Sequence Assembly

But **we can't "read" off the sequence** of an entire molecule **all at once**. But we *do* **have the ability to read** or detect ***short* pieces** (substrings) of DNA

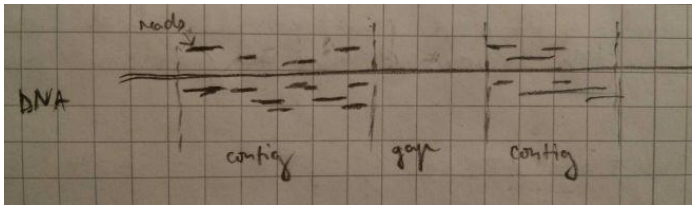
Key properties: **length in bp/read, error rate in %, price in $ per 1 million of base pairs**

* Sanger sequencing: 500-800 bp, 1%, **$2400**
* Next generation technologies: – 454 Genome Sequencer: 250-600 bp, 1%, **$10**
  – Illumina Genome Analyzer: 35-150 bp, 1%, **$0.15**
* Oxford Nanopore: x10 kbp/read, up to 30% error rate, the most portable sequencer

Shotgun sequencing

Statistics for shotgun sequencing

Given: G – genome length (3.10^9 nts), L – read length (500 nts), N – number of reads (tbd)
Calculate: coverage – $a=NL/G$



– **How many contigs** are there? **How big** are they?

– **How many reads** are in each contig?

– **How big are the gaps**?

Requirement: 99% in contigs, 1% in gaps gives – a=4.6, N=$3x10^7$, mean contig length 10^4, 100 reads/contig avg

The fragment assembly problem

Given: A set of reads (strings) $\{s_1, s_2, \ldots, s_n\}$

Do: Determine a large string *s* that "best explains" the reads

What do we mean by **"best explains"?**

Find a string *s* such that – **all reads** $s_1, s_2, \ldots, s_n$ are **substrings** of s
– ***s*** is **as short as possible**

What **assumptions** might we require?

– Reads are **100% accurate**

– Identical reads must come from the same location on the genome

– "**best" = "simplest"**

Example: Given the reads **{ACG, CGA, CGC, CGT, GAC, GCG, GTA, TCG}**. What is the shortest superstring you can come up with? **TCGACGCGTA** (length 10)

Algorithms for shortest superstring

Simple *greedy* strategy:

```
while # strings > 1 do
        merge two strings with maximum overlap
```

Other approaches are based on *graph theory…*

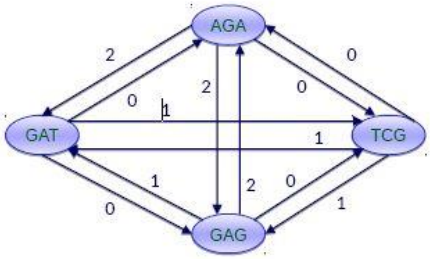
Overlap graph

For a **set of sequence reads** *S*, construct a **complete directed weighted graph** *G = (V,E,w)*

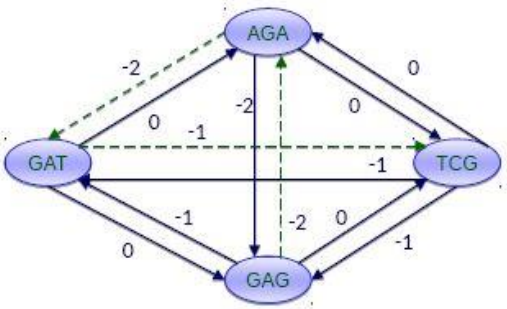
– with **one vertex per read** (v_i corresponds *to* s_i)

– $w(v_i,v_j)$ = *overlap*(s_i ,s_j) = **length of longest suffix of s_i that is a prefix of s_j**

Overlap graph example: **Let *S* = {AGA, GAT, TCG, GAG}**



Assembly as **finding a Hamiltonian path** (path through graph that visits each vertex exactly once)
Minimize superstring length = minimize weight of Hamiltonian path in overlap graph with edge weights negated (we go for **minimal weights** aka **max overlaps**)



path: **GAGATCG**
path weight: -5
string length: 7

finding Hamiltonian path is an **NP-complete problem**, but **nevertheless** overlap graphs **are often used** for sequence assembly

– **can detect repeats** (?)

– heuristical hierarchical decomposition • unitigs (no forks, no conflicts) solved first

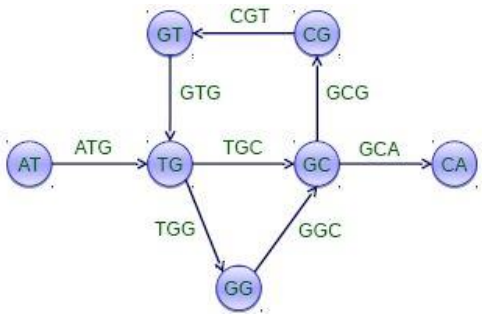
– mate-pairs to scaffold (?)

de Bruijn graph

– **edges** represent ***k*-mers** = subsequence of DNA **of lenght *k***

– **vertices** correspond to **(*k*-1)-mers**

{ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}

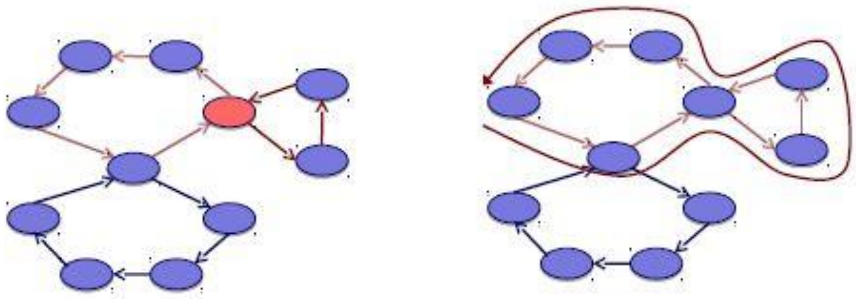


Can we find a **DNA sequence containing all *k*-mers**?

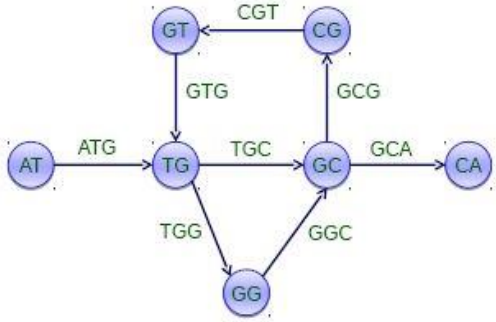
• In a de Bruijn graph, **can we find a path that visits every edge of the graph exactly once**?

Eulerian cycle algorithm

1) **start at any vertex** *v*, **traverse** unused **edges until returning** to *v*
2) while the **cycle is not Eulerian**
   – pick a **vertex *w* along the cycle** for which **there are untraversed outgoing edges**
   – **traverse unused edges until ending up back at *w***
   – **join two cycles** into one cycle



Assembly as finding Eulerian *paths* in de Bruijns graph, where resulting sequences **contain all *k*-mers**



assembly: ATGGCGTGCA or ATGCGTGGCA

Violating assumptions in de Bruijn graphs

Assume a sequence**: a_long_long_long_time**
length **m=21**, the **sequence contains repeats**
Choose ***k=5***, number of 5-mers is *m-k+1=17*
Assume different sets of k-mers:
- *ad a)* all 5-mers ➞ detected correct assembly,
- *ad b)* **omitting ong_t** ➞ two connected components, the overall graph is not Eulerian,
- *ad c)* **extra copy** of **ong_t** ➞ 4 semi-balanced nodes, graph not Eulerian,
- *ad d)* **errors and differences** between chromosomes, turn a copy of **long_** into **lxng_** ➞ graph not connected, largest component not Eulerian.

See. Picture

Short k-mers in de Bruijn graphs

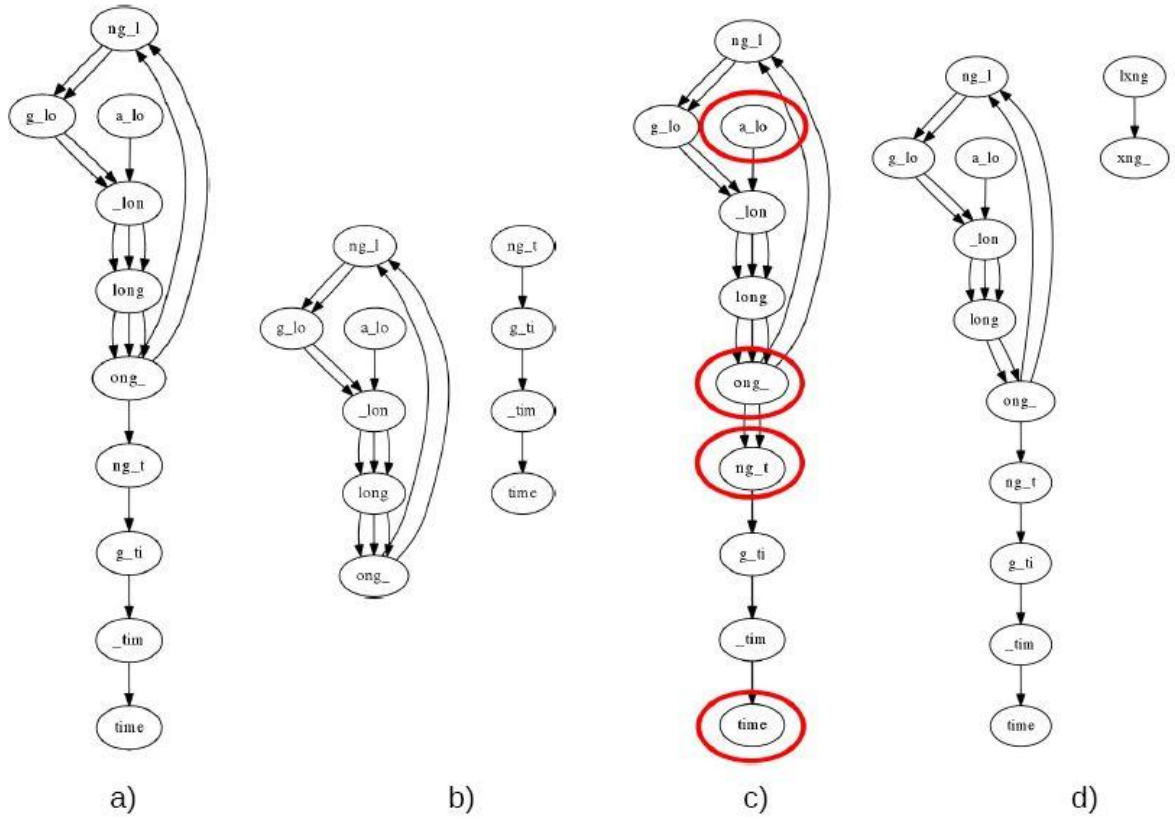
Only **short k-mers guarantee that none is missed** (must not be shorter, that the shortest read)

Still, **lenghts of k-mers don't matter in terms of complexity**, which remains ***O(N)*** where *N* = the total length of reads

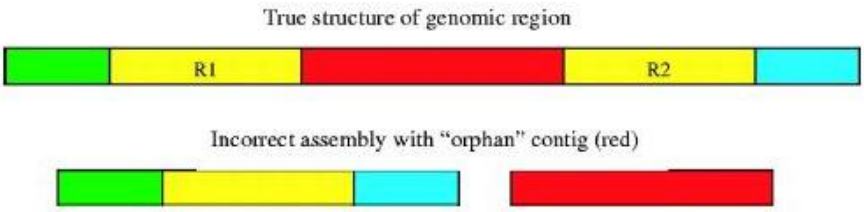
De Bruijn graph with *N* edges and *N* nodes too can be **constructed in *O(N)***, **Euler cycle** found **in *O(N).***

Repetitive sequences

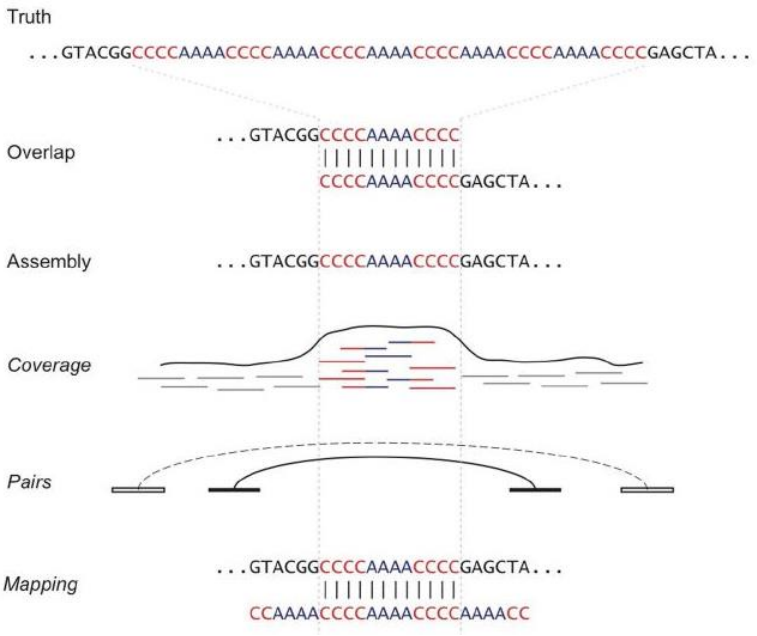
- Most common **source of assembly errors**
- If sequencing technology produces **|read| > |repeat|** size, **impact is much smaller**
- Most straightforward solution: **mate pairs** with spacing > **largest known repeat**



In scenario below both **green-yellow** and **yellow-blue borders** are contained in some *k*-mer. but **the yellow-red** one is not (a *k*-mer ends exactly at the border)

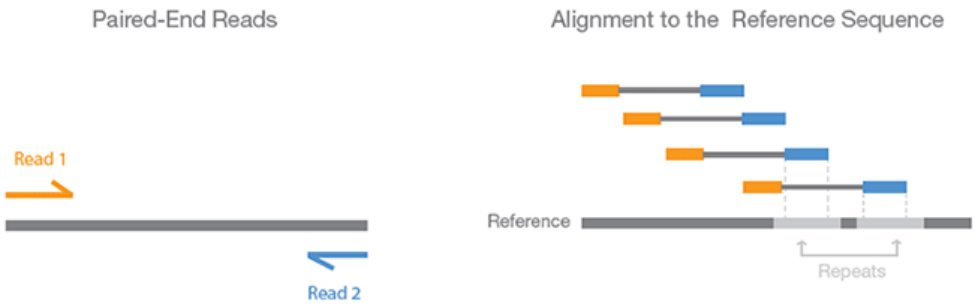


Mis-assembly of repetitive sequence



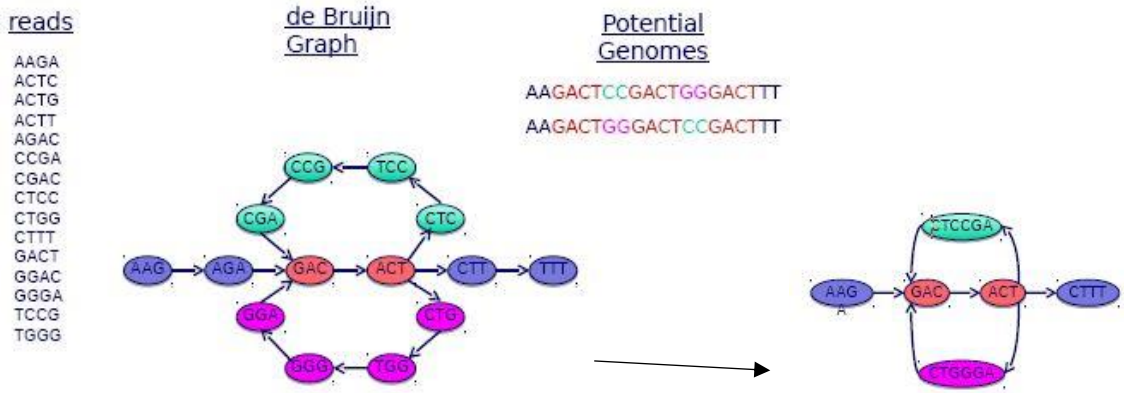
Paired and reads:

When we sequence a fragment from both ends and we know the fragment length, we can validate the assembly by checking, whether the distance between the two ends in the resulting genom is the same, as the distance between the ends in the fragment (the orange end serves as an anchor)

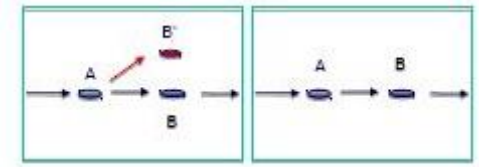


The Velvet assembler

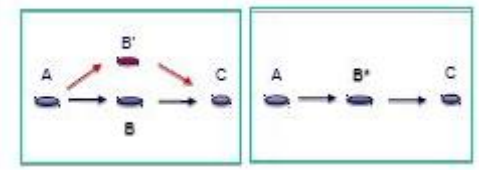
- built upon de Bruijn graphs
- includes **additional tricks** for
  - **reducing the size** of the **graph**
  - trying to **correct for errors in sequences**
  - taking advantage of paired-end reads



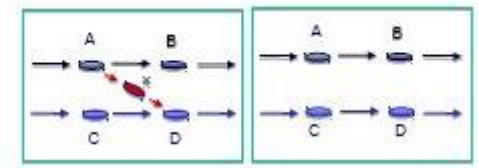
Velvet error correction



errors at the end of read → trim off „dead-end“ tips



errors in middle of read (pop-up bubbles)



chimeric edges → clip short, low coverage nodes

Pairwise Sequence Alignment

Given
– a pair of sequences (DNA or protein)
– a method for **scoring a candidate alignment**

Do
– determine the **correspondences between substrings** in the sequences such that the similarity score is maximized

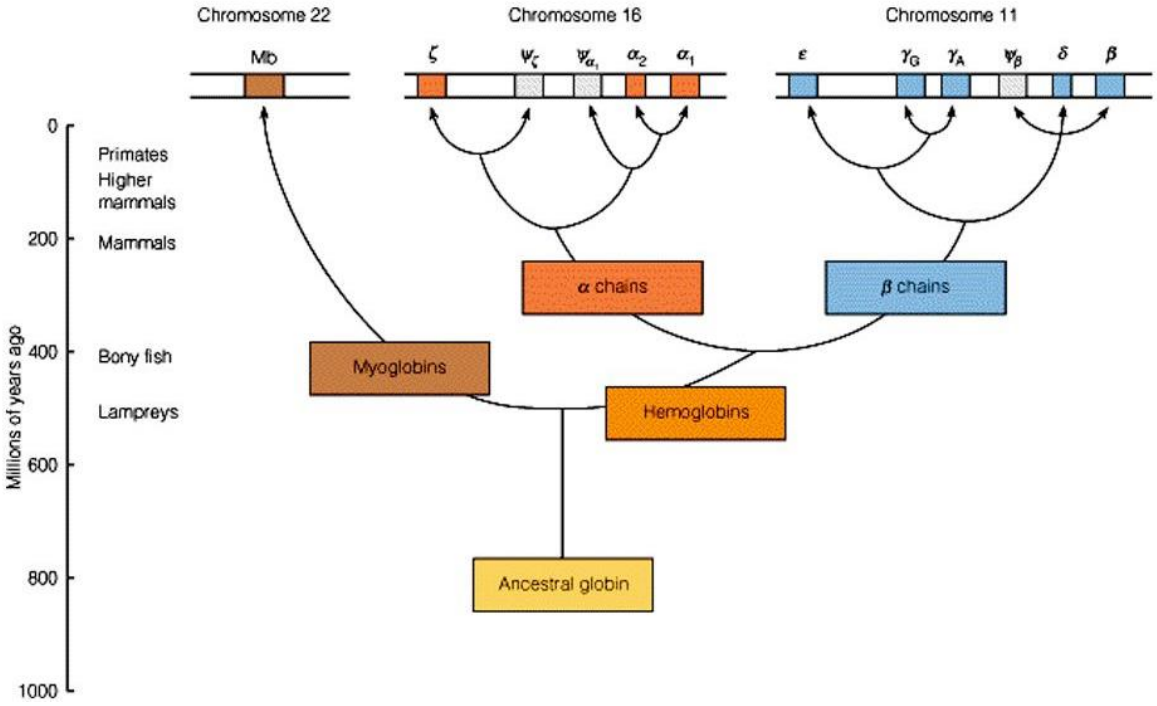
```
OprD    MKVMKWSAIALAVSAGSTQFAVADAFVS---DQAEAKGFIEDSSLDLLLR   47
PhaK    MSGK-------TTTMNRTHFMSAACLATLALPVPAMADFIGDSHARLELR   43
        *. .       ... ..*.*  *. ...     ..  .**.**   * **

OprD    NYYFNRDGKSGSGDRV---DWTQGFLTTYESGFTQGTVGFGVDAFGYLGL   94
PhaK    NHYINRDFRQSNAPQAKAEEWGQGFTAKLESGFTEGPVGFGVDAMGQLGI   93
        *.*.*** . ... ..   .*.*** .. *****.*.*******.* **.

OprD    KLDGTSDKTGTGNLPVMNDGK-PRDDYSRAGGAVKVRISKTMLKWGEMQP  143
PhaK    KLDSSRDRRNTGLLPFGPNSHEPVDDYSELGLTGKIRVSKSTLRLGTLQP  143
        ***...*. .** **   ... * ****  * . *.*.**. *. *..**
```

(Protein alignment)

The role of homology in alignment

• *homology*: **similarity** due to **descent from a common ancestor**

• often we can **infer homology from similarity** -> thus we can sometimes infer structure/function from sequence similarity



(Homology example, evolution of hemoglobin)

homologous sequence groups:

– *orthologous sequences*: sequences that differ because they are found in **different species** (e.g. human !-globin and mouse !-globin)
– *paralogous sequences*: sequences that differ because of a gene **duplication event** (e.g. human α-globin and human β-globin, various versions of both)

Important implications: while ***orthologs*** often fulfill **the same role**, ***paralogs*** tend to **diverge in their function**, so paralogy is a worse indicator of functional analogy than orthology.

DNA seqence edits

substitutions: ACGA ➡ AGGA

insertions: ACGA ➡ ACCGGAGA

deletions: ACGGAGA ➡ AGA

transpositions: ACGGAGA ➡ AAGCGGA

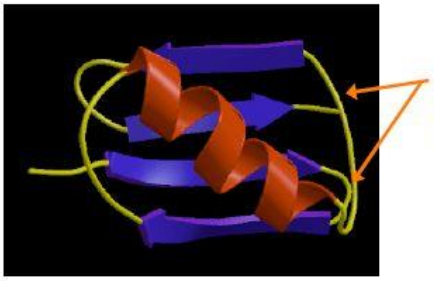
inversions: ACGGAGA ➡ ACTCCGA

Inversion:

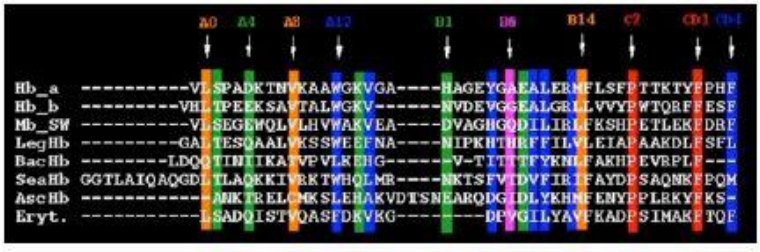
5′ ACGGAGA 3′
3′ TGCCTCT 5′
We have a DNA double helix, where the two opposing/complementary fragments get switched
Order of letters switched as well (5′ -> 3′)

- for **short DNA sequences** (*gene* scale) we will generally only consider
  - substitutions
  - insertions/deletions
- for **longer DNA sequences** (*genome* scale) we will consider additional events
  - transpositions
  - inversions

Why is it that **two "similar" proteins** may have **large insertions/deletions**?
– some insertions and deletions may not significantly affect the structure of a protein



- figure at right shows prototypical structure of globins
- figure below shows part of alignment for 8 globins



Issues in sequence alignment

- the sequences we're comparing typically **differ in length**
- there may be only a relatively s**mall region in the sequences that matches**
- we want to **allow partial matches** (i.e. some **amino acid pairs are more substitutable than others**)

• variable length regions may have been inserted/deleted from the common ancestral sequence

Types of alignment

• *global*: find best match of both sequences in their entirety

• *local*: find best subsequence match

• *semi-global*: find best match **without penalizing gaps on the ends of the alignment**

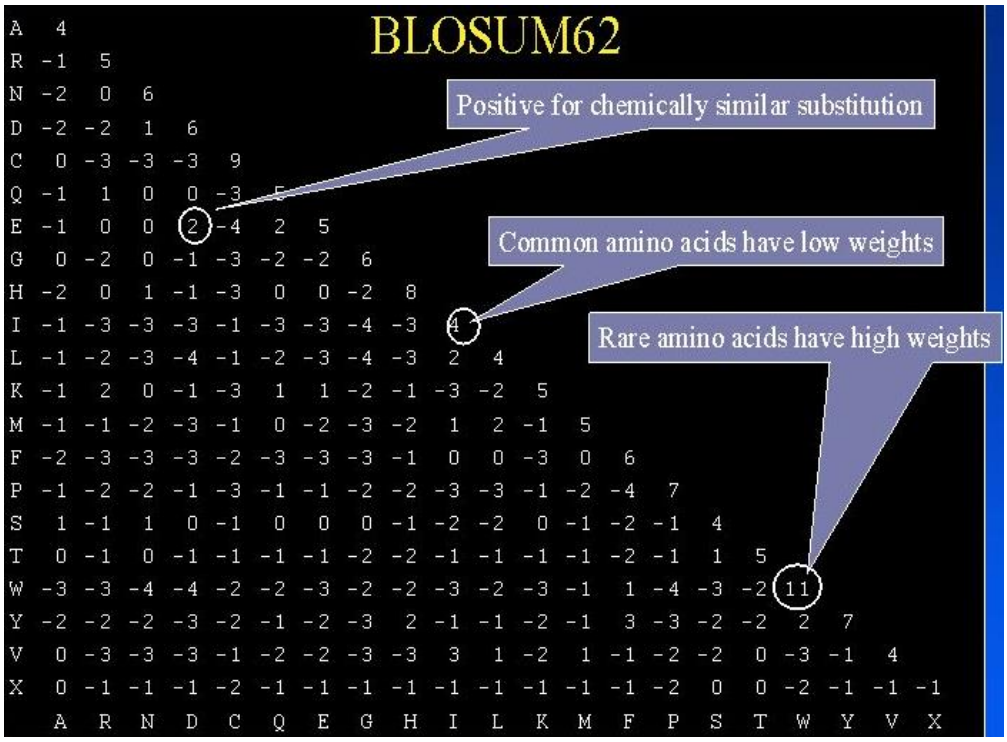
Scoring an alignment: what is needed?

• **substitution matrix**

– *s(a,b)* indicates score of aligning character *a* with character *b*

• **gap penalty function**

– *w(g)* indicates cost of a gap of length *g*



(substitution matrix for amino acides)

How do we construct subst. matrix? Where did the weights come from?

At the beginning, we start with an **empty matrix**, therefore we do the **alignment manually**. For that purposes, we start with a pool of **very similar proteins**. From those alignments, we **derive the initial** form of the matrix, which we then use for automated alignment. We update our matrix repeat the process until the weights don't change = **bootstraping**

The **weights** themselves are calculated as follows: $log\left(\frac{q_{i,j}}{p_i p_j}\right)$ where $p_i p_j$ are probabilities of AA *i,j* and $q_{i,j}$ is the probability of occurrence of their alignment
logarithm is to avoid small numbers coming from prob.

A W $q_{AA} = 1/6$
A W $q_{AW} = 4/6$
W A

BLOSUM62 = matrix where the **number** denotes how **close** were the **relations between proteins** in the pool

Linear gap penalty function

• the simplest case is when a **linear gap function** is used

$w(g) = -g \cdot d$ where d is a constant

• we'll start by considering this case, although **linear function is biologically wrong**. The following sequences have the **same score**: `A-A-A` vs. `A--AA` but the second one is **biologically more likely to happen**

EX: `VAHV---D--DMPNALSALSDLHAHKL`
`AIQLQVTGVVVTDATLKNLGSVHVSKG`

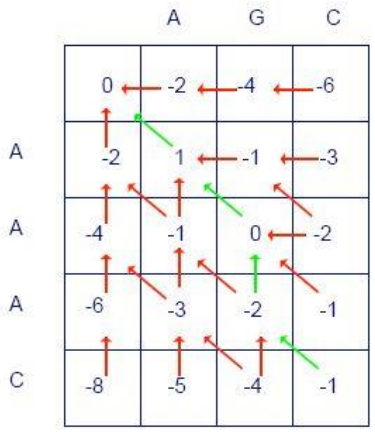
Score =
s(**V**,**A**) + s(**A**,**I**) + s(**H**,**Q**) + s(**V**,**L**) – 3*d* + s(**D**,**G**) – 2d

Pairwise alignment via dynamic programming

= determine **best alignment** of two sequences by determining **best alignment of all prefixes of the sequences**

- one way to specify the DP is in terms of its recurrence relation:

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



Rules:
s(i,j) = **+1** if $x_i = x_j$
= **-1** if $x_i \neq x_j$
d (penalty for aligning with a gap) = **-2**

one optimal alignment

x:	A	A	A	C
y:	A	G	-	C

Heuristic methods for sequence database searching

Heuristic alignment motivation

- *O(mn)* too slow for large databases with high query traffic
- heuristic methods **do fast approximation to dynamic programming**
  - –**FASTA** [Pearson & Lipman, 1988]
  - –**BLAST** [Altschul *et al*., 1990;
- consider the task of searching UniProtKB/Swiss-Prot against a query sequence:
  - – say our **query sequence is 362 amino-acids** long
  - – most recent release of **DB** contains **188,719,038** amino acids
  - – finding local **alignments** via **dynamic programming** would entail ***O*(10^{11})** matrix operations
- many servers **handle thousands of such queries a day** (NCBI > 500,000)

$$sesitivity = \frac{\#\ of\ significant\ matches\ detected}{\#\ of\ significant\ matches\ in\ DB}$$

Overview of BLAST (Basic Alignment Search Tool)

- given: query **sequence *q***, word length *w*, word score threshold *T*, segment score threshold *S*

• compile a **list of "words"** (of length ***w***) that **score at least *T*** when **compared to words from *q***

–**scan database** for **matches to words in list**

–**extend all matches** to seek high-scoring alignments

• return: alignments scoring at least *S*

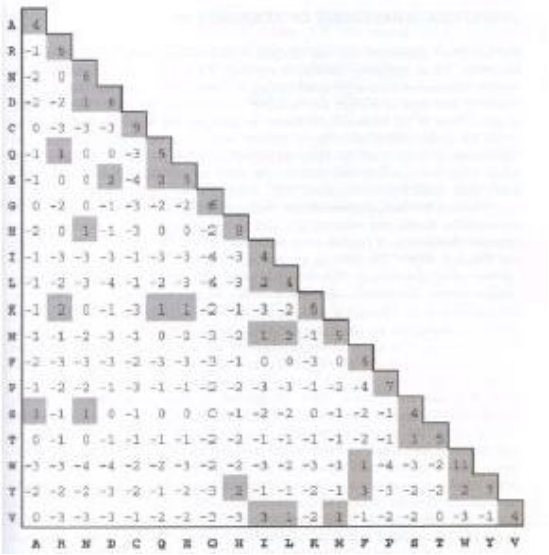
Determining query words

Given: query sequence: **QLNFSAGW**
word **length *w* = 2** (**default for protein usually *w* = 3**)
word score threshold *T* = 9

Step 1: determine **all words of length *w* in query sequence**
QL LN NF FS SA AG GW

Step 2: determine **all words** that **score at least *T*** when **compared to a word in the query sequence**

words from sequence	query words w/ T≥9
QL	QL=9
LN	LN=10
NF	NF=12, NY=9
…	
SA	none
…	

