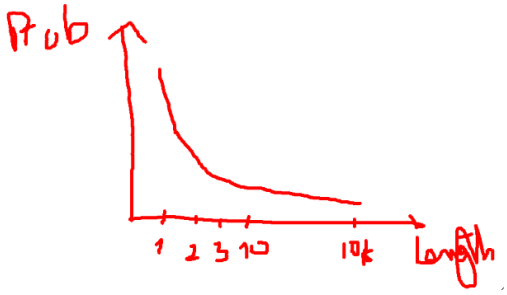
Insertion: insert n new nucleotides b/w 2 existing ones

Prob depends on insertion length



Spikes are due to **transposants:** organisms inserting nucleotides into seq

Deletion:



No peaks

Rates:

- transitions = 2.5 * transversions = 10 del of size 1 (N=1) = 10 ins (N=1)

Pairwise Sequence Alignment

Global alignment: Needleman-Wunsch

X(i,j) = score of optimal alignment of s1...si vs t1...tj

W want X(m,n) = score of opt aln S vs T

Initialization: X(i,0) = b*i, X(0,j) = b*j

Best aln (s1...si, t1...tj) = max(best(s1...si-1, t1...tj-1) + [si, tj] = X(i-1, j-1) + cost matrix M(si, tj),

best(s1...si-1, t1...tj) + [si, -] = X(i-1, j) + b,

best(s1...si, t1...tj-1) + [-, tj] = X(i, j-1) + b)

Filling out table

X_(m+1 x n+1) =		.	A	G	T
	.	0	-2	-4	-6
	A	-2	1	-1	-3
	C	-4	-1	0	-2
	A	-6	-3	-2	-1
	T	-8	-5	-3	-1

Recover alignment **from right to left**

Arrow to left ⇒ col's nucleotide is aligned with gap

Arrow to top ⇒ row's nucleotide is aligned with gap

2 alignments:

```
Red: ACAT
     AG-T
Blue: ACAT
      A-GT
```

Running time: O(n*m) to fill table, O(n+m) for traceback ⇒ total O(n*m)

Space: O(m*n)

2) Pairwise sequence alignment (20 points)

We have studied the Needleman-Wunsch dynamic programming algorithm that finds the optimal alignment between two sequences, given a substitution cost matrix M and a gap penalty scheme. We have studied two types of gaps penalties:
(i) linear gap penalty where the cost of an insertion or deletion of size n is cost(n) = a * n,
(i) affine gap penalty where the cost of an insertion or deletion of size n is cost(n) = a * n + b.

Write a polynomial-time dynamic programming algorithm that is able to find the optimal alignment for any given *arbitrary* convex indel cost function (i.e. a gap cost function cost(n) such that cost(n+ n') <= cost(n) + cost(n') for all n,n' >= 0; an example would be cost(n) = log(n)).
Hint: The running time of the algorithm should be $O(n^3)$.

We use a variant of the NW algorithm. Consider sequences S=s1…sm, T = t1…tn. Let X(i,j) be the score of the optimal alignment between prefixes of length i and j of S and T resepectively. Then,

$X(i,0) = cost(i)$ *for all* $i = 1…m$
$X(0,j) = cost(j)$ *for all* $j = 1…n$

For $i=1…m$,
  For $j= 1…n$
    $X(i,j) = max \{$
      $X(I,j) + M(si,sj),$
      $max_{g \text{ in } \{1…i\}} \{ X(i - g , j) + cost(g)\}$
      $max_{g \text{ in } \{1…j\}} \{ X(i , j - g) + cost(g)\}$
      $\}$

Then the algorithm proceeds as for NW…

Multiple Sequence Alignment (MSA)

