ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
Мэдээлэл, Холбооны Технологийн Сургууль

F.CS213 Биоалгоритм

Phylogenetic Analysis

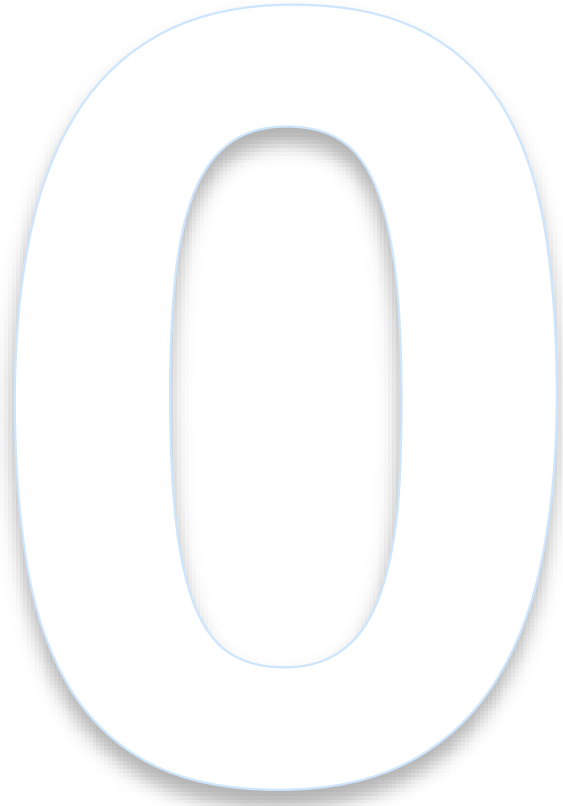
Филогенетик шинжилгээ

Лекц 8

Багш ***Ганбаатарын ГАНБАТ.*** *Өрөө* ***#304,*** *ganbatg@must.edu.mn,* ***99999467***

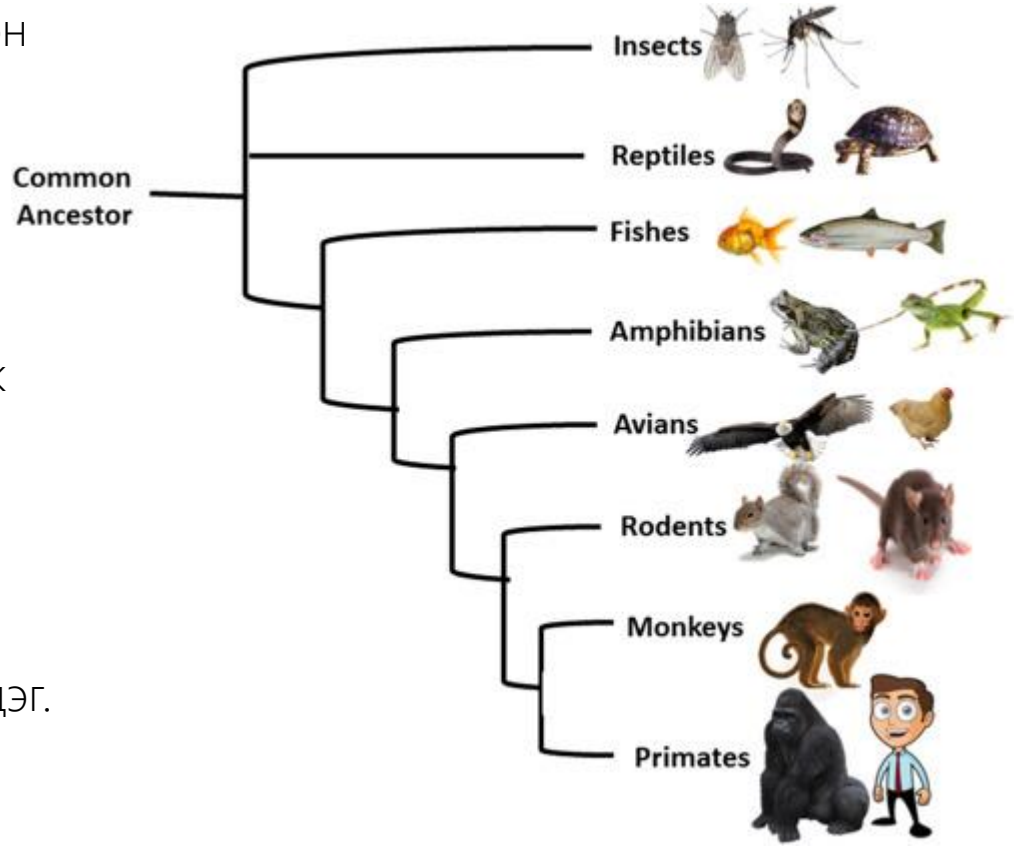
Лекцийн агуулга

- Удиртгал
  - Филогенетик мод
  - Филогенийн хэрэглээ ба тооцоолол
- Филогенетик шинжилгээ
  - Distance-Based арга
  - Maximum Parsimony
  - Статистик аргууд
- Distance-Based алгоритм хэрэгжүүлэлт
  - Хоёртын мод
  - UPGMA Алгоритм
- BioPython дахь Филогенетик шинжилгээ