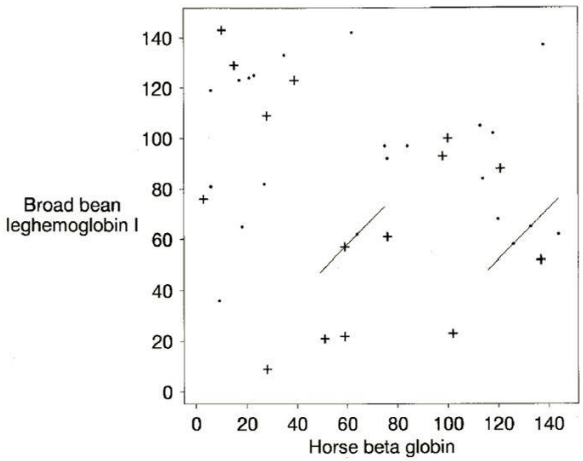


Step 3: **search database for all occurrences of query words**

–**index database sequences** into table of words (**pre-compute** this)

–**index query words into table** (at query time)

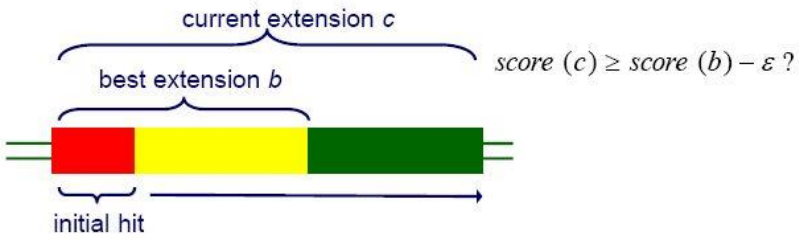
Step 4: **extend hits in both directions** (**without allowing gaps**) into local alignments



Each **cross** = **strong** hit, **dot** = **weaker** hit (in terms of score)

<u>Two hit method</u>
I want to **extend** those **hits**, which will **connect me to the other hits**. Therefore, extending weak hits may be in some cases better

- **terminate extension** in **one direction** when **score *falls certain distance below*** **best score for shorter extensions**



- return **segment pairs scoring at least *S* (best extension doesn't have to necesserily score above *S*)**

BLAST comments

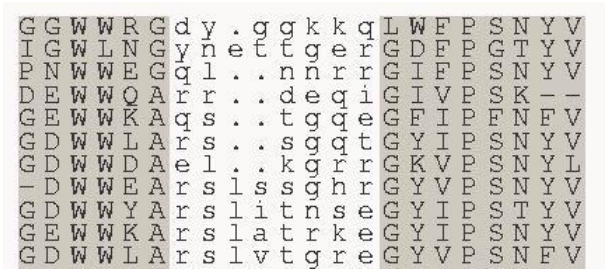
- it's **heuristic: may miss some good matches**
- it's **fast**: empirically, 10 to 50 times faster than Smith-Waterman
- the main **parameter** controlling the **sensitivity vs. running-time** trade-off is *T* (**threshold** for **what becomes a query word**) –small *T*: greater sensitivity, more hits to expand and vice versa
- large impact: –NCBI's BLAST server handles more than 500,000 queries a day
  –**most used bioinformatics program** in the world

Multiple sequence alignment

Task Definition
- **Given** – a **set** of **more than 2 sequences**
  – a **method for scoring an alignment**
- **Do:** – determine the **correspondences between the sequencessuch** that the **alignment score is maximized**

Motivation for MSA

- **establish input data** for **phylogenetic analyses**
- determine **evolutionary history of a set of sequences**
  – **at what point** in history did certain **mutations occur?**
- discovering a **common motif** in a set **of sequences** (e.g. **DNA** sequences that **bind the same protein)**
- **characterizing a set of sequences** (e.g. a protein family)
- **building *profiles* for sequence-database searching**
  – **PSI-BLAST** generalizes **a query sequence** into a **profile** to **search for remote relatives**



(Multiple Alignment of SH3 Domain)

Scoring a Multiple Alignment

- key issue: **how do we assess the quality of a multiple sequence alignment?**

$Score(m) = G + \sum_i S(m_i)$ where *G* is gap function and $S(m_i)$ score of the i-th column

- we'll discuss two methods – **sum of pairs (SP)**
  – **minimum entropy**

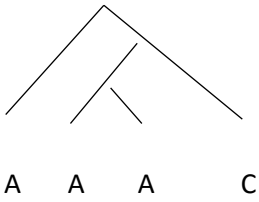
Scoring an Alignment: Sum of Pairs

- **compute the sum of the pairwise scores**

$$Score(m_i) = \sum_{k<l} s(m^k{}_i, m^l{}_i)$$

$m^k{}_i$ =character of the *k*th sequence in the *i* th column
s = substitution matrix

- this method is **biologically insufficient.** Consider this result:



column (A-A-A-C) with score -1 for mismatch

*s = 3*s(A, A) + 3*s(A, C) = 3 – 3 = 0*

Score is 0 even though the **sequences might be quite similar**

Scoring an Alignment: Minimum Entropy

- basic idea: try to **minimize the *entropy* of each column**
- another way of thinking about it**: columns that can be communicated using few bits are good**
- information theory tells us that an optimal code uses bits to encode a message $-log_2 p$ of probability *p*

  Ex: AAAA (0 bits), AAAC (<1 bit), AACC (1bit)

MSA: Dynamic Programming Approach

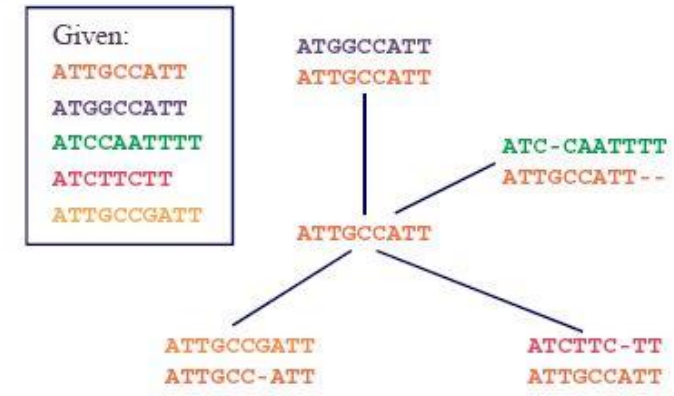
- We have **2 scoring methods** now *(sum of pairs* and *minimum enthropy*). Now we can utilize them to find **optimal alignments using dynamic programming**
- generalization of methods for pairwise alignment
  – consider ***k*-dimension matrix for *k* sequences** (instead of 2-dimensional matrix)
  – **each matrix element represents alignment score for *k* subsequences** (instead of 2 subsequences)
- given *k* sequences of length *n,* space complexity is $O(n^k)$
  -time complexity is $O(k^2 2^k n^k)$ – we have n^k, for which we examine 2^kneighbours if computed with sum of pairs, **is $O(k 2^k n^k)$** with minium enthropy

MSA: Heuristic Alignment Methods

- since **time complexity of DP** approach is **exponential** in the number of sequences, heuristic methods are usually used
- ***progressive alignment***: construct a **succession of pairwise alignments**
  – star approach
  – tree approaches, like CLUSTALW

Star Alignment Approach

- given: ***k* sequences** to be aligned $x_1, x_2, \dots, x_k$
  – pick **one sequence as the "center"** x_c
  – for each other sequneces x_i determine an **optimal alignment between x_i and x_c**
  – **merge pairwise alignments**
- return: multiple alignment resulting from **aggregate**
- approaches to **Picking the Center**
  1. try **each sequence as the center**, return **the best** multiple alignment
  2. compute **all pairwise alignments** and **select the string that maximizes**:
     $\sum_{i \neq c} sim(x_i, x_c)$ (ie. pick the most similar to the others)
- aggregating Pairwise Alignments **- "once a gap, always a gap"** rule (see example)



merging pairwise alignments

	present pair	alignment
1.	ATGGCCATT ATTGCCATT	ATTGCCATT ATGGCCATT
2.	ATC-CAATTTT ATTGCCATT--	ATTGCCATT-- ATGGCCATT-- ATC-CAATTTT
3.	ATCTTC-TT ATTGCCATT	ATTGCCATT-- ATGGCCATT-- ATC-CAATTTT ATCTTC-TT--
4.	ATTGCCGATT ATTGCC-ATT	ATTGCC-A TT-- ATGGCC-A TT-- ATC-CA-A TTTT ATCTTC- - TT-- ATTGCCG A TT--

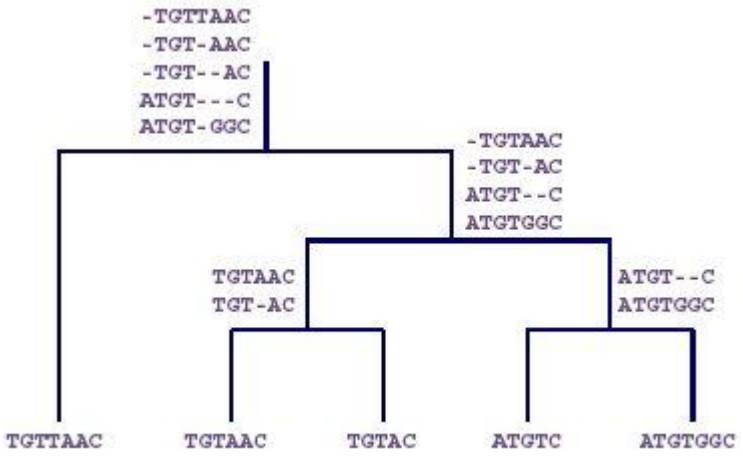
shift entire columns when incorporating a gap

Tree Alignments

- basic idea: organize multiple sequence alignment using **a *guide tree***
  - **leaves** represent **sequences**
  - internal **nodes** represent **alignments**
- determine alignments from bottom of tree upward
  - return **multiple alignment represented at the root of the tree**
- one common variant: the CLUSTALW algorithm [Thompson et al. 1994]

Doing the Progressive Alignment in CLUSTALW

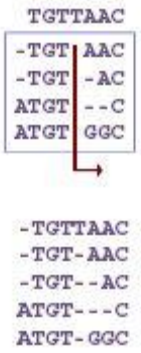
- depending on the internal node in the tree, we may have to align a
  - a sequence with a sequence
  - a sequence with a *profile* (partial alignment)
  - a *profile* with a *profile*
- in all cases we can use dynamic programming
  - for the profile cases, use SP scoring



(Tree alignment example)

• aligning sequences/profiles to profiles is essentially pairwise alignment

– shift entire columns when incorporating gaps



Building a tree:

1. do **pair alignment** for **all sequences**
2. pick the **best scored alignment** and **create a *profile***

The **profile is "locked",** which means we **do not shift individual sequences** in the profile, only **add gaps/shift the entire compound**

Introduction to Phylogenetic Trees

task definition
- Given – **data characterizing a set of species/genes**
- Do – **infer a *phylogenetic tree*** that accurately
  – characterizes the **evolutionary lineages** among the species/genes

Phylogenetic tree basics

• **leaves** represent things (**genes, species, individuals**/strains) being compared

• **internal nodes** are **hypothetical ancestral** units

• in a ***rooted*** **tree**, **path from root to a node represents an evolutionary path**

– the **root** represents **the common ancestor (see** *(Homology example, evolution of hemoglobin)*

• an ***unrooted*** **tree** specifies **relationships among things**, but **not evolutionary paths**

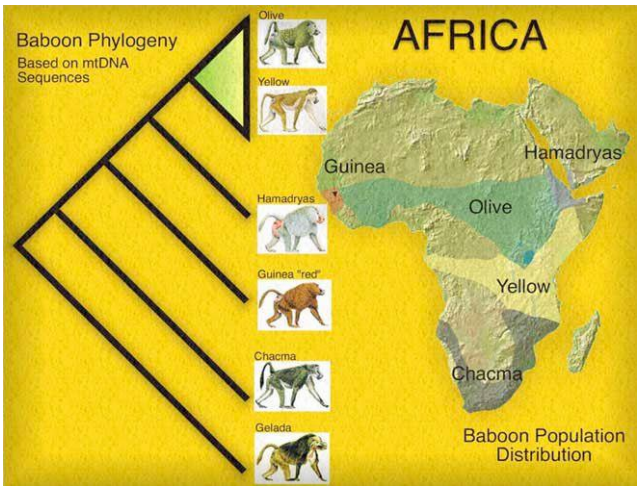
Why construct trees?

– to understand **lineage of various species**

– to understand **how various functions evolved**

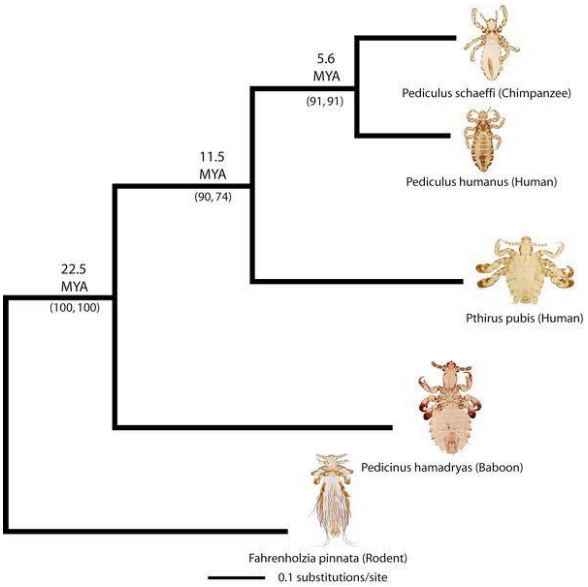
– to **inform multiple alignments**

– to identify what is **most conserved/important in some** class of **sequences**



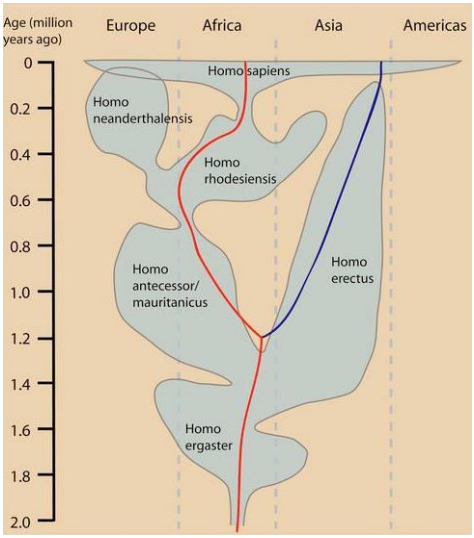
(Baboon phylogeny)

Genetic Analysis of Lice Supports Direct Contact between Modern and Archaic Humans



- inferred **phylogeny of lice species** closely **parallels accepted phylogeny of their hosts**
- can phylogeny of lice tell us something about evolution of hosts?