Scoring a MSA: **sum-of-pairs score**

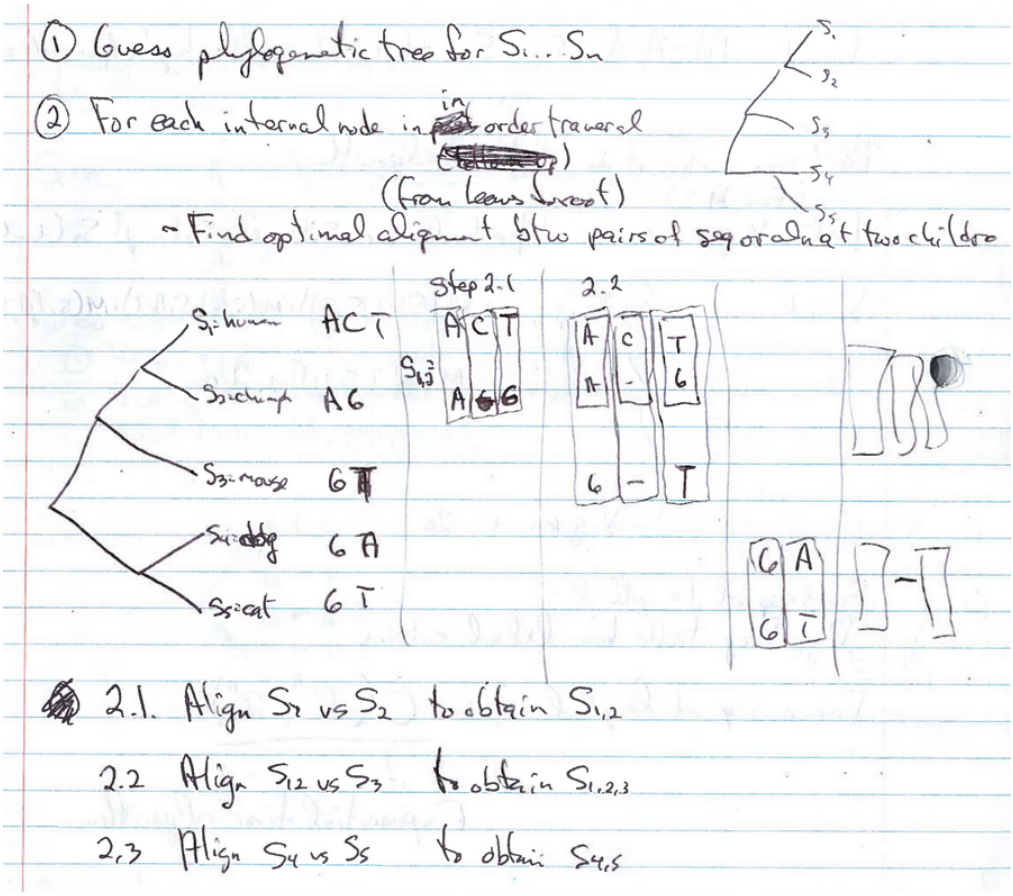
```
S1: AC-T
S2: AG-T
S3: CGAT
S4: CAAT

Ignore pairs of gaps
score(AC-T, AG-T) -> score(ACT, AGT)
```

Not making alignments, just scoring current alignments

$$\text{score} = \sum_{i,j} \text{score}(\text{Aln}(S_i, S_j))$$

Progressive Sequence Alignment:



Finally, score alignments using **sum of pairs score**

3) Multiple sequence alignment (20 points)
The algorithm seen in class for progressive multiple alignments aims at optimizing the sum-of-pairs score. Suppose that we are instead in the following problem:

Maximum Parsimony Multiple Alignment Problem
Given: A set of sequence S1,…,Sn, and a tree T with one leaf per sequence
Find: The alignment of S1,…,Sn such that the parsimony score of the alignment on tree T is minimized.

Here, for the purpose of computing the parsimony score of a given alignment column, we consider a gap as a fifth character, other than A, C, G, and T.

Explain how to modify the progressive alignment algorithm seen in class to adapt it to the Maximum Parsimony Multiple Alignment Problem.

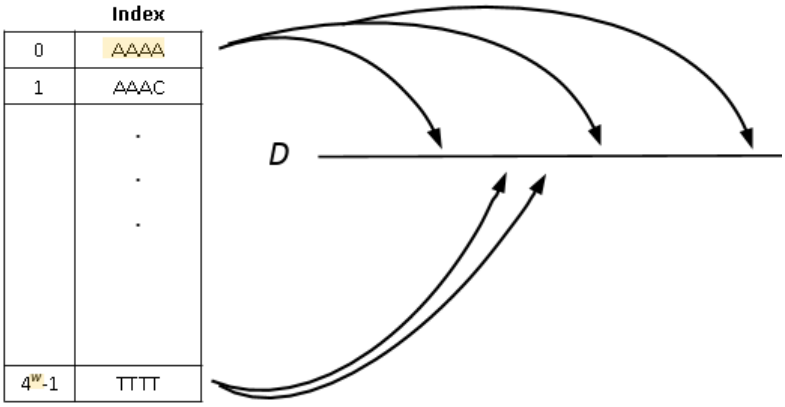
The only change that is that when computing the alignment for node u in the tree, we score the matching of one set of aligned nucleotides (coming from the left subtree of u) to another set of aligned nucleotides (coming from the right subtree of u), one use calculates the parsimony score rather than the sum-of-pairs score.

Fast Alignment Heuristics

Local alignment: BLAST

Query q and database D

1. Database indexing



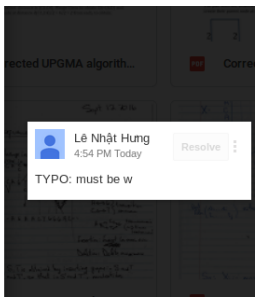
2. Scan for hit in D

• Given a query, *q*

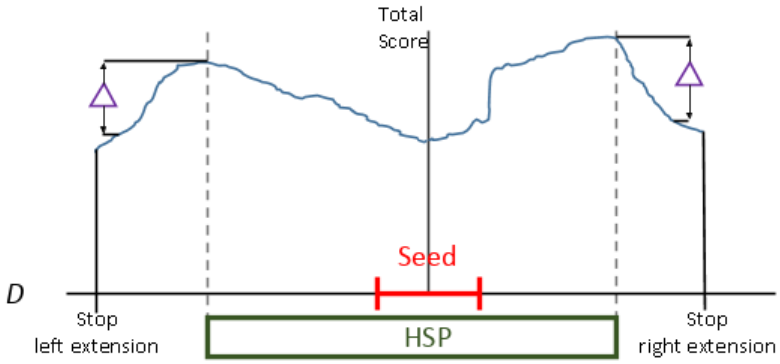
For each *w*-mer in q — O(|q|)
For each matches of q in *D* — O(w)
Investigate match further…