Удиртгал

- *Филогенетик (Phylogenetics):* Төрөл зүйлсийн хувьслын түүх болон хамаарлыг судалдаг.
- ***Филогены дүгнэлт (inference):*** Удамшлын онцлог, голчлон морфологи дээр суурилан гаргадаг.
  - Төрөл зүйлсийн хувьлын хамаарлыг харуулах ***филогенетикийн мод***-оор дүрсэлдэг.
  - Мутаци нь ДНХ дээр явагддаг болохоор ДНХ дарааллууд нь энэхүү судалгааг хувьслын шууд бүтээгдэхүүн дээр суурилан илүү найдвартай болгож шинэчилсэн.
- Биоинформатик дэхь филогенетик шинжилгээ (analysis)
  - Дарааллуудын өгөгдсөн багц нь ***нийтлэг өвөг (common ancestor)***–өөс байгалийн хувьслын процессоор хэрхэн үүссэнийг тодорхойлоход чиглэдэг.
  - Төрөл зүйлсийн нийтлэг өвгөөс мутаци болон задрахыг хоёр (эсвэл олон) мөчрөөр илэрхийлдэг



УДИРТГАЛ Филогенетикийн мод

- *Навч (leaves):* Мэдэгдэж буй дарааллууд (ДНХ, РНХ, уураг).
- *Зангилаа (internal nodes):* Доорх дарааллуудынхаа нийтлэг өвөг.
- *Үндэс (root):* Бүх дарааллын хувьд нийтлэг өвөг (таха) байх давтагдахгүй зангилаа.

- Үндэстэй мод: Зангилаа бүрийнх нь хувьд кластер (дарааллуудын олонлог)-аар илэрхийлж болно:
  - $\{1, 2\}, \{3, 4\}, \{1, 2, 3, 4\}, \{1, 2, 3, 4, 5\}$
  - Зангилааны өндөр – хугацааны хэмжээ
  - Үндэснээс навч руу – цаг хугацаа

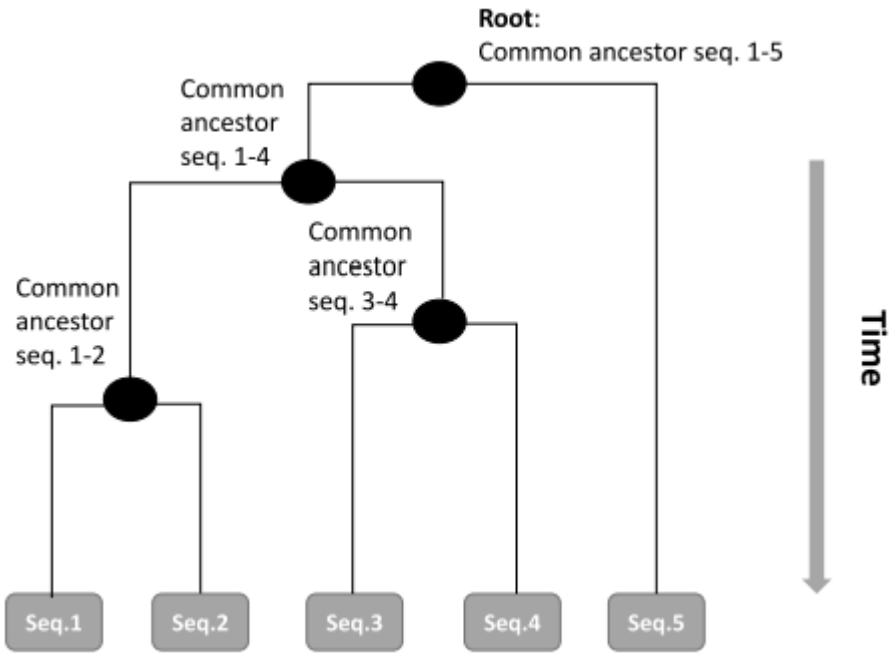
- Үндэсгүй мод нийтлэг өвгийг тодорхойлох бус навчис хоорондын хамаарлыг харуулахад чиглэнэ.

- Филогенийн хугацаа нь навч дээрх объектуудтай холбоотой.

- Филоген нь өндөр чанартай ДНХ-ийн дараалал (ерөнхийдөө рибосомын РНХ генийг (жнь 16S rRNA) кодлодог)-аас гарна.

- *The tree of life (*http://tolweb.org/tree/*)* гэх зэрэг зарим сонирхолтой төслүүд нь дэлхий дээрх бүх амьд организмуудын глобал филогенийг үүсгэхийг оролддог.

- ***Төрөл зүйл хоорондын (intraspecific)*** филогенийг зүйлүүдийн заагийг тодруулахад ашиглаж болно.



5 дараалал дээрх филогенийг илэрхийлэх мод.