Phylogeny below supports a **theory of human evolution** in which:



– ***H. erectus*** and the **ancestors of *H. sapiens*** had **little or no contact** for a **long period of time**

– there was **contact between *H.erectus*** and ***H. sapiens*** as late as **30,000 years ago**

Data for building trees

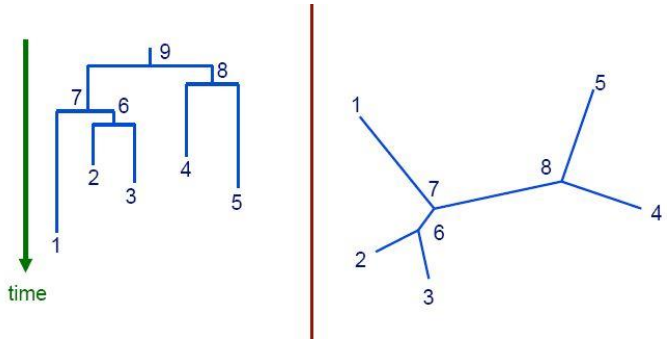
- trees can be **constructed from various types of data**

– ***distance-based***: measures of **distance between** species/**genes**

– ***character-based***: **morphological features** (e.g. #legs), DNA/protein sequences

– ***gene-order***: linear **order of orthologous genes** in given genomes

Rooted vs. unrooted trees



Number of possible trees

- given *n* sequences, there are

  $\prod_{i=3}^{n}(2i-5)$ possible **unrooted trees**

  $(2n-3)\prod_{i=3}^{n}(2i-5)$ possible **rooted trees**

# taxa (n)	# unrooted trees	# rooted trees
4	3	15
5	15	105
6	105	945
8	10,395	135,135
10	2,027,025	34,459,425

Phylogenetic tree approaches

- three general types of **methods**:

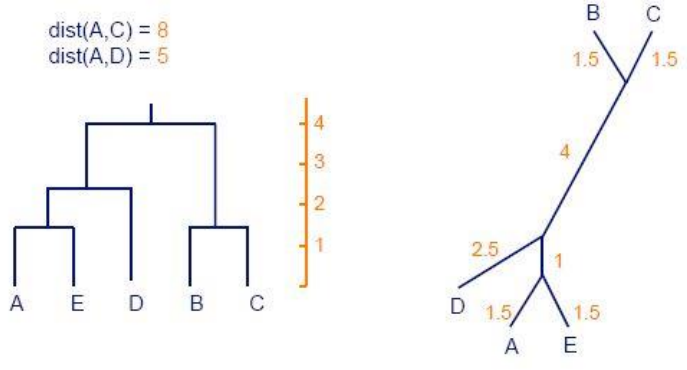
  – ***distance***: find **tree** that accounts for **estimated evolutionary distances**

  – ***parsimony***: find the **tree** that requires **minimum number of changes to explain the data**

  – ***maximum likelihood***: find the tree that **maximizes the likelihood of the data**

Distance-Based Approaches to Inferring Phylogenetic Trees

Distance representation in rooted and unrooted trees

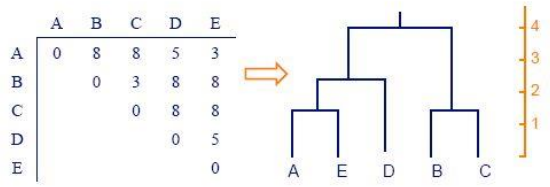


distances represented by summed height of edges to reach common ancestor

distances represented by summed length of edges to reach common ancestor

given: an $n \times n$ matrix M where M_{ij} is the distance between taxa *i* and *j!*

do: build an edge-weighted tree such that the distances between leaves *i* and *j* correspond to M_{ij}



Distances

• commonly obtained from sequence alignments. in alignment of sequence *i* with sequence *j:*

$$dist(i,j) = \frac{\#mismatches}{\#mismatches + \#matches}$$

• **properties of a distance metric**

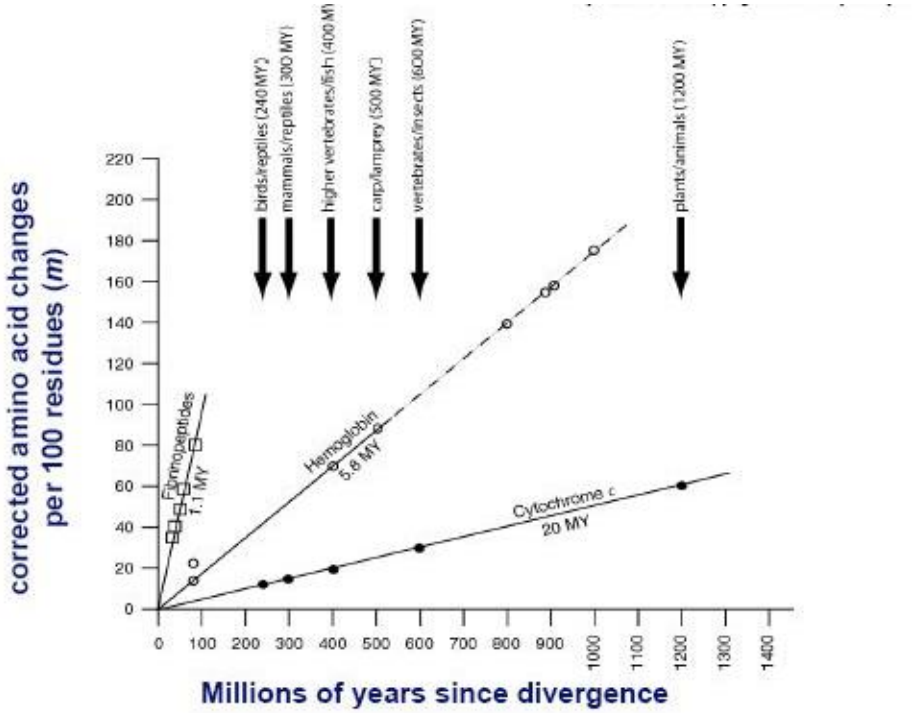
$$dist(x_i, x_j) \geq 0$$
$$dist(x_i, x_i) = 0$$
$$dist(x_i, x_j) = dist(x_j, x_i)$$
$$dist(x_i, x_j) \leq dist(x_i, x_k) + dist(x_k, x_j)$$

The molecular clock hypothesis

• In the 1960s, sequence data were accumulated for small, abundant proteins such as globins, cytochromes c, and fibrinopeptides. **Some proteins appeared to evolve slowly**, while **others evolved rapidly**.

• Linus Pauling, Emanuel Margoliash and others proposed the **hypothesis of a molecular clock**: ***For every given protein, the rate of molecular evolution is approximately constant in all evolutionary lineages***

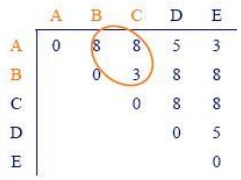
- E. Margoliash, *who wrote: "It appears that the number of residue differences between* cytochrome c *of any two species is mostly conditioned by the time elapsed since the lines of evolution leading to these two species originally diverged. If this is correct, the cytochrome c of all mammals should be equally different from the cytochrome c of all birds. Since fish diverges from the main stem of vertebrate evolution earlier than either birds or mammals, the cytochrome c of both mammals and birds should be equally different from the cytochrome c of fish. Similarly, all vertebrate cytochrome c should be equally different from the yeast protein.*



• the molecular clock **assumption is not generally true**: **selection pressures vary across time periods, organisms, genes within an organism, regions within a gene**

• if the **assumption does hold**, then the data is said to be ***ultrametric***

• **ultrametric data**: for **any triplet of sequences**, *i*, *j*, *k*, the **distances are either all equal**, or **two are equal and the remaining one is smaller**
• **M(i,j)** can be considered **the time since *i* and *j* diverged in evolution:**
"A and B** had **last common ancestor 8 million years ago"



	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0

The UPGMA method

• (Unweighted Pair Group Method using Arithmetic Averages)

• **given ultrametric data**, UPGMA will **reconstruct the tree *T*** that is **consistent with the data**

• basic idea: – **iteratively pick two taxa/clusters and merge them**
– **create new node in tree for merged cluster**

• **distance between clusters C_i and C_j** of taxa is defined as

$$d_{ij} = \frac{1}{|C_i||C_j|} \sum_{p \in C_i, q \in C_i} d_{pq}$$

(**avg. distance between pairs of taxa from each cluster**)

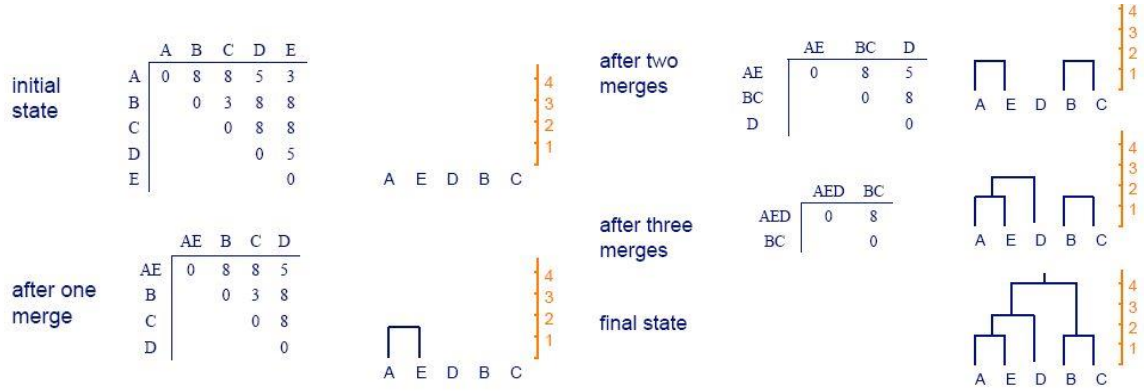
UPGMA algorithm

assign **each taxon to its own cluster**
define **one leaf for each taxon**; place it at **height 0**
while more than two clusters
find **two clusters *i*, *j* with smallest d_{ij}**
define a **new cluster $C_k = C_i \cup C_j$**
define a **node *k* with children *i* and *j*; place it at height $d_{ij}/2$**
replace clusters *i* and *j* with *k*
compute distance between *k* and other clusters
join last two clusters, *i* and *j*, by root at height $d_{ij}/2$

• given a new cluster C_k formed by merging C_i and C_j
• we can calculate the distance between C_k and any other cluster C_l as follows:

$$d_{kl} = \frac{d_{il}|C_i| + d_{jl}|C_j|}{|C_i| + |C_j|}$$

UPGMA example



Neighbor joining

• unlike UPGMA
– **doesn't make molecular clock assumption**
– produces **unrooted trees**

• **does assume *additivity***: **distance between pair of leaves is sum of lengths of edges connecting them**

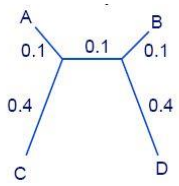
• like UPGMA, **constructs a tree by iteratively joining subtrees**

• two key differences
  – **how pair of subtrees to be merged is selected on each iteration**
  – **how distances are updated after each merge**

Picking pairs of nodes to join in NJ

• at **each step**, we **pick a pair of nodes to join**; **should we pick a pair with minimal d_{ij}?**

• suppose the real tree looks like this and we're picking the first pair of nodes to join?



$d_{AB} = 0.3$

$d_{AC} = 0.5$

wrong decision to join A and B: need to consider distance of pair to other leaves

• to avoid this, pick pair to join based on D_{ij}

$$D_{ij} = d_{ij} - (r_i + r_j)$$

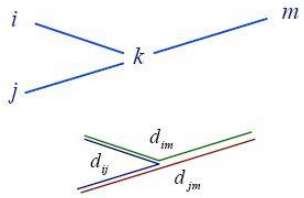
$$r_i = \frac{1}{|L| - 2} \sum_{k \in L} d_{ik}$$

where *L* is the set of leaves and r_i **is average distance of node *i* to all nodes except *i* and *j***

Updating distances in neighbor joining

• given a **new internal node *k*, the distance to another node *m*** is given by:

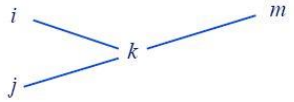
$$d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$$



k_m to each node *m* can be calculated only if *additivity* holds. Otherwise we have to take the average of all k_m

• can **calculate the distance from a leaf to its parent node** in the **same way:**

$$d_{ik} = \frac{1}{2}(d_{ij} + d_{im} - d_{jm})$$



$$d_{jk} = d_{ij} - d_{ik}$$

• we can **generalize** this so that **we take into account the distance to all other leaves**

$$d_{ik} = \frac{1}{2}(d_{ij} + r_i + r_j)$$

$$r_i = \frac{1}{|L| - 2} \sum_{k \in L} d_{ik}$$

- this is more robust if data aren't strictly additive

Neighbor joining algorithm

define the **tree *T* = set of leaf nodes**
L* = *T
while more than two subtrees in *T*
  pick the pair *i, j* in *L* with **minimal D_{ij}**
  add to *T* a new node *k* joining *i* and *j*

determine **new distances**

$$d_{ik} = \frac{1}{2}(d_{ij} + r_i + r_j)$$

$$d_{jk} = d_{ij} - d_{ik}$$

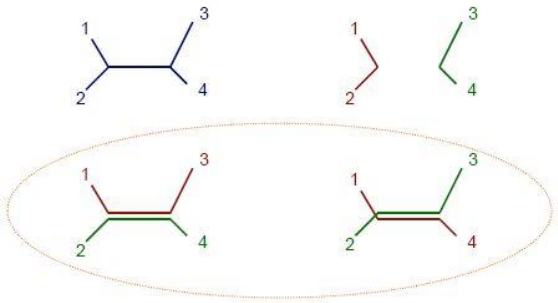
$$d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$$

remove *i* and *j* from *L* and **insert *k* (treat it like a leaf)**

join two remaining subtrees, *i* and *j* with edge of length

Testing for additivity

• for every set of four leaves, *i*, *j*, *k*, and *l*, two of the distances $d_{ij}+d_{kl}$, $d_{ik}+d_{jl}$ and $d_{il}+d_{jk}$ must be equal and not less than the third

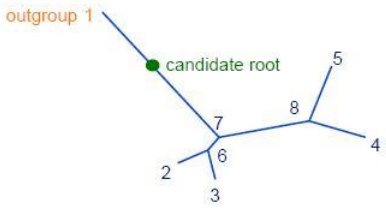


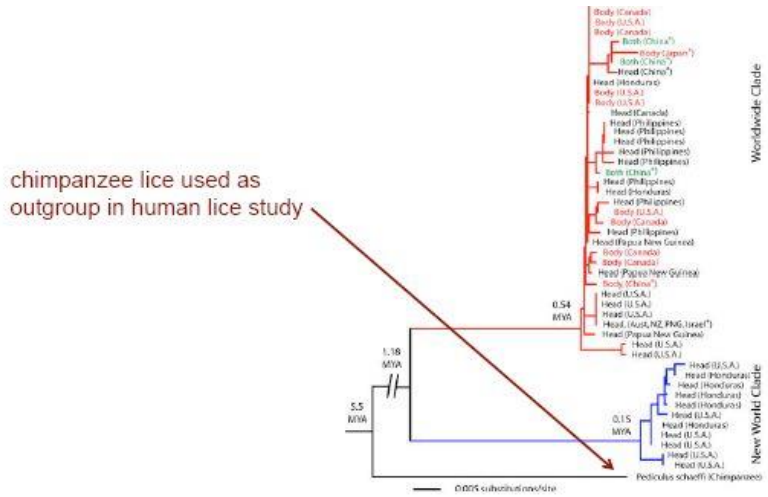
Rooting trees

• **finding a root in an unrooted tree** is sometimes accomplished by **using an *outgroup***

• ***outgroup***: a **species known to be more distantly related to remaining species** than **they are to each other**

• **edge joining the outgroup to the rest of the tree is best candidate for root position**





Comments on distance-based methods

• if the given **distance data is ultrametric** (and these distances represent real distances), then **UPGMA will identify the correct tree**

• if the **data is additive** (and these distances represent real distances), then **neighbor joining will identify the**

correct tree

• otherwise, the **methods may not recover the correct tree**, but they may **still be reasonable heuristics**

• **neighbor joining is commonly used**

Parsimony-Based Approaches to Inferring Phylogenetic Trees

Phylogenetic tree approaches have three general types

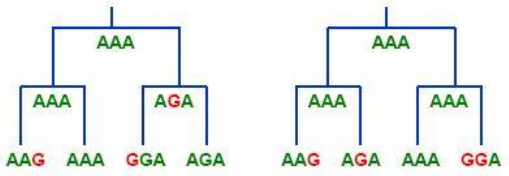
– ***distance***: find tree that accounts for estimated evolutionary distances
– ***parsimony***: find the **tree** that requires **minimum number of changes** to explain the data
– ***maximum*** *likelihood*: find the tree that maximizes the likelihood of the data

Parsimony based approaches

given: character-based data

do: find tree that explains the data with a minimal number of changes
• focus is on finding **the right tree topology**, not on **estimating branch lengths**

EX: there are various trees that could explain the phylogeny of the sequences **AAG, AAA, GGA, AGA** including these two:



• parsimony prefers the first tree because it **requires fewer substitution events**

• these approaches involve two **separate components**
1. a procedure to **find the minimum number of changes needed to explain the data** (for a given tree topology)
2. a **search through the space of trees**