How many hits do we expect for a *w*-mer of size 11?

$$\frac{3\times10^9}{4^{11}} = 1000$$



3. Ungapped extension



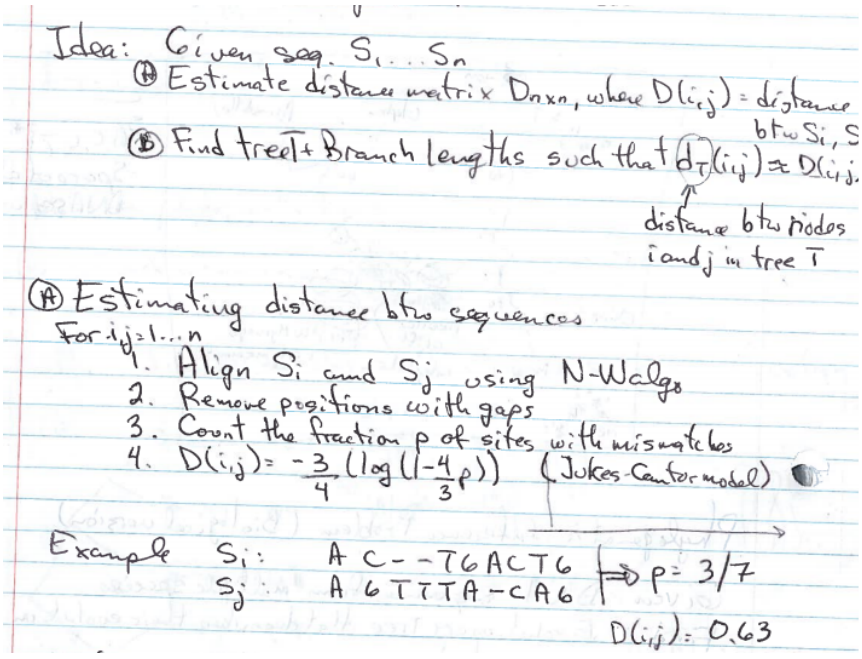
Time? Linear in size of extension

4. If score of HSP in ungapped extension, feed HSP in gapped extension

Phylogenetic Inference

Distance based

Estimate distance matrix D:



Alg: UPGMA

Given: a distance matrix D with n species:

1) Initialize n clusters, $C_1,\ldots,C_n$, each with a single species in it. Create a leaf node for each of the clusters.

2) Define the distance between two clusters as the average pairwise distance between members of the two clusters:

$$d(C_i, C_j) = \frac{\sum_{a \in C_i} \sum_{b \in C_j} D(a,b)}{|C_i| * |C_j|}$$

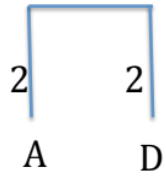
3) Repeat:

3.1 Pick the two clusters C_i and C_j such that d(C_i,C_j) is minimized.
3.2 Create a new cluster $C_k = C_i \cup C_j$
3.3 Create a new node in the tree, make it the parent of nodes i and j, at height d(C_i,C_j)/2.
3.4 Add cluster C_k to the list of clusters, and remove clusters C_i and C_j.

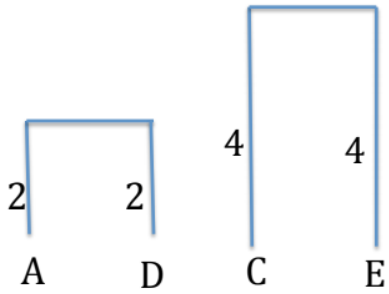
Example: Consider the following distance matrix D:

	A	B	C	D	E
A	-	16	16	4	16
B		-	10	16	10
C			-	16	8
D				-	16
E					-

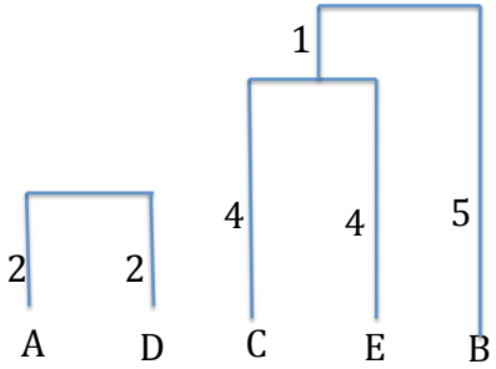
First set C1={A}, C2={B}, C3={C}, C4={D}, C5={E}.
The pair with the smallest distance is (C1,C4). Merge them to obtain C6={A,D} and create their parent node at distance d(C1,C4)/2 = 4/2 = 2 from each, to obtain:



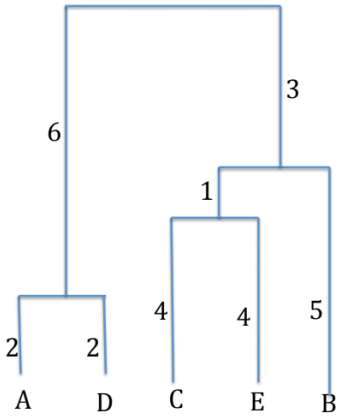
The next pair of clusters that is the closest is C3 and C5, with d(C3,C5)=8. Merge them to obtain C7={C,E} and create their parent node at distance d(C3,C5)/2 = 8/2 = 4 from each, to obtain:



The next pair of clusters that is the closest is C7={C,E} and C2={B}, with d(C7,C2)=(10+10)/2=10. Merge them to obtain C8={C,E,B} and create their parent node at height d(C7,C2)/2 = 10/2 = 5, to obtain:



There are only two clusters C6 and C8. Merge them and place their parent node at height d(C6,C8)/2 = ((16+16+16+16+16+16+16+16)/6) /2 = 8, to obtain:



Parsimony based

Large Parsimony Problem

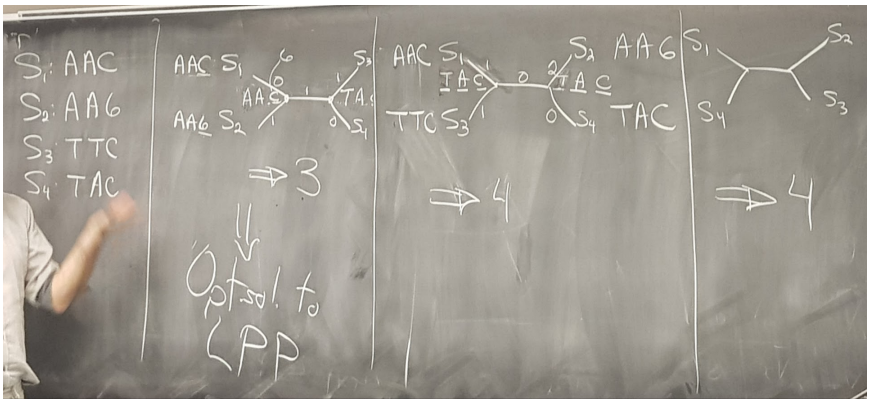
Given: n seqs S1, ..., Sn aligned to a MSA with columns with gaps removed

```
    x x
S1: AC-TA    CTA
S2: -CCTA -> CTA
S3: ACCAA    CAA
```

Find:

- Unrooted tree T, whose leaves are S1, ..., Sn
- Sequences Sn+1, Sn+2, ..., Sn+(n-2) of all internal nodes of T, s.t.

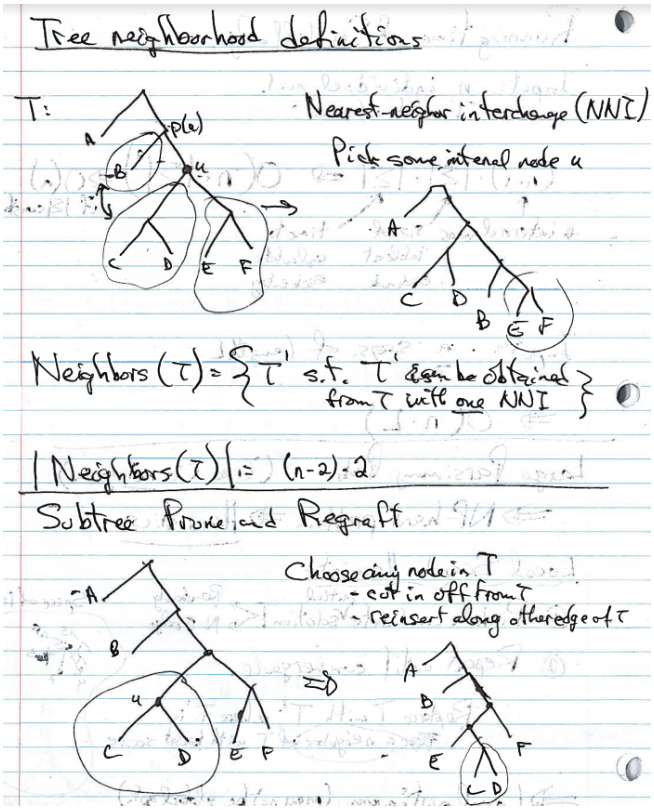
$$\sum_{(i,j)\in E(T)} d(S_i, S_j) \text{ is minimized}$$
$$d(S_i, S_j) = \#\text{ subsets b/w } S_i, S_j$$
$$E(T) \text{ edges of } T$$



Greedy search using nearest neighbor interchange (NNI):

1. Choose tree T0 randomly
2. Repeat...

$$T_{n+1} = argmin\{parsScore(T) : T \in NNI(T_n)\}$$

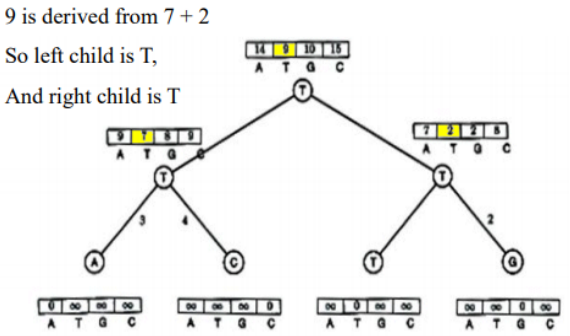


Small Parsimony Problem

Sankoff Algorithm:

Sankoff Algorithm: Dynamic Programming

- Calculate and keep track of a score for every possible label at each vertex
  - $s_t(v)$ = minimum parsimony score of the **subtree** rooted at vertex v if v has character t
- The score at each vertex is based on scores of its children:
  - $s_t(\textbf{parent}) = \min_i \{s_i(\textbf{ left child }) + \delta_{i,t}\} + \min_j \{s_j(\textbf{ right child }) + \delta_{j,t}\}$



Pseudocode:

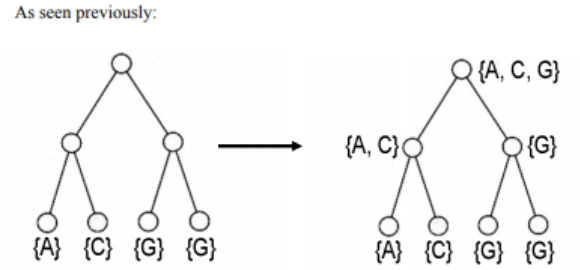
```
for each node u in tree (from leaves back to root):
    for each a in {A, C, G, T}:
        if u is leaf:
            Fu[a] = 0 if su == a else inf
        else:
            Fu[a] = min( Fv[a], min_{a != b}(Fv[b] + 1) )
                  + min( Fw[a], min_{a != b}(Fw[b] + 1) )
```

Fitch's Algorithm:

If u is a leaf then
$X_u = \{x\}$, where x is the nucleotide at leaf u.
Score(u) = 0

If u is an internal node with children v and w, then

If $(X_v \cap X_w = \emptyset)$ then
$X_u = X_v \cup X_w$
Score(u) = Score(v) + Score(w) + 1
Else
$X_u = X_v \cap X_w$
Score(u) = Score(v) + Score(w)



- Sankoff gives **same output** if have match=0 mismatch=1 scoring matrix

  Proof:

  For Sankoff, we have for each node *u*

  $$s_x(u) = \text{minimum score of subtree rooted at node } u \text{ if } u \text{ had nucleotide } x$$

  We see that nucleotide x is optimal for node *u* if

  $$s_x(u) = min_i\{s_i(u)\}$$

  Let

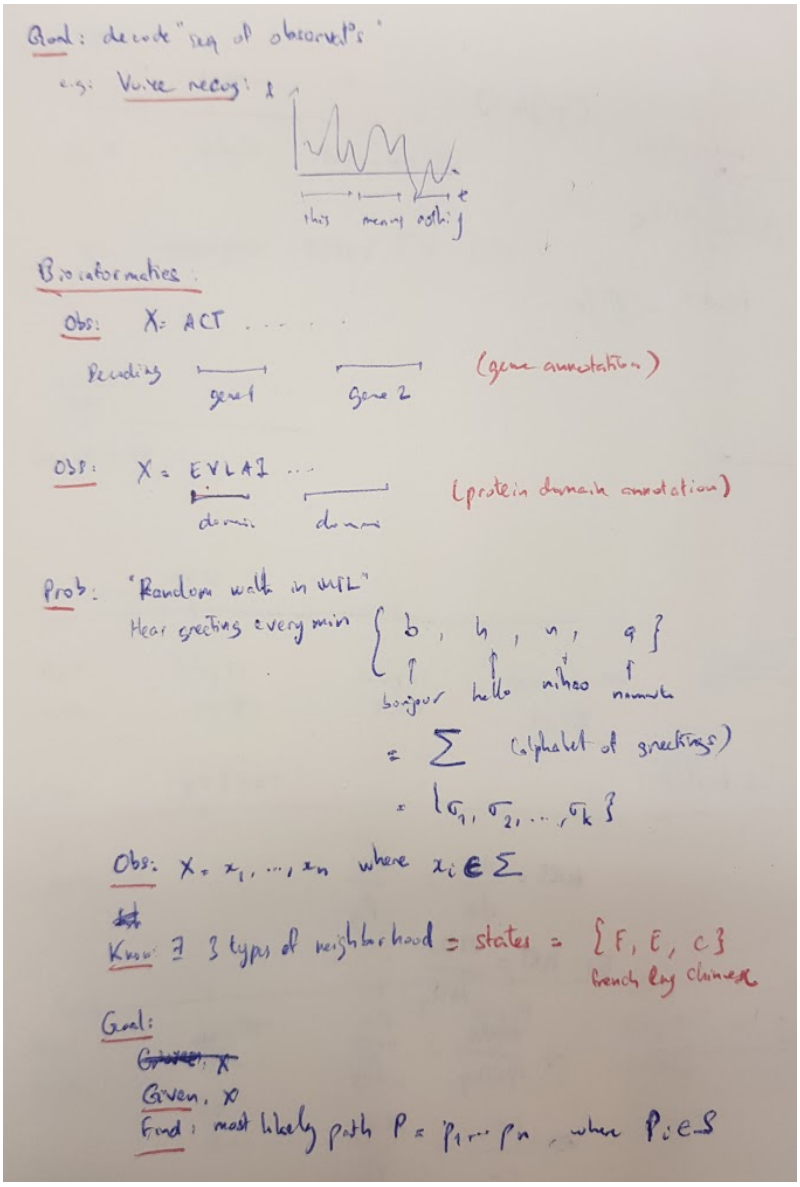
  $$S_u = \text{set of optimal nucleotides for node } u$$