УДИРТГАЛ Филогенийн хэрэглээ ба тооцоолол

- Филогенетик шинжилгээ нь нэлээд хэрэглээний ач холбогдолтой.
  - Шинэ төрөл, зүйлсийг ангилах, таксономын ангиллыг тодорхойлох.
  - Шүүх эмнэлэг, жнь, эцэг тогтоох, хүнсний хордолт эсвэл илрүүлэлт,
  - Халдвар судлал, эмгэг төрүүлэгчдийн дэгдэлт,
  - Олдмол мутацийн талаар илүү ихийг мэдэх, ховордсон амьтдыг хамгаалах бодлого
- Үүргийн аннотацын түвшинд уургийн дарааллын филогени нь дараалал зэрэгцүүлэлтийн тусламжтай:
  - Тухайн дарааллын үүргийг илүү сайн ойлгох,
  - Илэрсэн домайнуудыг функциональ үүрэгтэй нь холбох
- Филогенийг мөрдөн гаргах тооцоолол
  - Хамгийн боломжит хувьслын модыг тодорхойлох
  - Өгөгдсөн олонлог дахь дарааллууд хувьслын хамааралтай бол нийтлэг өвгөөс мөрдөн гарна.
  - Филогений мод хэд хэдэн хувилбартай байх боломжтой тул оновчлолын асуудал болно.
  - Дарааллын тоо багаар өсөхөд модны хувилбарын тоо маш хурдан өсдөг тул тооцооллын хүндрэлтэй болно.
    - Жнь: 30 дараалал байхад 10^{40} орчим хувилбарын модтой.

Филогенетикийн шинжилгээний алгоритмуудын анги

- Филогенетик шинжилгээний алгоритмуудыг зорилгын функцийг тооцоолох стратегиас хамаарч ангилдаг.
  - *Distance-based алгоритмууд:*
    - Дарааллууд дээрх ***хоёрлосон зайн (pairwise distance)*** матрицын тооцооолол дээр суурилдаг.
    - Оролтын матриц дахь зайтай нийцэж байгаа модыг хайж олох;
  - *Maximum parsimony:*
    - Дарааллуудын ялгааг тайлбарлахад зориулан мутацийн тоог хамгийн бага байх модыг олох.
  - *Statistical/Bayesian аргууд:*
    - Төрөл төрлийн мутаци үүсгэх магадлалтай загваруудыг тодорхойлж,
    - Дээрх магадлалд үндэслэсэн модыг байгуулахад тус загваруудыг ашиглах,
    - Таамагласан загварын дагуу дарааллыг тайлбарлах хамгийн их магадлалтай модыг хайж олдог.

Distance-Based аргачлал

Объектив функц: Дараалал зэрэгцүүлэх замаар модон дахь навчнуудын хоорондох *зай(distance)*-н тогтворжилтыг хэмжинэ.

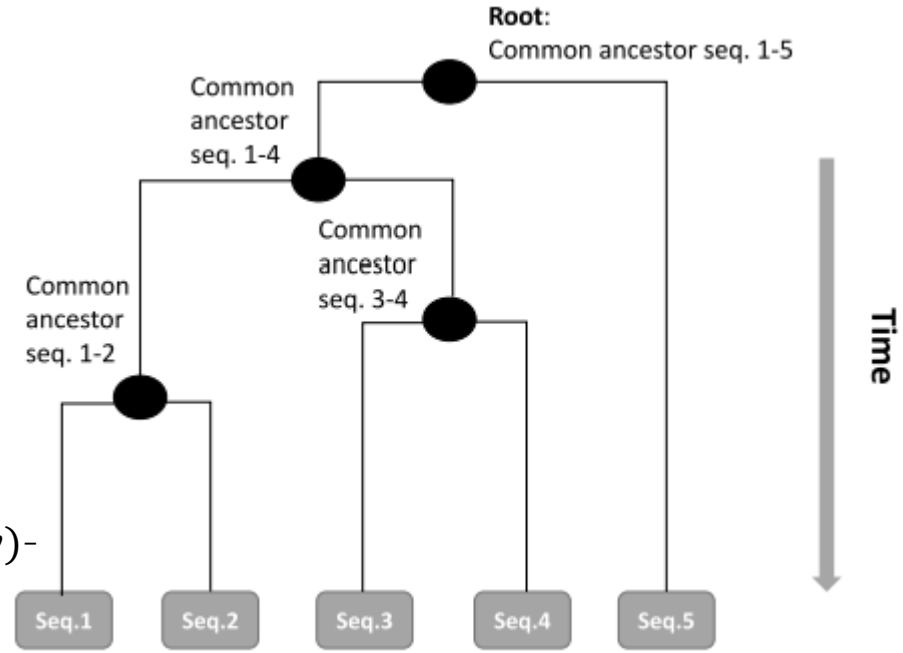
- Дарааллуудын хоёрлосон зайнууд руу модны бүтэц болон зангилаатай холбогдсон ирмэгүүдийн өндрийг тохируулахыг оролдоно.
- Зайнд суурилсан аргууд нь эхлээд дарааллын хоорондох зайн матрицыг тодорхойлдог.
  - Зай нь ижил төстэй байдлын урвуу байна
  - Дарааллуудыг зэрэгцүүлэх бөгөөд уг зэрэгцүүлэлт дээр үндэслэн зайг тооцоолно.
  - Энгийн арга: Зэрэгцүүлэлтийн *mismatch* эсвэл *gap* бүхий баганын хувийг тооцоолж болно (Бүлэг 6).
- Өгөгдсөн зайн матрицын хувьд зорилгын функцийг алдааны функц байдлаар тодорхойлж болно.
  - Модон дахь зай болон матриц дахь зайны хоорондох ялгааг багасгахад чиглэдэг.
  - Нэг нийтлэг арга бол квадрат алдааны нийлбэрийг авч үздэг:

$$score(T) = \sum_{i,j \in S} \left(d_{ij}(T) - D_{ij}\right)^2$$

- S нь оролтын дарааллууд
- T нь мод
- $d_{ij}(T)$ нь T модны i, j дараалал(навч)-ын зай
- $D_{ij}(T)$ нь оролт (дараалал зэрэгцүүлэлт)-ын D матрицын i, j дарааллын зай

Зай (Distance)

- Үндэстэй филогенетик T модны хоёр зангилаа (u ба v) хоорондын зай нь u–аас v хүрэх замын босоо зайтай уялдана.
- Хэрэв w нь u ба v хоёрын хамгийн ойрын нийтлэг өвөг бол u ба v хоорондын зайг $d_{uw} + d_{wv}$ байна. Жнь:
  - Зурган дээр $\mathbf{Seq\ 1}$ ба $\mathbf{Seq\ 4}$ навчнуудын хооронд зайг тооцохдоо нь эхлээд $\mathbf{Seq\ 1}$-ээс $\mathbf{Common\ ansester\ 1-4}$ нийтлэг өвөг рүү, дараа нь доошүү $\mathbf{Seq\ 4}$-руу очно.
  - Эдгээр хоёр зайг зангилааны өндрийн зөрүүгээр тооцдог:
    - $d_{uw} = h(w) - h(u)$ ба $d_{wv} = h(w) - h(v)$,
    - энд $h(x)$ нь x зангилааны өндөр
    - w нь u ба v хоёрын нийтлэг өвөг учир $d(w)$ нь $d(u)$ ба $d(v)$-ээс их байна.



- Молекул-цагны хувьд нь модны бүх мөчир дэх мутацийн хурд жигд байна гэж үзвэл бүх навч, үндэс хоорондын зай ижил байна
  - Энэ тохиолдолд мод нь *ultrametric* бөгөөд навчны өндрийг 0 гэж үзэж болно. Тэгвэл $d_{uv} = 2 \times h(w)$ байна.
- Эдгээр ерөнхий зарчмууд дээр суурилсан зорилгын функц бүхий хэд хэдэн арга танилцуулагдсан.
  - Харамсалтай нь дарааллын тоо өсөхөд шийдлийн орон зай маш хурдацтай нэмэгдэж, NP-hard гэж батлагдсан.
  - Шаардлагатай тооны дарааллын хувьд баталгаатай шийдийг үр ашигтайгаар тооцоолох алгоритм байдаггүй.

UPGMA

Практикт хэрэглэдэг ихэнх алгоритмууд нь хьюристик буюу тухайн асуудлын ихэнх тохиолдлын хувьд практикт боломжтой шийдлүүдийг өгдөг байна.

- *Арифметик дунджийг ашиглах жингүй хос бүлгийн арга (Unweighted Pair Group Method Using Arithmetic Averages)*-ын хамгийн энгийн бөгөөд түгээмэл нь *хуриттлан шаталсан кластерийн алгоритм (agglomerative hierarchical clustering algorithms)* дээр суурилдаг.
- Эхлээд дараалал бүрийг (модны навч) модны харгалзах кластертаа 0 өндөрт байгаа гэж үздэг.
- Алгоритм нь хамгийн ойрын хос дараалал/кластеруудыг хоорондох зайн хагастай нь тэнцүү өндөртэй зангилаа үүсгэн нэгтгэдэг.
- Эдгээр дарааллууд дараагийн алхамд зайг нь уг дарааллуудын зайн дундажаар тооцсон кластер болгон ашиглана,
- Холбогдсон дарааллуудын багана, мөрүүдийг арилган шинэ кластерын мөр, багана үүсгэх байдлаар D-г шинэчлэнэ.
- Дараачийн давталтуудад алгоритм нь хамгийн бага зайтай хос кластеруудыг олж, дараах алхмуудыг давтана:
  - кластеруудыг нэгтгэх,
  - өгөгдсөн өндөртэй модонд дотоод зангилаа нэмэх,
  - D зайны матрицыг шинэчлэх.
- Бүх дараалал нь модны үндэс дээр нэг кластер дотор харгалзах үед алгоритм зогсоно.