Finding minimum number of changes for a given tree

• **basic assumptions**
– **any state** (e.g. nucleotide, amino acid) can **convert to any other state**
– the "**costs" of these changes are uniform**
– **positions are independent**; we can compute the min **number of changes** for **each position separately**

• **brute force approach**
– for **each possible assignment of states** to the **internal nodes**, **calculate the number of changes**
– report tne min number of changes found
• runtime is $O(Nk^N)$, where k = number of possible character states (4 for DNA)
N = number of leaves

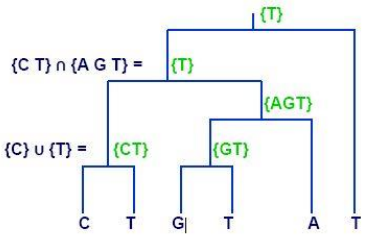
Fitch's Algorithm [1971]

1. traverse **tree from leaves to root** determining **set of possible *states*** (e.g. nucleotides) for **each internal node**
2. traverse **tree from root to leaves picking ancestral states** for internal nodes

Step 1 (possible states for internal nodes)

• do a post-order (**from leaves to root**) traversal of tree and **determine possible states R_i** of internal node i with children j and k

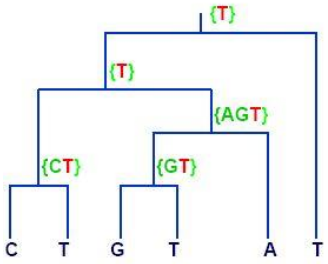
$$R_i = \begin{cases} R_j \cup R_k & \text{if } R_j \cap R_k = \emptyset \\ R_j \cap R_k & \text{otherwise} \end{cases}$$



Step 2 (select states for internal nodes)

• do a pre-order (from **root to leaves**) traversal of tree and **select state of internal node** *j* with parent *i*

$$r_j = \begin{cases} r_i & \text{if } r_i \in R_j \\ \text{arbitrary state } \in R_j & \text{otherwise} \end{cases}$$



Weighted parsimony

• **instead of** assuming **all state changes** are **equally likely**, use different costs *S*(*a*,*b*) for different changes

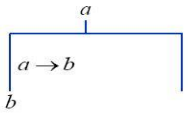
1st step of algorithm is to propagate costs up through tree

• want to determine **cost** $R_i(a)$ **of assigning character** a to node *i*

• for **leaves**: $R_i(a) = \begin{cases} 0 & \text{if character } a \text{ is at leaf} \\ \infty & \text{otherwise} \end{cases}$

• for an **internal node** *i* with children *j* and *k*:

$$R_i(a) = \min_b(R_j(b) + S(a)) + \min_b(R_k(b) + S(a))$$



Example: weighted parsimony

$R_3[A]=\infty, R_3[C]=\infty, R_3[G]=0, R_3[T]=\infty$

$R_4[A]=\infty, R_4[C]=\infty, R_4[G]=\infty, R_4[T]=0$

$R_2[A]=R_3[G]+S(A,G)+R_4[T]+S(A,T)$

$\vdots$

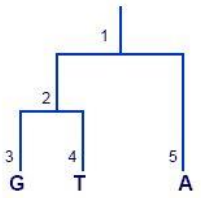
$R_2[T]=R_3[G]+S(T,G)+R_4[T]+S(T,T)$

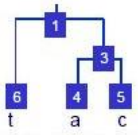
$R_5[A]=0, R_5[C]=\infty, R_5[G]=\infty, R_5[T]=\infty$

$R_1[A]=\min\left(R_2[A]+S(A,A),\ \ldots,\ R_2[T]+S(A,T)\right)+R_5[A]+S(A,A)$

$\vdots$

$R_1[T]=\min\left(R_2[A]+S(T,A),\ \ldots,\ R_2[T]+S(T,T)\right)+R_5[A]+S(T,A)$





$R_3(a)=0+0.8=0.8$

$R_3(c)=0.8+0=0.8$

$R_3(g)=0.2+0.7=0.9$

$R_3(t)=0.9+0.5=1.4$

	a	c	g	t
a	0	0.8	0.2	0.9
c	0.8	0	0.7	0.5
g	0.2	0.7	0	0.1
t	0.9	0.5	0.1	0

$R_1(a)=0.9+\min\{0.8,\ 0.8+0.8,\ 0.2+0.9,\ 0.9+1.4\}=1.7$

$R_1(c)=0.5+\min\{0.8+0.8,\ 0.8,\ 0.7+0.9,\ 0.5+1.4\}=1.3$

$R_1(g)=0.1+\min\{0.2+0.8,\ 0.7+0.8,\ 0.9,\ 0.1+1.4\}=1.0$

$R_1(t)=0+\min\{0.9+0.8,\ 0.5+0.8,\ 0.1+0.9,\ 1.4\}=1.0$

The minimal cost characters for node 1 are either **g** or **t**. The minimal cost character for node 3 is **g**. The maximum parsimony approach would prefer the other tree (exercise left to the reader).

2nd step of algorithm is to do *a pre-order* (**from root to leaves**) traversal of tree

– for **root node**: select **minimal cost character**

– for each **internal node**: select the **character** that **resulted in the minimum cost explanation of the character selected at the parent**

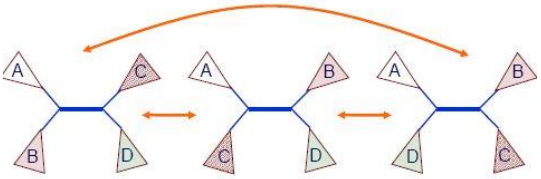
Exploring the space of trees

• we've considered how to **find the minimum number of changes** for a **given tree topology**

• need some search procedure for **exploring the space of tree topologies**

Heuristic method: nearest neighbor interchange

• for any internal edge in a tree, there are **3 ways the four subtrees can be grouped**
• **nearest neighbor interchanges** move from one grouping to another

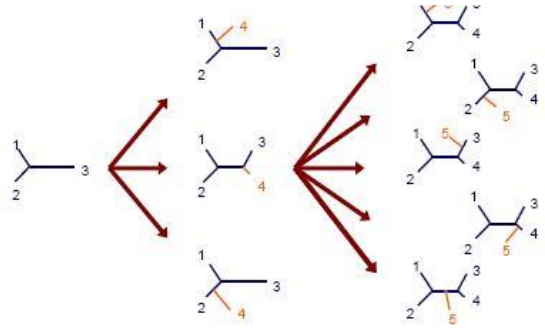


hill-climbing with nearest neighbor interchange

```
given: set of leaves L
create an initial tree t incorporating all leaves in L
best-score = parsimony algorithm applied to t
repeat
    for each internal edge e in t
        for each nearest neighbor interchange
            t' = tree with interchange applied to edge e in t
            score = parsimony algorithm applied to t'
                if score < best-score
                best-score = score
                best-tree = t'
    t = best-tree
until stopping criteria met
```

Exact method: branch and bound

• each **partial tree** represents a **set of complete trees**
• the **parsimony score on a partial tree** provides a ***lower bound* on the best score** in the set
• search by repeatedly **selecting and growing the partial tree** with the **lowest lower bound**



```
given: set of leaves L
use heuristic method to grow full initial tree t'       // quickly e.g. with Fitch's algorighm
initialize Q with a partial tree with 3 leaves from L
repeat
    t = tree in with lowest lower bound
    if has incorporated all leaves in L
        return t
    else
        create new trees by adding next leaf from L to each branch of t
        compute lower bound for each tree
        for each new tree n
            if lower-bound(n) < score(t')
            put n in Q sorted by lower bound
```

Rooted or unrooted trees for parsimony?

• we described parsimony **calculations in terms of rooted trees**
• but we described the **search procedures in terms of unrooted trees**
• *unweighted parsimony*: **minimum cost** is **independent of where root** is located
• *weighted parsimony*: **minimum cost** is independent of root if substitution cost is a metric (refer back to definition of metric from distance-based methods)

Comments on branch and bound

• it is a *complete* search method – **guaranteed to find optimal** solution

• may be much **more efficient than exhaustive search**
- in the worst case, it is no better

• **efficiency** depends on
– the **tightness of the lower bound**
– the **quality of the initial tree**

Comments on tree inference

• **search space** may be **large**, but
– can find the optimal tree efficiently in some cases
– **heuristic** methods can be applied

• **difficult to evaluate** inferred **phylogenies**: ground **truth not usually known**
– can look at **agreement** across **different sources of evidence**
– can look at repeatability across subsamples of the data
– can look at **indirect predictions**, e.g. conservation of sites in proteins

• some newer methods use data based on **linear order of orthologous genes** along chromosome

• phylogenies for **bacteria, viruses** not so straightforward because of ***lateral transfer*** of genetic material (not through ofspring); "local" phylogenies might be more appropriate

Probabilistic methods for phylogenetic tree reconstruction

Downsides to parsimony methods

• **Scoring function** parameters (costs for substitutions) are rather **arbitrary**
– The most "parsimonious" tree critically depends on these parameters

• Parsimony methods require assignments of character states to the ancestral nodes

– **Only considers score of best assignment**, which may not be the true one (the **probabilistic distribution would be better**)

Alternative to parsimony: probabilistic-model based tree scoring

• Instead of **cost *S(a,b)*** of a substitution occurring along a branch, we will use a **prob. *P(child = a|parent= b)***

• For a given tree, **instead** of finding a ***minimal cost assignment*** to the ancestral nodes, we will ***sum the probabilities of all possible ancestral states***

• Instead of finding a **tree with *minimum cost*** we will find **a tree the *maximizes likelihood*** (probability of the data given the tree)

Probabilistic model setup

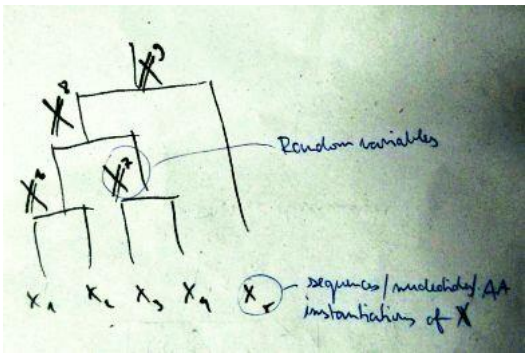
• We observe *n* sequences $x^1, x^2, \dots, x^n$

• We are **given a tree *T*** and want to model $\boldsymbol{P(x^1, x^2, \dots, x^n|T)}$
– This is the ***likelihood*** (probability of the observed sequences given the model, the tree)

• For **simplicity**, we'll just consider the case that **our sequences are of length 1** (just one character)

• To generalize to longer sequences, we assume ***independence* of each position** (position = each column of an ungapped multiple alignment)
– **Probability of sequences** = **product of probability of each position/column**

It will be easier to first consider a model in which we **represent the states of the internal nodes of the tree with random variables** $X^{n+1}, \dots, X^{2n+1}$ (assuming rooted binary tree, where $x^1 \dots, x^n$ are sequences):



• Then the **probability of any <u>particular configuration</u> of states at all nodes** in the tree will be defined as:

$$P(x^1, x^2, \ldots, x^n|T) = q_{x^{2n+1}} \prod_{i=1}^{2n-2} P(x^i|x^{\alpha(i)})$$

• $q_{x^{2n+1}}$is the prior probability of the state of the root node
• $\alpha(i)$is the index of the parent node of node *i*

• Key assumption: **state of node *i* is conditionally independent** of the states of **its ancestors given** the state of its **parent**

• For simplicity, we are ignoring branch lengths for now

The likelihood

• We only care about the probability of the observed (extant) sequences

• Need to marginalize (sum over possible instantiations of ancestral states) to obtain the likelihood:

$$P(x^1, x^2, \ldots, x^n|T) = \sum_{x^{n+1}, \ldots, x^{2n+1}} q_{x^{2n+1}} \prod_{i=1}^{2n-2} P(x^i|x^{\alpha(i)})$$

• But there is an exponential number of terms in this sum!
  - **dynamic programming** to the rescue once again!

Felsenstein's algorithm

• **Initialize**: $k = 2n - 1$
• **Recursion**:

– If *k* is a **leaf node** $\quad P(L_k|a) = \begin{matrix} 1 \; if \; a = x^k \\ 0 \; otherwise \end{matrix}$

– Else, compute $P(L_i|a)$ and $P(L_j|a)$ for all *a* at **daughters *i* and *j*:**

$$P(L_k|a) = \sum_b P(b|a)P(L_i|b) \sum_c P(c|a)P(L_i|c)$$

b and *c* represent the **states of node *i* and node *j***, respectively

• **Termination** – likelihood is equal to:

$$\sum_a P(L^{2n-1}|a)q_a$$

Concluding remarks on probabilistic-model (likelihood) based approach

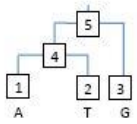
• Very **similar to the weighted parsimony** case
  – Main differences are at
    • Leaf nodes (they are assigned valeus 0 and ∞ vs. 0 and 1)
    • **Minimization** versus **summation** for internal nodes

• Can it be used to **infer ancestral states** as well?
  – Instead of **summing**, we would **maximize**
  – As in the parsimony case, we would need to keep track of the maximizing assignment

• **Substitution probabilities *P(a|b)* can be derived from principled mathematical models** and/or **estimated from data**

Example

What is probability for the following set of residues



b

a	A	C	G	T
A	0.7	0.1	0.1	0.1
C	0.1	0.7	0.1	0.1
G	0.1	0.1	0.7	0.1
T	0.1	0.1	0.1	0.7

Assume the above conditional probability matrix P(b|a) for all branches

	A	C	G	T
$P(L_1\|x)$	1	0	0	0
$P(L_2\|x)$	0	0	0	1
$P(L_3\|x)$	0	0	1	0
$P(L_4\|x)$	0.07	0.01	0.01	0.07
$P(L_5\|x)$	0.0058	0.0022	0.0154	0.0058

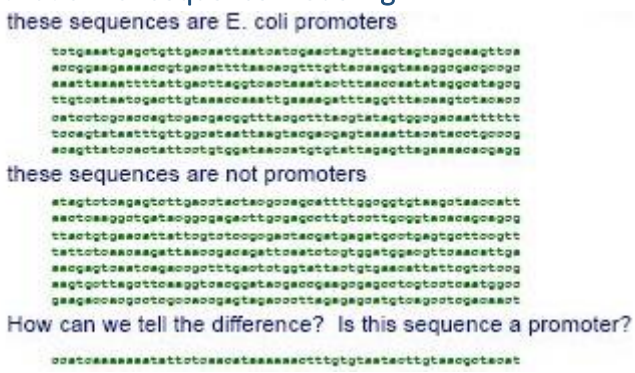
Probability of sequence given tree is 0.25(0.0058+0.0022+0.0154 + 0.0058)=0.0073

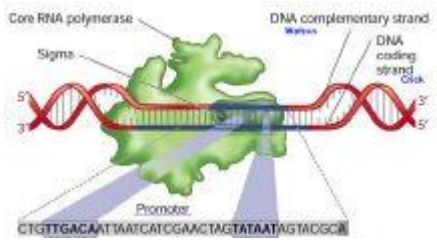
For instance $P(L_4|A)$ was calculated as:

$$P(L_k|a) = \sum_{b=\{A\}} P(b|a)P(L_1|b) \sum_{c=\{T\}} P(c|a)P(L_2|c) =$$
$$P(A|A)P(L_1|A)P(T|A)P(L_2|T) = 0.7 \cdot 1 \cdot 0.1 \cdot 1 = 0.07$$

Markov Chain Models

Motivation for sequence modeling

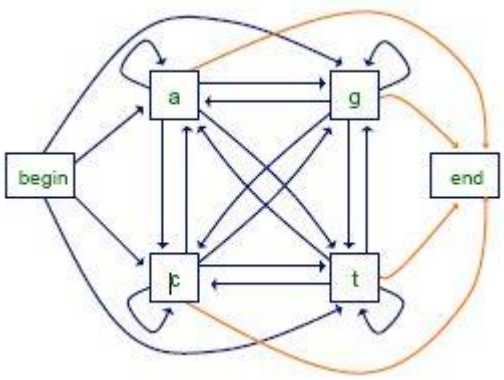




there are many cases in which we would like to represent the statistical regularities of some class of sequences

– **genes** (which parts of DNA likely encode genes?)
– various **regulatory sites in DNA** (which regions are e.g. **promoters?**)
– **proteins** in a **given family** (does a given protein belong to this family?)