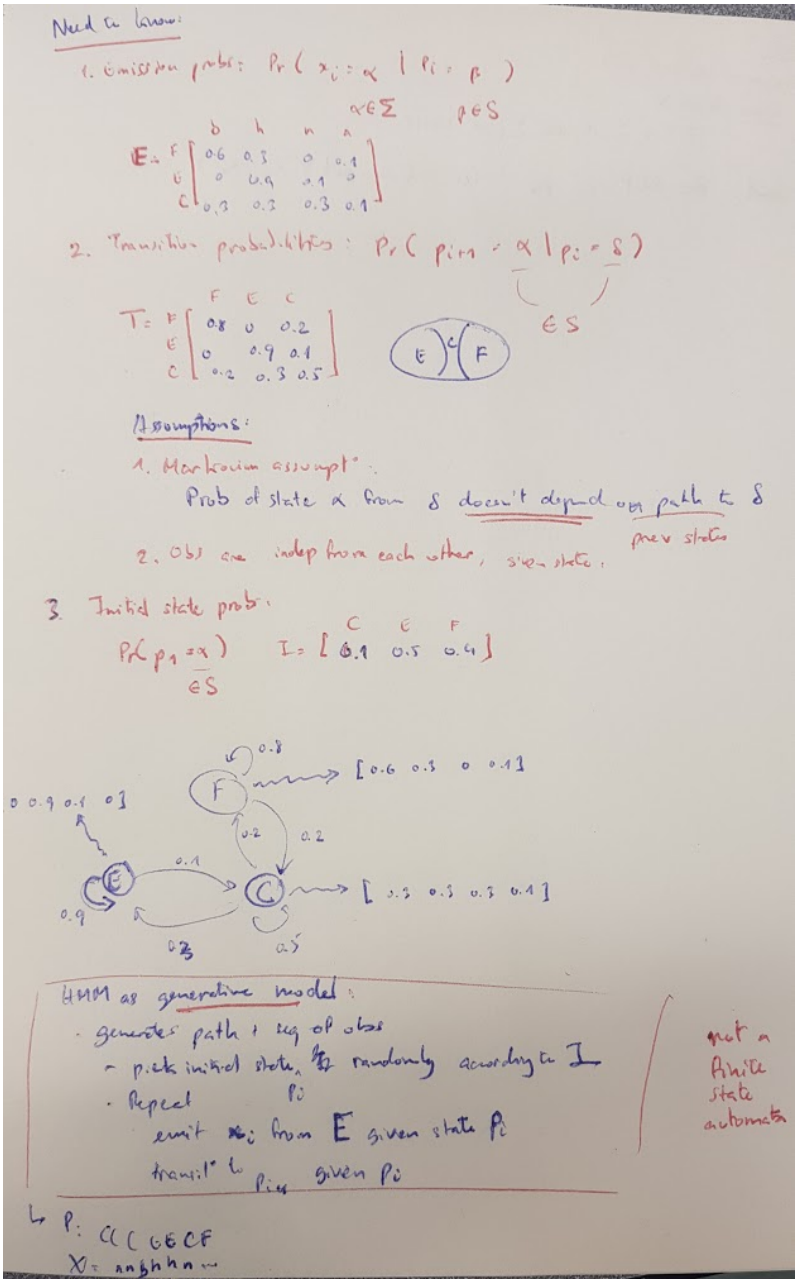
  Then:

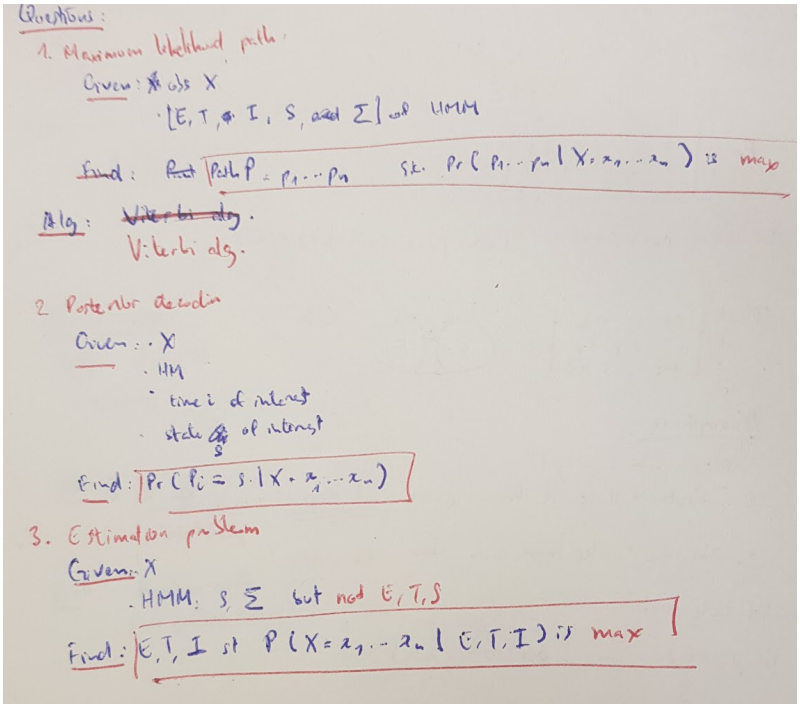
  $$\begin{aligned} S_u &= S_v \cup S_w \quad \text{if } S_v \cap S_w = \emptyset \\ &= S_v \cap S_w \quad \text{otherwise} \\ &v, w \text{ child nodes of } u \end{aligned}$$

  This is identical to Fitch's recurrence.