Жишээ - 1

(A) Distance matrix

	S1	S2	S3	S4
S2	2			
S3	5	4		
S4	7	6	4	
S5	9	7	6	3

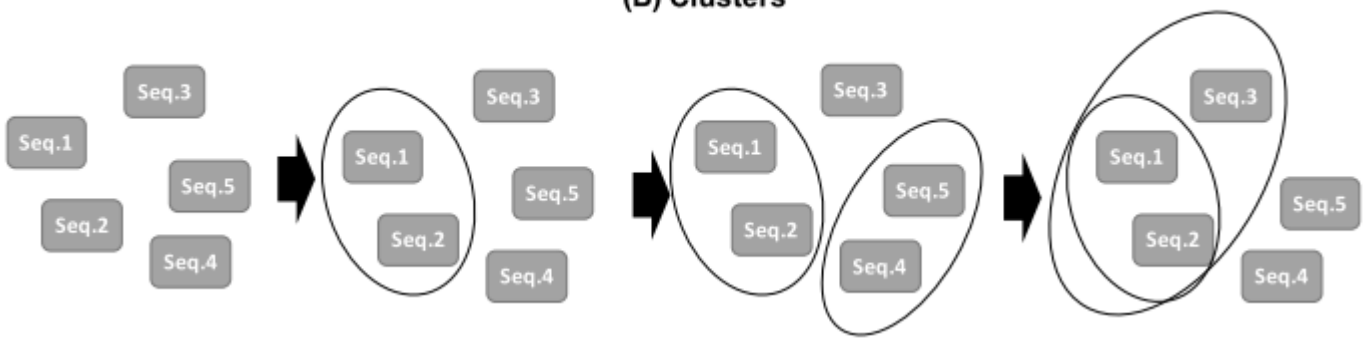
	S1, S2	S3	S4
S3	4.5		
S4	6.5	4	
S5	8	6	3

D	S1, S2	S3
S3	4.5	
S4, S5	7.25	5

D	S1, S2,S3
S4, S5	6.5

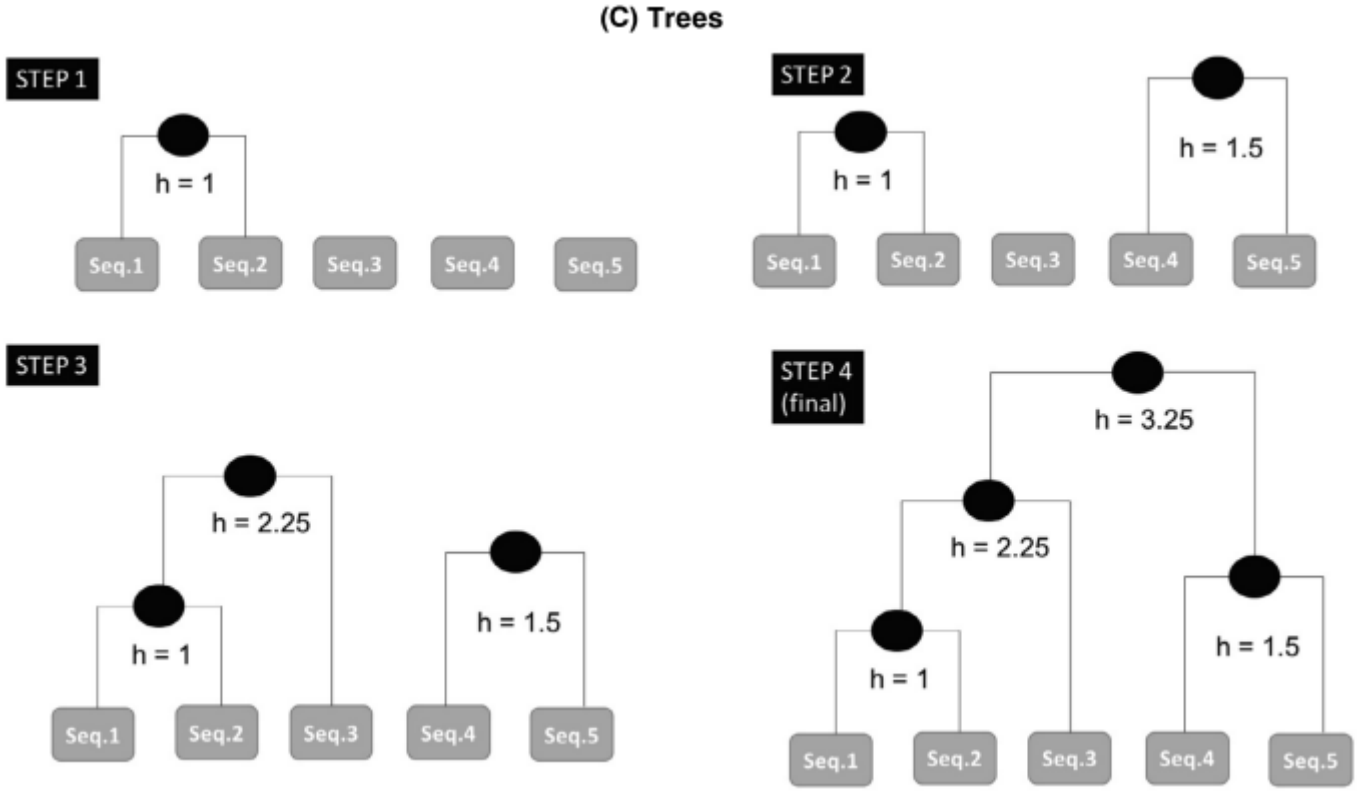
- Хамгийн зүүн талынх нь оролтын зайны матриц.
- D матрицыг шинэчилж, $\mathbf{Seq\ 1}$ ба $\mathbf{Seq\ 2}$-ын мөр, баганууудыг устгаж, шинэ кластерынхыг үүсгэж байна.