• a **Markov chain** model is defined by
  – a **set of states**
    • some **states *emit* symbols**
    • other states (e.g. the *begin* and *end* states) are *silent*
  – a **set of transitions with associated probabilities**
    • the **transitions** emanating **from a given state** define a distribution over the possible next states



$$P(x_i = a|x_{i-1} = g) = 0.16$$
$$P(x_i = t|x_{i-1} = g) = 0.34$$
$$P(x_i = g|x_{i-1} = g) = 0.38$$
$$P(x_i = c|x_{i-1} = g) = 0.12$$

can also have an *end* state; allows the model to represent

– a distribution over sequences of different lengths
– **preferences** for **ending** sequences with **certain symbols**

Markov chain models

• from the chain rule we have:

$$P(X) = P(X_1)P(X_2|X_1)P(X_3|X_1X_2) \dots$$

• key property of a (1st order) Markov chain: **the probability of each X_i depends only on the value of X_{i-1}**:

$$\boldsymbol{P(X)} = P(X_1)P(X_2|X_1)P(X_3|X_1X_2) \dots = \boldsymbol{P(X_1)} \prod_{\boldsymbol{i=2}}^{\boldsymbol{L}} \boldsymbol{P(X_i|X_{i-1})}$$

Example:

$$P(cggt) = P(c)P(g|c)P(g|g)P(t|g)P(end|t)$$

Estimating the model parameters

• **given** some **data**, how can **we determine the probability parameters** of our model?
• one approach: ***maximum likelihood estimation***
  – given a set of data *D*
  – set the parameters θ to maximize $P(D|\theta)$
  – i.e**. make the data *D* look as likely as possible under the model**

Example: suppose we want **to estimate the parameters *P(a), P(c), P(g), P(t)*** and we're given the sequences

accgcgctta
gcttagtgac
tagccgttac

• then the maximum likelihood estimates are:

$$P(a) = \frac{6}{30} = 0.2, \quad P(g) = \frac{7}{30} = 0.233, \quad P(c) = \frac{9}{30} = 0.3, \quad P(t) = \frac{8}{30} = 0.267$$

• problem arises when there's **no occurence of some observant** – then the **parameter** becomes **0**

gccgcgcttg
gcttggtggc
tggccgttgc

P(a) = 0/30 = 0

• instead of **estimating** parameters strictly **from the data**, we could start with some **prior belief for each**
• for example, we could **use *Laplace estimates*:** $\boldsymbol{P(a)} = \frac{\boldsymbol{n_a+1}}{\sum_i (\boldsymbol{n_i+1})}$ → $P(a) = \frac{0+1}{34}$

or more general ***m-estimates:*** $\boldsymbol{P(a)} = \frac{\boldsymbol{n_a+p_a m}}{(\sum_i \boldsymbol{n_i})+\boldsymbol{m}}$ $\boldsymbol{P(c)} = \frac{9+0.25x8}{30+8}$ with *m=8*

where $\boldsymbol{p_a}$ is prior probability of a and ***m*** is number of „virtual" instances

• to **estimate a 1st order parameter**, such as ***P(c|g),*** we count the number of times that *g* follows the history *c* in our given sequences
• **using Laplace estimates** with the sequences above:

$$P(a|g) = \frac{0+1}{12+4}, \quad P(g|g) = \frac{3+1}{12+4}, \quad P(c) = \frac{7+1}{12+4}, \quad P(t) = \frac{2+1}{12+4}, \dots$$

Higher order Markov chains

• the **Markov property** specifies that the prob. of a state **depends only** on the **probability of the previous state**
  - but we can **build more "memory"** into our states by using a **higher order Markov model**

• in an ***n*th order Markov model**: $P(x_i|\boldsymbol{x_1}, \dots, x_{i-1}) = P(x_i|\boldsymbol{x_{i-n}}, \dots, x_{i-1})$

• additional history can have predictive value

• **Example:** predict the next word in this sentence fragment
"… the__" (duck, end, grain, tide, wall, …?)

Now predict it given more history
„… against the ___" (duck, end**, grain, tide, wall**, …?)
„swim against the ___" (duck, end**,** grain**, tide,** wall, …?)

Selecting the order of a Markov chain model

• but the number of parameters we need to estimate grows exponentially with the order

– for modeling DNA we need parameters for an *n*th order model (*n* 4-letter graphs interconnected)

• the **higher the order**, the **less reliable** we can expect our **parameter estimates to be**
– estimating the parameters of a **2nd order Markov chain** from the complete genome of ***E. Coli***, we'd see **each word > 72,000 times** on average
– estimating the parameters of an **8th order chain**, we'd see each word **~ 5 times** on average

• an ***n*th order** Markov chain over some **alphabet *A*** is **equivalent** to a **first order** Markov chain over the **alphabet A^n of *n*-tuples**

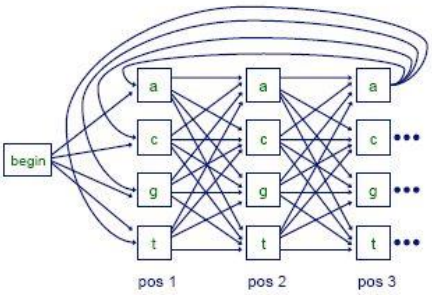
- **EX**: a **2nd order** Markov model for DNA = a **1st order** Markov model over alphabet (states)

AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, TT

- we process a **sequence one character at a time** A C G G T→ AC – CG – GG – GT

Inhomogenous Markov chains

• in the Markov chain models we have considered so far, **the probabilities do not depend on our position** in a given **sequence**



• in an ***inhomogeneous* Markov model**, we can have **different distributions at different positions**

• consider **modeling codons** in protein coding regions

Example application

• CpG islands
– **CG dinucleotides** are **rarer** in eukaryotic DNA **than expected given** the marginal **prob.** of **C** and **G**

– but the **regions upstream of genes are richer** in **CG** dinucleotides than elsewhere **– *CpG islands***
→ useful **evidence for finding genes**

• predict **CpG islands with Markov chains**
1. train **two Markov models**: one to represent **CpG island** sequence regions, another to **represent other sequence regions** (*null*)

2. **given a test sequence**, use two models to determine **probability that sequence is a CpG island**
– i.e classify the sequence (***CpG* or *null*)**

$P(c \mid a)$

+	*a*	*c*	*g*	*t*
a	.18	.27	.43	.12
c	.17	.37	.27	.19
g	.16	.34	.38	.12
t	.08	.36	.38	.18

CpG

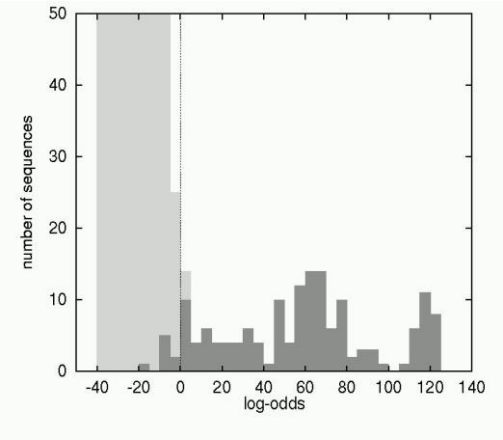
-	*a*	*c*	*g*	*t*
a	.30	.21	.28	.21
c	.32	.30	.08	.30
g	.25	.24	.30	.21
t	.18	.24	.29	.29

null

3. using **Bayes' rule** tells us: $P(CpG|x) = \frac{P(x|CpG)P(CpG)}{P(x)} = \frac{P(x|CpG)P(CpG)}{P(x|CpG)P(CpG)+P(x|null)P(null)}$

- where $\boldsymbol{P(x|CpG)}$ is obtained by **running the Markov chain inference**

- if we **don't take into account prior probabilities** of two classes $P(CpG|x)$ and $P(null|x)$ then we just need to **compare** $P(x|CpG)$ and $P(x|null)$

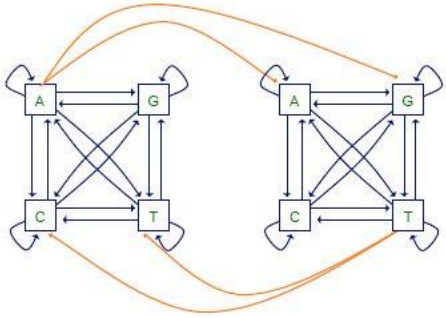


- **light bars** represent **negative sequences**
- **dark bars** represent **positive sequences** (i.e. **CpG** islands)
- **the dashed line** is the treshold. Here in **log** domain it's **0**, otherwise 1

Hidden Markov Models

The hidden part of the problem

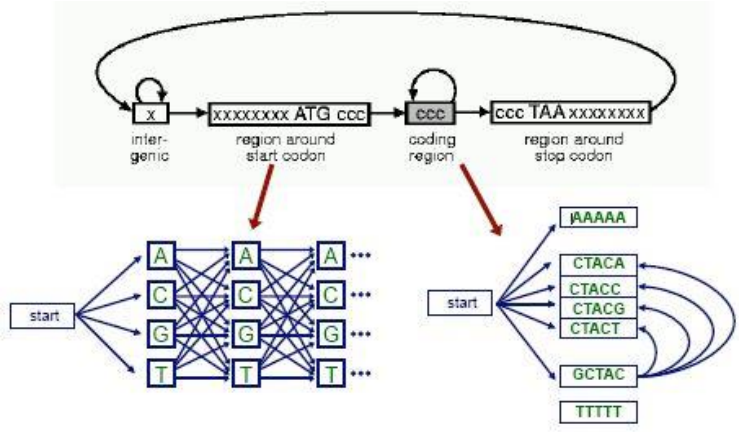
- we'll distinguish between the ***observed*** **parts** of a problem and the ***hidden*** **parts**
- in the **Markov models** we've considered previously, it is **clear which state accounts for each part** of the observed sequence
- in the model below, there are **multiple states** that could account for **each part of the observed sequence** – **<u>this is the hidden part of the problem</u>**



A typical question using HMM is e.g.:

Given say a **T** in our input sequence, **which state emitted** it?

Simple HMM for gene finding



The parameters of an HMM (transition and emittion)

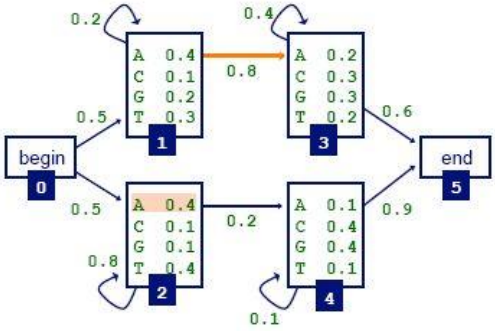
• as in Markov chain models, we have **transition probabilities** (at time *i*)

$$\alpha_{st} = P(\sigma_i = s | \sigma_{i-1} = t)$$ … probability of a **transition from state *s* to *t***

where $\boldsymbol{\sigma}$ represents a **path (sequence of states)** through the model

• since we've decoupled states and characters, we might also have **emission probabilities** (at time *i*)

$$e_s(c) = P(x_i = c | \sigma_i = s)$$ … probability **of emitting character c in state *s***



HMM with **5 states** with transition probabilites between them

Each state has also its **own emittion probabilities** of the characters

Three important questions

• **How likely is a given sequence?**
the Forward algorithm

• What is the **most probable "path" for generating a given sequence?**
the Viterbi algorithm

• **How can we learn the HMM parameters given a set of sequences?**
the Forward-Backward (Baum-Welch) algorithm

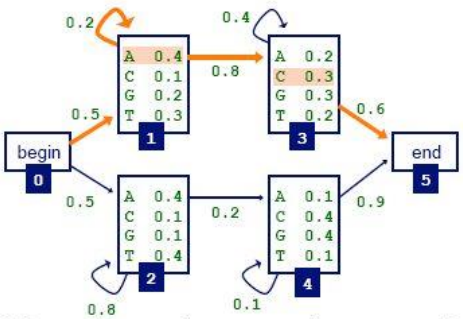
How likely is a given sequence?

• the probability that the path of states $\sigma_0 \dots \sigma_N$ is taken and the sequence $x_1 \dots x_L$ is generated:

$$P(x_1 \dots x_L, \sigma_0 \dots \sigma_N) = \prod_{i=1}^{L} e_{\sigma_i}(x_i) a_{\sigma_i \sigma_{i+1}}$$

i.e. product of probability **of transition between states** and probability of **emitting the symbol**

EX: How likely is that the sequence **ACC** was generated by the **orange path**?



$$P(\text{AAC}, \pi) = a_{01} \times e_1(\text{A}) \times a_{11} \times e_1(\text{A}) \times a_{13} \times e_3(\text{C}) \times a_{35}$$
$$= 0.5 \times 0.4 \times 0.2 \times 0.4 \times 0.8 \times 0.3 \times 0.6$$

• problem is there **is exponential number of paths** (for two states and seq. length *L* there are 2^L paths)
- how to **find the best one**? → **Forward algorithm (dynamic programming)**

Forward algorithm

• define $f_s(i)$ to be the **probability of being in state s** having **observed the first *i* characters** of *x*

• we want to compute $f_S(L)$, the probability **of being in the end state** having **observed all of** *x*

• can define this recursively

• because of the Markov property (**probability of the current state depends only on the states directly preceeding it**), **don't have to explicitly enumerate every path** – use **dynamic programming** instead

• e.g. assume state 4 can accessed from states 1 and 2, then we can compute $f_4(i)$ as **probability of emitting** the symbol at state 4 times **sum of product of probabilities of being in state 1,2 given *i-1* characters** $f_2(i-1)$ a $f_1(i-1)$ **times probability of transitioning from states 1,2 to state 4**

• the **dynamic programming matrix** will be calculated as (relation for **every state *l*** with **i chars read**):

$$f_l(i) = e_l(i) \sum_k f_k(i-1)\, \alpha_{kl}$$

- recursion for silent states $f_l(i) = \sum_k f_k(i)\, \alpha_{kl}$

- given the sequence x = TAGA
- initialization

$f_0(0) = 1 \quad f_1(0) = 0 \ \ldots \ f_5(0) = 0$

- computing other values

$$f_1(1) = e_1(T) \times (f_0(0)a_{01} + f_1(0)a_{11}) = 0.3 \times (1 \times 0.5 + 0 \times 0.2) = 0.15$$

$$f_2(1) = 0.4 \times (1 \times 0.5 + 0 \times 0.8)$$

$$f_1(2) = e_1(A) \times (f_0(1)a_{01} + f_1(1)a_{11}) = 0.4 \times (0 \times 0.5 + 0.15 \times 0.2)$$

• • •

$$P(TAGA) = f_5(4) = (f_3(4)a_{35} + f_4(4)a_{45})$$

Finding the most probable path: the Viterbi algorithm

• define $v_k(i)$, to be **the probability of the most probable path** accounting for **the first *i* characters** of *x* and ending in state *k*

• we want to compute $v_S(L)$, the probability of the most probable path accounting for all of the sequence and ending in the end state

- can define recursively, **use DP to find efficiently**

• **recursive definition**:

$$v_l(i) = e_l(x_i)\max_k[v_k(i-1)a_{kl}]$$

$$probable\_track_l(i) = \arg\max_k[v_k(i-1)a_{kl}]$$

- silent states $v_l(i) = \max_k[v_k(i)a_{kl}]$

$$probable\_track_l(i) = \arg\max_k[v_k(i)a_{kl}]$$

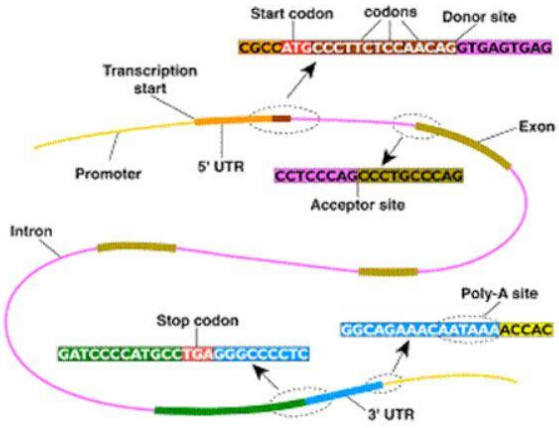
How can we learn the HMM parameters given a set of sequences?