HHMs


Goal: decode "seq of observed's"
e.g: Voice recog:
this means nothing
Bioinformatics:
Obs: X = ACT ...
Reading
gene 1
gene 2
(gene annotation)
Obs: X = EVLAI ...
domain
domain
(protein domain annotation)
Prob: "Random walk in NFL"
Hear greetings every min
{ b , h , n , q }
bonjour hello nihao namaste
= Σ (alphabet of greetings)
= {σ1, σ2, ..., σk}
Obs: X = x1, ..., xn where xi ∈ Σ
Know: ∃ 3 types of neighborhood = states = {F, E, C}
French Eng chinese
Goal:
Given: X
Given, X
Find: most likely path P = p1 ... pn where pi ∈ S


Need to know:
1. Emission prob: $Pr(x_i = \alpha \mid \pi_i = \beta)$
$\alpha \in \Sigma$ $\beta \in S$
E = F [0.6 0.3 0 0.1]
E [0 0.9 0.1 0]
C [0.3 0.3 0.3 0.1]
a t g c
2. Transition probabilities: $Pr(\pi_{i+1} = \alpha \mid \pi_i = \beta)$
$\in S$
T = F [0.8 0 0.2]
E [0 0.9 0.1]
C [0.2 0.3 0.5]
F E C
F E C
Assumptions:
1. Markovian assumpt°:
Prob of state α from β doesn't depend on path to β
prev state
2. Obs are indep from each other, given state.
3. Initial state prob:
$Pr(\pi_1 = \alpha)$ $\in S$
I = [0.1 0.5 0.4]
C E F
0.8
[0.6 0.3 0 0.1]
F
0.2
0.2
0 0.9 0.1 0]
E
0.1
C
[0.3 0.3 0.3 0.1]
0.9
0.3
0.5
HMM as generative model:
- generates path + seq of obs
- pick initial state π_1 randomly according to I
- Repeat
emit x_i from E given state π_i
transit° to π_{i+1} given π_i
not a finite state automata
P: C C C G E C F
X = a g h h h m



Maximum Likelihood Path

GIven: X=x1...xn

Find: P=p1...pn st P(P|X) is max

Recall:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Therefore:

$$P(P|X) = \frac{P(X|P)P(P)}{P(X)}$$

1. From the independence assumptions:

$$P(X|P) = \prod_{i=0}^{n} P(x_i|p_i) = E(p_i, x_i)$$

2. Using independence and transition probabilities:

$$P(P) = P(p_1) \prod P(p_{i+1}|p_i) = I(P_1)T(P_i, P_{i+1})$$

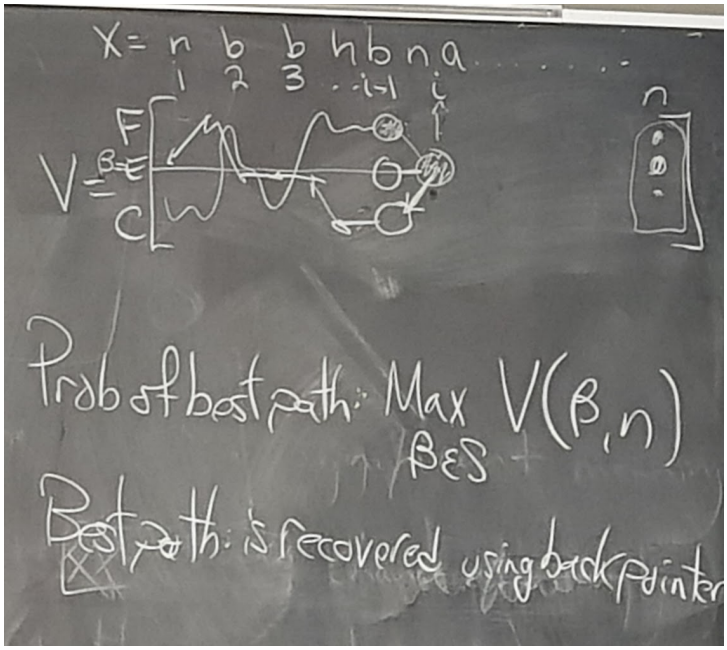
3. Denominator isn't needed: there's no P in it

Viterbi Algorithm

Define

$$V(\beta, i) = \text{prob of most likely path of len i, given } X = x_1...x_n, \text{ assuming } p_i = \beta$$
$$\beta \in S, i \in \{1...n\}$$

$$V(\beta, i) = max_{p_1...p_i, p_i=\beta}\{P(P = p_1...p_i, X = x_1...x_i)\}$$



Fill out V:

For i = 1:

$$V(\beta, 1) = I(\beta)E(x_i, \beta) \quad \forall \beta \in S$$

For i > 1:

$$V(\beta, i) = E(\beta, x_i) max_{\text{prev state } \delta}\{V(\delta, i-1)T(\delta, \beta)\}$$

Runtime:

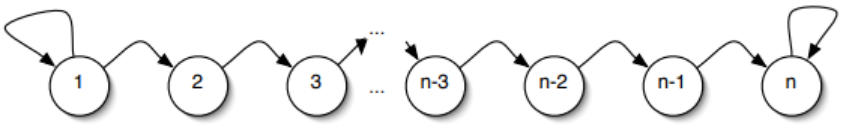
$$O(n|S|^2) \text{ time}$$
$$n|S| \text{ cells}, |S| \text{ prev cells to find max for each cell}$$

4) Hidden Markov Models (20 points)

a) (5 points) Consider a general hidden Markov model with n states. What is the worst-case running time of the Viterbi algorithm on a sequence of length L ?

$O(L\ n^2)$

b) (15 points) Now, consider the following hidden Markov model with n states. Is it possible to modify the Viterbi algorithm so that, on this particular type of linear HMM, the running time is asymptotically faster than in (a) ? Describe the modification required, and give the running time of the improved algorithm.



Use the following recurrence for the Viertbi algorithm:

*When k=1, V(k, i) = V(k,i-1) * T(1,1) * E(k, xi)*
*When k= 2... n-1, V(k, i) = V(k-1,i-1) * T(k-1,k) * E(k, xi)*
*When k= n, V(k, i) = max { V(k-1,i-1) * T(k-1,k) * E(k, xi), V(k, i-1) * T(k,k) * E(k, xi) }*

⇒ O(Ln) time

Assignments

Assignment 1

Multi-gap-free alignment:

$$M(i,j) = s(T_i, S_j) + max \begin{cases} M(i-1,j-1) \\ S_{gap}(i-1,j-1) \\ T_{gap}(i-1,j-1) \end{cases}$$
$$S_{gap}(i,j) = b + max \begin{cases} M(i-1,j) \\ T_{gap}(i-1,j) \end{cases}$$
$$T_{gap}(i,j) = b + max \begin{cases} M(i,j-1) \\ S_{gap}(i,j-1) \end{cases}$$

S, T sequences
$S_{gap}(i,j)$ score of best alignment of $S[1..j]$ and $T[1..i]$ ending in a gap in S
$T_{gap}(i,j)$ score of best alignment of $S[1..j]$ and $T[1..i]$ ending in a gap in T
s substituion cost matrix (instead of M)
b gap penalty cost

Runtime:

$$3(m+1)(n+1) = O(3mn) = O(mn)$$

LCS:

The Smith-Waterman scoring scheme is

$$F(i,j) = \max \begin{cases} 0, \\ F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases}$$

From Durbin et al.

whereas LCS's is

$$F(i,j) = \begin{cases} F(i-1,j-1) + 1 & \text{if } x_i = y_i \\ max(F(i-1,j), F(i,j-1)) & \text{if } x_i \neq y_i \end{cases}$$

To obtain a scheme equivalent to the above, we set match score 1, mismatch score 0 and indel penalty cost 0.

Then, performing traceback will yield an alignment **containing** the longest common subsequence.

Finally, eliminating pairs with gaps and mismatches from the final alignment yields the longest common subsequence.

Assignment 2

Small parsimony with quantitative traits:

$$min \sum_{(u,v)} \sum_{i}^{k} |D_u(i) - D_v(i)| = \sum_{i}^{k} min \sum_{(u,v)} |D_u(i) - D_v(i)|$$

The above reduce the problem to *k* small parsimony problems, one for each trait *i*.

Therefore, we will solve them by filling *Du(i)*, for each *i*, for all internal node *u* of the tree *T*. The vectors *Du*'s will be filled according to a Fitch inspired algorithm designed for quantitative values.

The algorithm is as follows:

```
for each trait i:
    for each leaf u:
        Du(i) = { Du(i) } # Initialize leaf values into sets

    choose_optimal_sets(i)
    choose_optimal_values(i)

def choose_optimal_sets(i)
    for each interal node u in in-order traversal (from leaves to root):
        let v, w = u.children

        if Dv(i) and Dw(i) overlap:
            Du(i) = intersection( Dv(i), Dw(i) )
        else:
            Du(i) = set of integers between Dv(i) and Dw(i)
                  # E.g. {1, 2} and {5, 10} -> {2, 3, 4, 5}

def choose_optimal_values(i)
   for each node u in breadth-first search (from root to leaves):
      if u is root:
          Du(i) = arbitrary value in Du(i)

      else: # u has parent
          let p = u.parent

          if Du(i) contains >1 values:
              Du(i) = value in Du(i) closest to Dp(i)
                    # E.g. Du(i) = {1, 2, 3} and Dp(i) = 5 -> Du(i) = 3
                    #              {1, 2, 3}           = 2          = 2

          else: # Du(i) contains only 1 value
              Du(i) = value in Du(i)
```