(B) Clusters



Жишээ - 2

- STEP 1: $\mathbf{Seq\ 1}$ ба $\mathbf{Seq\ 2}$-ыг нэгтгэн модны $h = 1$ өндөрт байрлах зангилааг үүсгэнэ.
  - кластер дахь зайг дарааллуудын (энэ тохиолдолд $\mathbf{Seq\ 1}$ ба $\mathbf{Seq\ 2}$) зайны дунджаар тооцоолно.
- STEP 2: Дараагийн алхамд $\mathbf{Seq\ 4}$ ба $\mathbf{Seq\ 5}$-ыг нэгтгэн кластер үүсгэж, өмнөхтэй төстэй явагдана.
- STEP 3: $\mathbf{Seq\ 1}$ ба $\mathbf{Seq\ 2}$-ын кластерыг $\mathbf{Seq\ 3}$-тай нэгтгэдэг бол STEP 4 нь үлдсэн хоёр кластерыг нэгтгэнэ.



Кластеруудын зай

- A ба B кластеруудын зайг тооцоолохдоо тэдгээрийн элементүүдийн бүх хосын зайн дунджаар тооцно.

$$\frac{1}{|A|.|B|}\sum_{i\in A}\sum_{j\in B} D_{ij}$$

- Алгоритмд кластеруудыг нэгтгэдэг. Хэрэв тухайн алхамд A ба B кластеруудыг нэгтгэж шинэ кластер ($A \cup B$) үүсэх бол шинэ кластераас X кластер хүртэлх зайг өмнө нь матрицад тооцсон зайны жигнэсэн дундажаар тооцоолно. Энд $D(X,Y)$ нь D дахь X ба Y кластеруудын зай.

$$D(A \cup B, X) = \frac{|A|.D(A,X) + |B|.D(B,X)}{|A| + |B|}$$

- Альтернатив хувилбар болох ***Арифметик дундаж тооцсон жинтэй хос бүлгийн арга (Weighted Pair Group Method with Arithmetic Mean - WPGMA)*** алгоритмд шинэ кластераас бусад руу нь хүрэх зайг арифметик дунджаар тооцдог.

$$D(A \cup B, X) = \frac{D(A,X) + D(B,X)}{2}$$

- UPGMA алгоритм нь энгийн, түгээмэл хэдий ч олон хязгаарлалттай.
  - Алгоритмд мутацийн хурдыг жигд байхаар авч үздэг тул моднууд нь ultrametric байна.
  - Хэрэв оролтын зайн матриц ultrametric байвал алгоритм нь оновчтой (optimal) шийдийг буцаана.
  - Гэсэн ч практикт бараг үнэн байдаггүй ба ихэнх нөхцөлд алдаатай үр дүнд хүргэдэг.

Neighbor joining-NJ

Хөршүүдийн нэгдэл (Neighbor Joining - NJ):
Өөр өөр удмын хувьд мутацийн хурдыг тогтмол байхыг шаарддаггүй.

- Үндэсгүй модны гаргалгаанд зорилгоор анх бүтээгдсэн бөгөөд зарим тохиолдолд үндэс нэмж үндэстэй болгодог.
  - Олон арга байдгийн нэг нь модны хамгийн хол хоёр навчийг холбосон үндсийг нэмнэ.
- UPGMA-тай харьцуулбал NJ-ийн гол ялгаа,
  - Нэгтгэх кластеруудыг сонгохдоо кластер хоорондын зайнаас гадна бусадаасаа хол байгааг нь сонгохыг зорьдог.
  - Үүний тулд анхны зайны матриц D-г ашиглан хамгийн ойрын кластеруудыг сонгоход туслах Q матрицыг үүсгэнэ.
  - Өөр хоорондоо хамгийн богино зайтай (эхний гишүүн)
  - Бусад кластеруудаас их зайтай (сүүлийн хоёр нийлбэрээр хэмжээг нь тодорхойлсон)

$$Q_{ij} = (n-2)D_{ij} - \sum_{k=1}^{n} D_{ik} - \sum_{k=1}^{n} D_{jk}$$

- Алгоритмын алхам бүрт D матрицыг дахин тооцоолно.
  - a болон b зангилааг холбож үүсгэсэн шинэ кластер нь u байхад бусад i кластер/зангилаанууд руу хүрэх зай:

$$D_{ui} = \frac{1}{2}(D_{ai} + D_{bi} - D_{ab})$$

Maximum Parsimony

Объектив функц: Оролтын дарааллуудыг тайлбарлахын тулд модноос мөрдөн гарч буй мутацийн тоог үнэлэх.