• here the task is more difficult, than in the Markov models – there are **no annotated sequences**

• **parameters** can be learnt either in **general way (Bawn-Welsh EM algorithm)** or we can exploit **biological traits** and use **multiple sequence alignment:**

Applications

Given: an **uncharacterized DNA** sequence
Do: locate the **genes** in the sequence, including the **coordinates of individual *exons* and *introns***



Sources of evidence for gene finding

• **signals**: the sequence *signals* (e.g. splice junctions) involved in gene expression

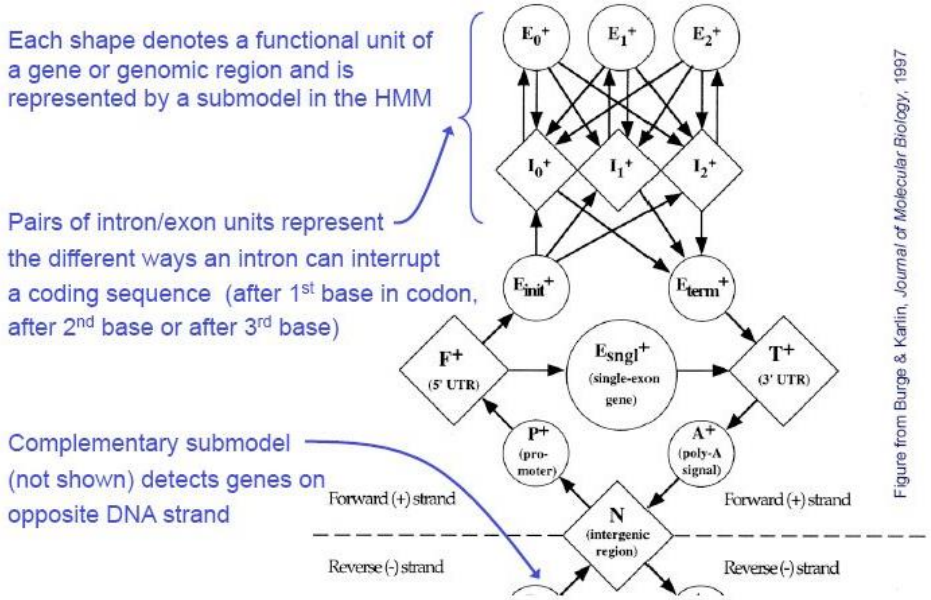
• **content**: statistical properties that distinguish proteincoding DNA from non-coding DNA

• **conservation**: signal and content properties that are conserved across related sequences (e.g. syntenic regions of the mouse and human genome)

```
UUU F 0.46   UCU S 0.19   UAU Y 0.44   UGU C 0.46
UUC F 0.54   UCC S 0.22   UAC Y 0.56   UGC C 0.54
UUA L 0.08   UCA S 0.15   UAA * 0.30   UGA * 0.47
UUG L 0.13   UCG S 0.05   UAG * 0.24   UGG W 1.00

CUU L 0.13   CCU P 0.29   CAU H 0.42   CGU R 0.08
CUC L 0.20   CCC P 0.32   CAC H 0.58   CGC R 0.18
CUA L 0.07   CCA P 0.28   CAA Q 0.27   CGA R 0.11
CUG L 0.40   CCG P 0.11   CAG Q 0.73   CGG R 0.20

AUU I 0.36   ACU T 0.25   AAU N 0.47   AGU S 0.15
AUC I 0.47   ACC T 0.36   AAC N 0.53   AGC S 0.24
AUA I 0.17   ACA T 0.28   AAA K 0.43   AGA R 0.21
AUG M 1.00   ACG T 0.11   AAG K 0.57   AGG R 0.21

GUU V 0.18   GCU A 0.27   GAU D 0.46   GGU G 0.16
GUC V 0.24   GCC A 0.40   GAC D 0.54   GGC G 0.34
GUA V 0.12   GCA A 0.23   GAA E 0.42   GGA G 0.25
GUG V 0.46   GCG A 0.11   GAG E 0.58   GGG G 0.25

[Codon/a.a./fraction per codon per a.a.]
Homo sapiens data from the Codon Usage Database
```

Search by content:
encoding a protein affects the statistical properties of a DNA sequence

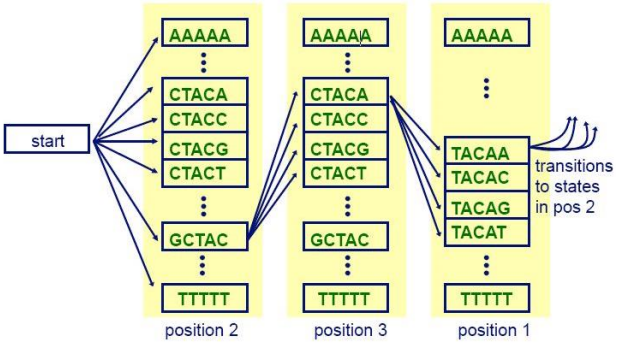
The GENSCAN HMM for Eukaryotic Gene Finding



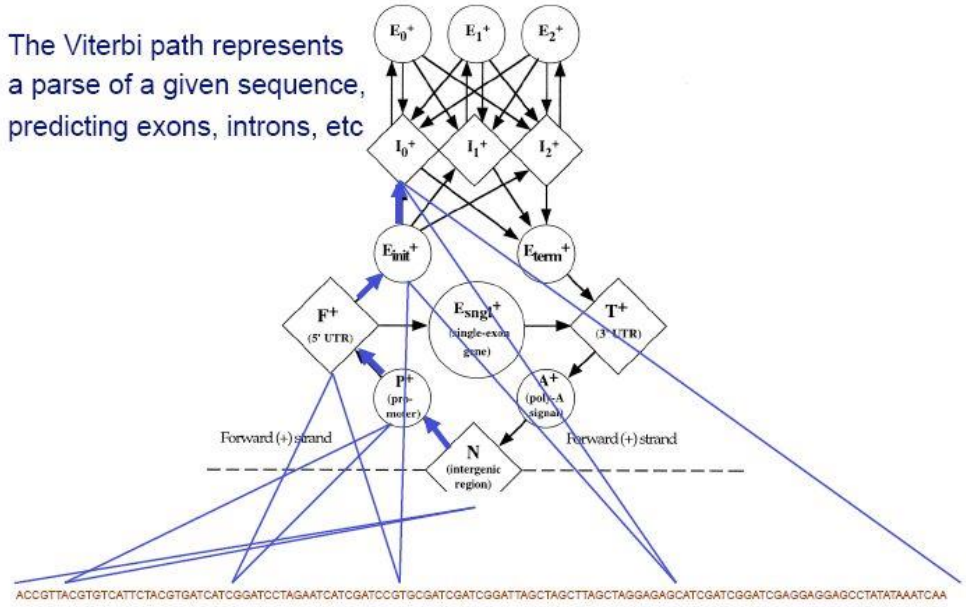
GENSCAN uses a variety of submodel types

sequence feature	model
exons	5th order inhomogenous
introns, intergenic regions	5th order homogenous
poly-A, translation initiation, promoter	0th order, fixed-length
splice junctions	tree-structured variable memory

In case of exons, we can use a **reading frame of 3 letters (codon)** with a **5th order Markov chain**



- given a sequence, find the **most probable path through the model for the sequence**
- this **path** will **specify the coordinates** of the predicted genes (including **intron** and **exon boundaries**)
  - the **Viterbi algorithm** is used to compute this path



Statistical Microarray Data Analysis

Method of **high-throughput screening using microarray data**

Hypothesis are induced to **find significantly differentially expressed genes** (because of insufficient number od data and highly complex mutual influence we consider **gene sets instead** – we just have too many attributes and too few samples to work with genes alone)

Measuring RNA abundances

• **changes in phenotype** don't have to have a analogy in the DNA (may be **caused by epigenetic** …).
It's therefore suitable to **measure the abundance on the level of proteins**. This abundance is however **difficult/expensive**. We must move a level lower and **we measure the inference between RNA** and **phenotype**.

• what is **varied**: individuals, strains, cell types, environmental conditions, disease states, etc.
  - each of these elements influence the gene expression
• what is **measured**: RNA quantities for thousands of genes, exons or other transcribed sequences

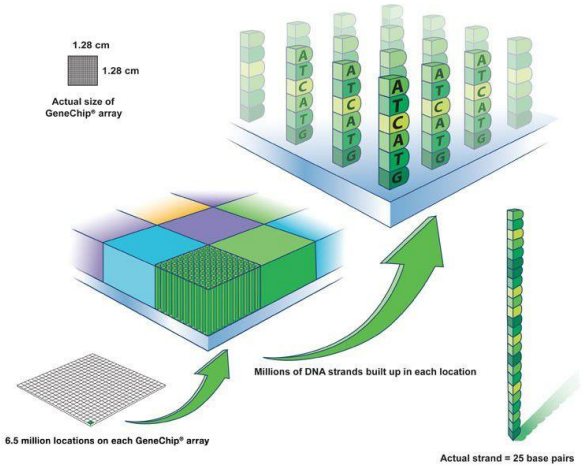
Oligonucleotide arrays

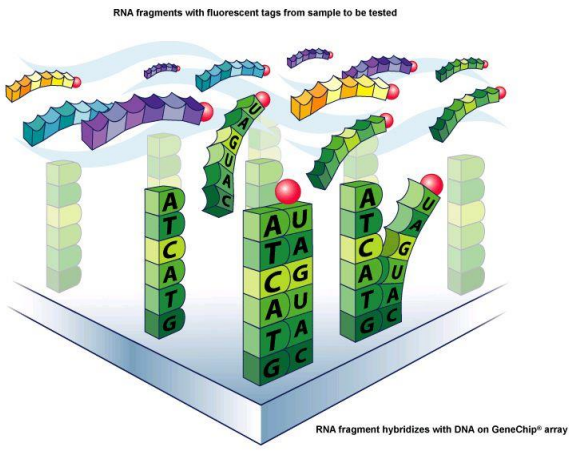
• **given a gene to be measured**, select **different *n*-mers** for the gene
  - can also select *n*-mers for noncoding regions of the genome

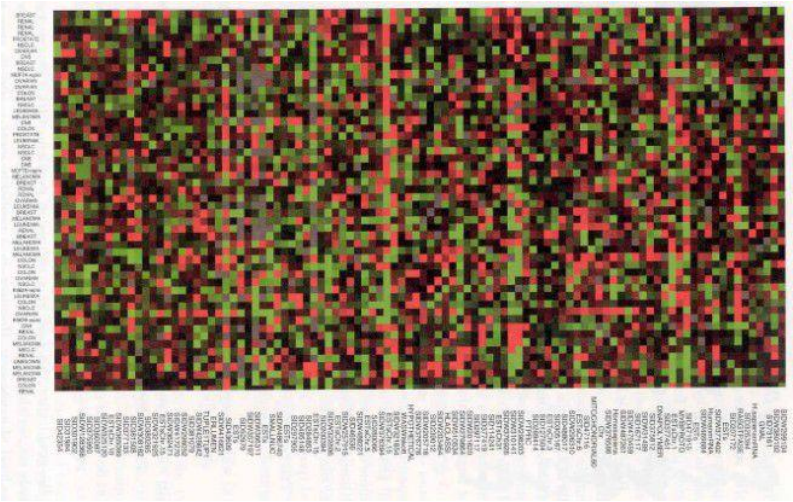
• **selection** criteria
  –**specificity** (has to **characteristic enough** for that gene)
  –hybridization properties
  –ease of manufacturing

Microarrays and Hybridization

• selected *n*-mers are placed on **squares** (one for each gene) - **pixels** of an image/chip
• sample **RNA is reversely re-written to cDNA** (complementary DNA)
• during the **hybridization** samples of **fragmented and colored cDNA** are applied on this chip
  - cDNA fragments will **bind to its counterpart** on the chip and thanks to the **coloring** we know which **genes are expressed**

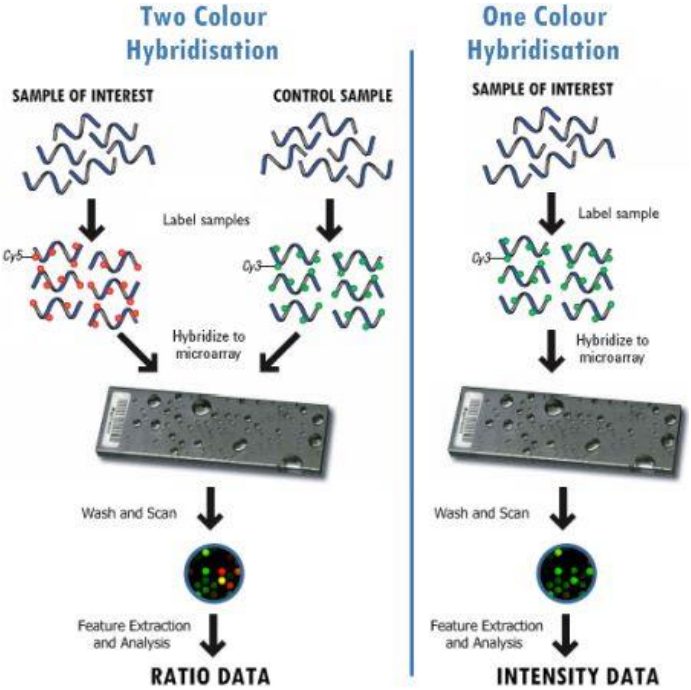






Microarray data
Each **row is a microarray of an individual** and each **column is an examined gene**

One-color vs two-color microarray



Goals of microarray data analysis

- Human **disease diagnostics and treatment**
  - disease **predispositions/risk factors**
  - monitor **disease stage** and **treatment progress**
- **Agricultural** diagnostics and development
  - find **plant pathogens** to improve plant protection
  - efficiacy and economy in plant biotechnology
- Analysis of **food and GMOs**
  - determine the integrity of food
  - detect alterations and contaminations

Other measurements

- apart from MAs, we can **characterize cells** in terms of **protein or metabolite** (small molecule) **abundances**
  - not as common as mRNA profiling, however, because the **technology for doing it is not as mature**
- also, there are **miRNA, SNP or DNA methylation arrays**.
  - changes **in gene expression** may be **not caused only by changes in exons** (and consequently in RNA), but also by **methylation changes** in the **promoter** region or **miRNA effects**

Ways of MA data analysis

• **Predictive modeling: molecular classifiers**

- large potential applicability, **but risk of low reliability** and comprehensibility
  – e.g., 70% accuracy is not enough when explanation is missing
  –decision based on a large number of genes is expensive
- SVM, RF, kNN, classification rules etc. where gene **expression are attributes**
- *classifying samples*: to **which class does a given sample belong**
- *classifying genes*: to **which functional class does a given gene belong** (what does it do)

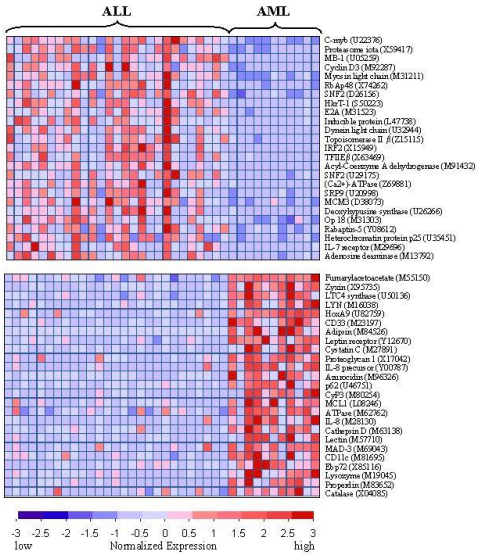
• rather simpler tasks of **Descriptive modeling**