- *Occam-ийн сахлын хутганы зарчим (Occam's razor principle)*-ыг дагуу буюу өгөгдлийг тайлбарлах илүү богино модыг сонгох эрмэлзэлтэй.
- Дарааллуудын мутаци (gaps эсвэл mismatches), өөрчлөлтийг үүсгэж байгаа MSA баганын олонлогийг тодорхойлох
  - Энэ мэдээлэлд үндэслэн хамгийн цөөн мутацитай модыг илрүүлэхийг зорьдог.
- Хамгийн "*тансаг (parsimonious)*" модыг тодорхойлох хялбар арга - Бүх боломжит шийд болон тэдний өртөг жагсаах.
  - NP-hard, практикт хэрэгжих боломжгүй стратеги буюу дарааллын тоо 10-аас бага байхад шийдэгдэнэ.
- Өөр нэг хувилбар бол ***branch and bound*** аргуудыг хэрэглэх (10-р бүлэг).
  - Оновчтой шийд агуулаагүй нь тодорхой хайлтын хэсгийг ялгаж орхино
  - Практикт энэ нь оролтын дарааллын тоог 20 хүртэл нэмэгдүүлэх боломжтой.
- Илүү олон дарааллыг авч үзэхийн тулд хьюристик аргуудыг боловсруулах шаардлагатай.
  - nearest-neighbor interchanges, sub-tree pruning, and regrafting, tree bisection and reconnection,
  - Генетик алгоритмууд эсвэл аннеал симуляци зэрэг мета-хьюристик tree rearrangement аргууд,
- Ерөнхийдөө эдгээр аргууд нь модны мөчрүүд болон дарааллын мутациуд хоорондын хамаарлыг хялбар тодорхойлох давуу талыг бий болгодог. Гэсэн хэдий ч, дарааллууд нь хол байх үед илүү хязгаарлагдмал.

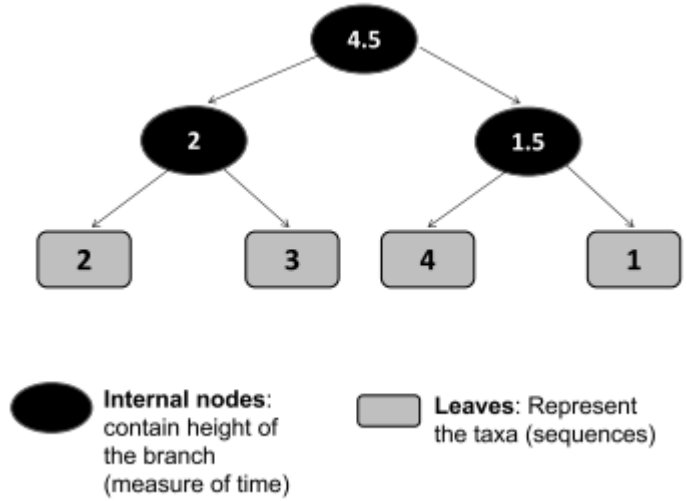
Statistical Methods

- Maximum likelihood аргууд нь ДНХ мутаци илрүүлэх статистик моделд суурилдаг.
  - Боломжит модны магадлал (likelihood)-ыг тооцохдоо модны мутацийн тохиолдол бүрийн магадлалыг үржүүлдэг.
  - Кимурагийн хоёр параметрт, Jukes-Cantor, Tamura-Nei зэрэг ДНХ хувьслын загварууд.
- Ерөнхийдөө тооцооллын хувьд нэлээд чанартай ч тооцооллын хүндрэл нь мөн NP-hard.
  - DP - Pruning algorithm нь дэд модны магадлалыг тооцоолох замаар хүндрэлийг багасгахад ашиглаж болно
- Maximum likelihood аргууд нь авч үзэж болох мутаци загваруудын хувьд маш уян хатан байдаг.
  - Нэг давуу тал нь удмууд дээр хувьслын ялгаатай хурдыг зөвшөөрснөөр статистикийн хувьд уян хатан байдлыг бий болгодог.
- *Байесийн (Bayesian)* дүгнэлтийн аргууд нь өмнөх аргуудтай ижил төстэй хувилбар юм.
  - Филогенетикийн боломжит моднуудад хэрэглэж байсан өмнөх магадлалын хуваарилалт дээр суурилдаг.
  - Хайлтын аргууд нь ерөнхийдөө Марковын гинжин хэлхээний Монте-Карлогийн алгоритмуудын хувилбаруудыг ашигладаг

Implementing Distance-Based Algorithms in Python

- UPGMA алгоритм.
- Дараалал хэрэгжүүлэлт, зэрэгцүүлэлтийн параметрүүд болон алгоритмууд дээр суурилсан үндсэн классууд (8.3-р хэсэг).

1. Хувьслын модыг (хоёртын мод хэлбэрээр) хэрэгжүүлэх класс
    - Хуримтлан шаталсан кластерын алгоритмыг хэрэгжүүлэх класст нийцүүлнэ
2. Биологийн дараалал бүхий UPGMA алгоритмыг оролт болгон ажиллуулах классыг хэрэгжүүлнэ.



Филогенетик модыг дүрслэх хоёртын модны жишээ.