- any **genes with similar expression profiles**?
  –clustering, bi-clustering
  –the genes potentially being regulated together
- any **genes potentially discriminating among classes**?
  –t-tests, SAM
  –potential risk factors
- can we **characterize these genes**?
  –significant GO terms, pathways, locations (chromozomes)
  –focus on human disease diagnostics and treatment

ALL/AML dataset

• **distinguishing** between **two acute leukemia** types
  – acute lymphoblastic leukemia (**ALL**); largely a **pediatric disease**
  –acute myeloid leukemia (**AML**), the most frequent **leukemia form in adults**

• microarray chip with **probes for 7129 genes**, **72 class-labeled samples** (47 ALL, 25 AML)



T-test,
Familywise error rate = mushroom picking (as much mushrooms and as little poisonous as possible)
False Discover Rate = rasperberry picking – false positive shouldn't be higher than a specified rate)
36 genes have w=83 (null distr. Expects cca 15) than FDR would be 40%. If it is not sufficient for us, we can move more to the right, where the FDR decreases
Problem with this technique is that it assumes genes comes from iid distribution. But genes usually ho

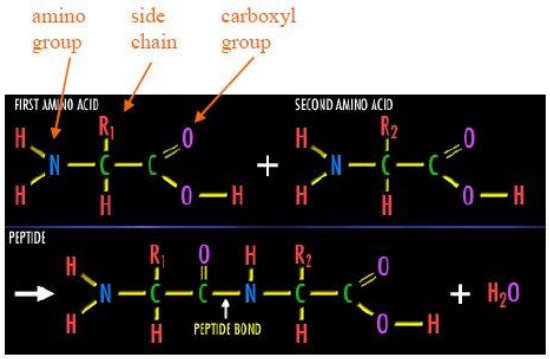
Protein Structure Prediction

The Protein Folding Problem

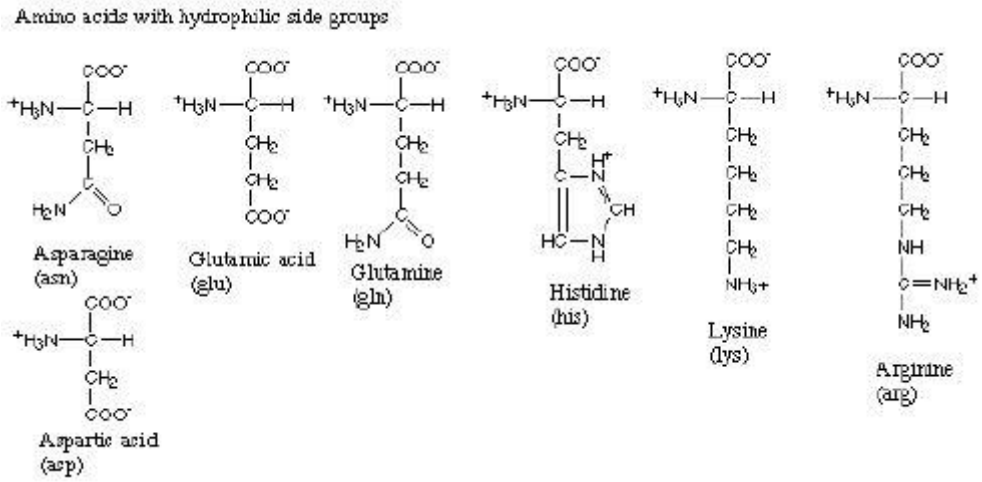
- we know that the **function** of a protein is determined by its **3D shape** (*fold*, *conformation*)
- in general, we **can't predict the 3D shape** of a protein given only its **amino-acid sequence**
  - but methods that give us a *partial* description of the 3D structure are still helpful

Protein Architecture

- proteins are **polymers** consisting of **amino acids** linked by ***peptide* bonds**
- each amino acid consists of
  - a **central carbon** atom
  - an **amino group NH_2**
  - a **carboxyl group COOH**
  - a **side chain**



- **differences in side chains** distinguish **different amino acids:**



- **side chains** vary in: **shape, size, polarity, charge**

What Determines Fold?

- in general, the **amino-acid sequence** of a protein **determines the 3D shape** of a protein
- but some **exceptions:**
  - all proteins can be denatured
  - some molecules have multiple conformations
  - some proteins **get folding help from *chaperones***
  - ***prions*** can change the conformation of other proteins
- what **physical properties** of the protein determine its fold?
  - rigidity of backbone (peptide skeleton)
  - **interactions among amino acids**, including

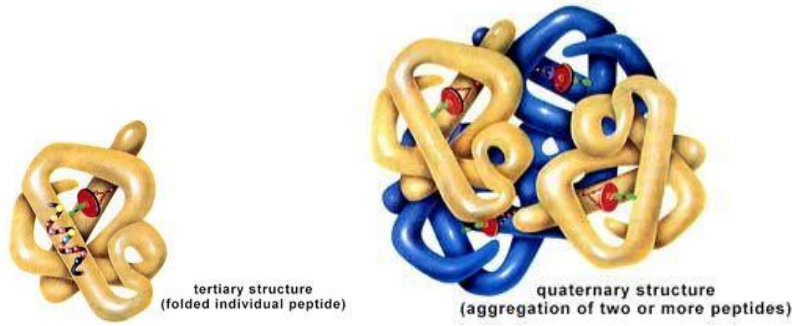
- **electrostatic** interactions
- **van der Waals** forces
- **volume constraints**
- **hydrogen, disulfide bonds**

– interactions of **amino acids with water**

Levels of Description

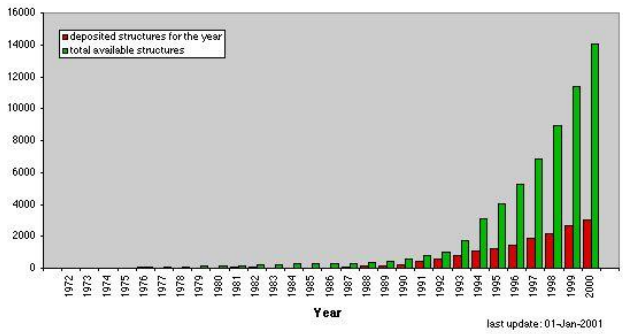
- **secondary** structure refers to certain **common repeating structures**
  - it is a "local" description of structure
- **2 common secondary structures**
  - σ helices
  - β sheets
- a **3rd category**, called ***coil*** or ***loop***, refers to everything else (they inter-connect protein parts)

σ helices β sheets and coils



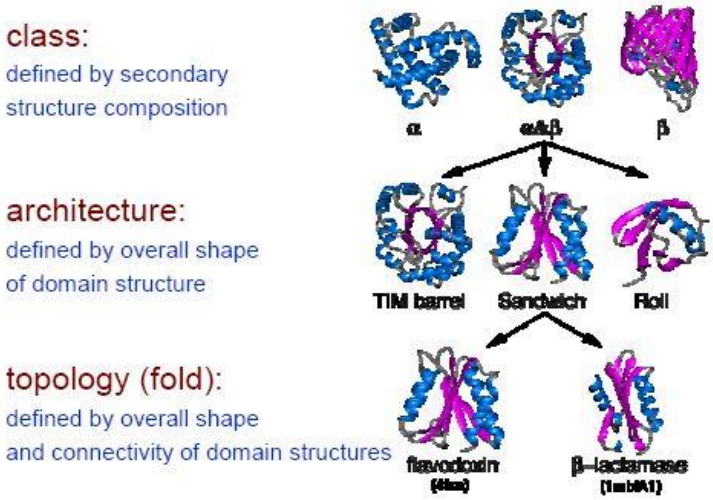
Determining Protein Structures

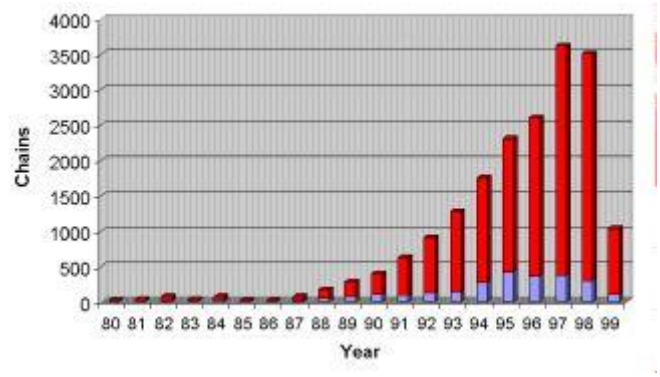
- protein **structures** can be determined experimentally (in most cases) by
  - **x-ray crystallography**
  - **nuclear magnetic resonance** (NMR)
- but this is very expensive and time-consuming



Comparison of:
new protein <u>structure</u> entries (orange - SWISS-PROT) and
new protein entries (green – DPB)

Top Levels of CATH Taxonomy





Comparison of:
new protein fold entries (blue) and
old protein fold entries (red)

We see there is **no such a increase in new CATH folds ->** proteins are just combinations of the existing folds

Approaches to Protein Structure Prediction

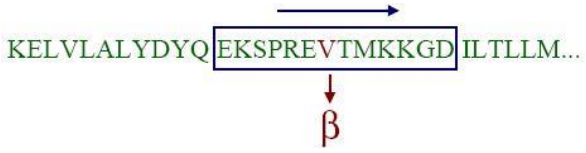
- prediction in **1D**
  - – secondary structure (what part is helix, what part is sheet)
  - – **solvent accessibility** (how is a part accessbile to water – hydrophobic in the center of the sequence)
  - – transmembrane helices
- prediction in **2D**
  - – input is a matrix of AA residues and their chemical attributes
  - – predicting inter-residue/strand contacts
- prediction in **3D**
  - – homology modeling
  - – fold recognition (e.g. via threading)
  - – *ab initio* prediction (taking into account all chemical properties; extremely comples)

Secondary Structure Prediction

- **given**: an amino-acid sequence
- **do:** predict a secondary-structure state (a, b, coil) for each residue in the sequence

KELVLALYDYQEKSPREVTMKKGDILTLLM...
cccββββccccccccccccββββcccccββββββ...

1) make **prediction for a given residue** by considering a **window of n** (typically 13-21) **neighboring residues**
2) **learn model** that performs **mapping from window of residues to secondary structure** state



Homology Modeling

• **observation**: proteins with **similar sequences** tend to fold into **similar structures**

• **given**: a **query sequence** Q, database of protein structures

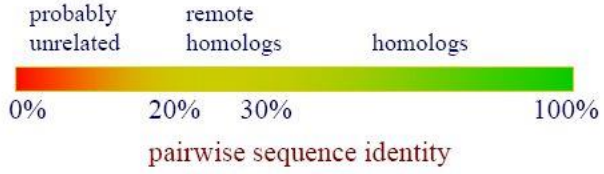
• **do**: – find protein P such that
- **structure of P is known**
- **P has high sequence similarity to Q**

– return **P's structure** as an **approximation** to **Q's structure**

• **homologs** – proteins with **different DNA**, but **similar function** thus **similar structure**

• most pairs of proteins with similar structure are **remote homologs** (< 25% sequence similarity)

• **homology modeling usually doesn't work for remote homologs** ; most pairs of proteins with < 25% sequence identity are **unrelated**



Protein Threading

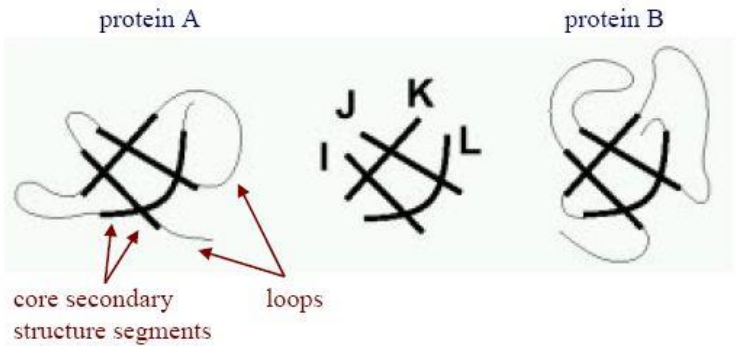
• generalization of homology modeling
- **homology modeling**: align sequence **to sequence**
– **threading**: align sequence to ***structure***

• key ideas
– **limited number of basic folds** found in nature

– amino acid preferences for different structural environments provides sufficient information to choose among folds

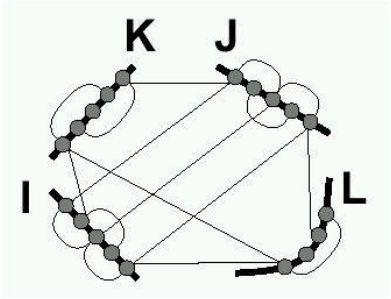
Components of Threading Approach

1. **library of core fold** templates
2. objective **function to evaluate** any particular **placement of a sequence into a core template**
3. method for **searching over space of alignments** between sequence and each **core template**
4. method **for choosing the best template given alignments**



We try to **map the proten B** into the **structure (fold) given by protein A**

Core Template with Interactions



• small **circles represent amino acid positions**

• **thin lines** indicate **interactions** represented in model (**Van der Vaals forces, electric forces** etc.)

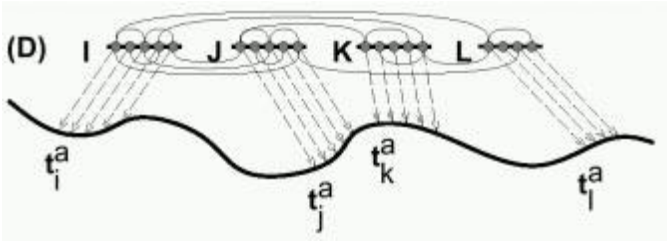
Objective Functions

• the objective function **scores the sequence/structure compatibility** between
  – **sequence** of amino acids
  – their corresponding **positions in the core template**

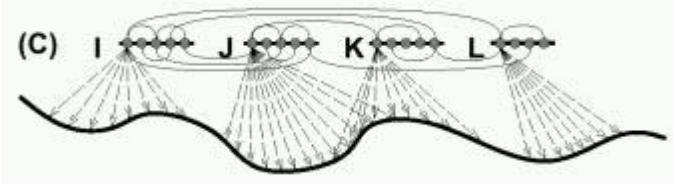
• it **takes into account factors** such as
  – a.a. preferences for **solvent accessibility**
  – a.a. preferences for **particular secondary structures**
  – **interactions** among spatially **neighboring amino acids**

Threading

• a threading can be represented as a **vector**, where each **element indicates the index of the amino acid** placed in **the first position of each core segment**



• finding the **optimal alignment is NP-hard** in the general case where
  – there are **variable length gaps** between the **core segments**
  – the **objective function** includes **interactions between neighboring amino acids**



Objective Function

$$f(\vec{t}) = \sum_{v \in V} f_{vertex}(v, \vec{t}) + \sum_{\{u,v\} \in E} f_{edge}(\{u, v\}, \vec{t}) + \sum_{\lambda \in \lambda_T} f_{loop}(\lambda_i, \vec{t})$$

$\vec{t}$ a vector characterizing a threading (each element indicates sequence position that starts each segment)

u, v amino acid positions in the core template

Searching the Space of Alignments

• **higher-order interactions <u>not</u> allowed** (only **α/β distinction**, **not** interaction between **each pair** of AAs)
  – dynamic programming

• **higher-order interactions allowed**
  – **heuristic** methods (fast but might not find the optimal alignment)
  – **exact** methods (e.g. **branch&bound,** which might take exponential time)

Branch and Bound Search

```
initialize Q with one entry representing the set of all threadings
repeat
    l ← set in Q with lowest lower bound
    if l contains only 1 threading
        return l
    else
        split l into smaller subsets
        compute lower bound for each subset
        put subsets in Q sorted by lower bound
```

• the **general objective function** with **pairwise interactions** is:

$$f(\vec{t}) = \sum_i g_1(i, t_i) + \sum_i \sum_{j>i} g_2(i, j, t_i, t_j)$$

first sum represents the **score** for **individual segments** (how well they fit the template)
second sum = scores for **segment interactions** (how chosen segments interact)

– create **subsets** so they **are easy to compte lower bounds** for to speed up