IMPLEMENTATION

Binary Trees

Хувьслын модыг дүрслэхдээ “Хоёртын мод” өгөгдлийн бүтцийг ашиглана.

- Хоёртын мод ашиглаж байгаа учир бүх мутаци хоёр удам үүсгэнэ.
- Хоёртын мод нь Зүүн ба Баруун дэд модтой зангилаануудаар илэрхийлэгдсэн рекурсив өгөгдлийн бүтэц,
  - Навч – мод: зангилаанд мэдээллийг агуулдаг, дэд моднууд NULL,
  - Зангилаа (үндэс зэрэг) – мод: мэдээлэл агуулсан, баруун болон зүүн дэд модтой.
- Филогенетик модны хувьд
  - Навч– оролтын дарааллыг төлөөлөх
  - Зангилаа – мутаци тохиолдлыг илэрхийлэх салаалах цэг.
- Зангилаа нь өндөр (тоон утга) ба Навчнууд нь дарааллын индексийг (бүхэл тоо) агуулна.

Класс

```
class BinaryTree:

    def __init__(self, val, dist=0, left = None, right = None):
        self.value = val
        self.distance = dist
        self.left = left
        self.right = right

    def print_tree(self):
        self.printtreerec(0, "Root")

    def print_tree_rec (self, level, side):
        tabs = ""
        for i in range(level): tabs += "\t"
        if self.value >= 0:
            print(tabs, side, " – value:", self.value)
        else:
            print(tabs, side, "– Dist.: ", self.distance)
            if (self.left != None):
                self.left.print_tree_rec(level+1, "Left")
            if (self.right != None):
                self.right.print_tree_rec(level+1, "Right")
```

- BinaryTree класс - рекурсив:
  - *value* – бүхэл тоо, зангилаа бол –1, навч бол дарааллын индекс;
  - *distance* – зангилааны өндрийг хадгалах тоон утга (навч бол 0);
  - *left* ба *right* – зүүн ба баруун дэд мод; навч бол None байна.

```
def test():
    a = BinaryTree(1)
    b = BinaryTree(2)
    c = BinaryTree(3)
    d = BinaryTree(4)
    e = BinaryTree(–1, 2.0, b, c)
    f = BinaryTree(–1, 1.5, d, a)
    g = BinaryTree(–1, 4.5, e, f)
    g.print_tree()

if __name__ == '__main__':
    test()
```

get_cluster

```
class BinaryTree:

    def get_cluster(self):
        res = []
        if self.value >= 0:
            res.append(self.value)
        else:
            if (self.left != None):
                res.extend(self.left.get_cluster())
            if (self.right != None):
                res.extend(self.right.get_cluster())
        return res

def test():
    a = BinaryTree(1)
    (...)
    print(f.get_cluster())
    print(g.get_cluster())

if __name__ == '__main__':
    test()
```

- Тухайн зангилаан доор ямар навч (дараалал) байгааг тодорхойлох
  - Ө.х тухайн модонд харгалзах кластерыг буцааана.
- Get_cluster: алгоритм зохиохдоо ерөнхий бүтэц нь хоёртын модоор дагадаг.
  - *Зангилаа:* эхлээд зүүн дэд модны методыг рекурсив байдлаар дуудаж, дараа нь үр дүнг нэгтгэн баруун модны аргыг дуудна (энэ тохиолдолд үр дүнгийн хоёр багцыг нэгтгэх);
  - *Навч:* үр дүнг буцаах рекурсийг дуусгах (энэ тохиолдолд нэг утгатай олонлог).

IMPLEMENTATION

UPGMA Algorithm

```
class NumMatrix:

    def __init__(self, rows, cols):
        self.mat = []
        for i in range(rows):
            self.mat.append([])
            for j in range(cols):
                self.mat[i].append(0.0)

    def __getitem__(self, n):
        return self.mat[n]

    def num_rows (self):
        return len(self.mat)

    def num_cols (self):
        return len(self.mat[0])

    def get_value (self, i, j):
        if i>j: return self.mat[i][j]
        else: return self.mat[j][i]

    def set_value(self, i, j, value):
        if i>j: self.mat[i][j] = value
```

- Ролт нь зайны матрицыг хадгалах, удирдах боломжийг олгох NumMatrix класс.
  - мөр/баганын тоо буцаах, мөр ба баганын индексээр утгуудад хандах/тохируулах, матрицыг хэвлэх, мөр/багана нэмэх, хасах, матрицын хуулбарыг буцаах.
  - min_dist_indexes: Матрицын мөр, баганууудын хамгийн бага утгыг буцаадаг (0-үүдийг орхино).
- Матриц нь гурвалжин хэлбэртэй тул зөвхөн мөрийн индекс нь баганын индексээс их байх нүднүүдийг авч үзнэ (бусад нь 0).

```
def min_dist_indexes (self):
    m = self.mat[1][0]
    res= (1,0)
    for i in range(1,self.num_rows()):
        for j in range(i):
            if self.mat[i][j] < m:
                m = self.mat[i][j]
                res = (i, j)
    return res
```

IMPLEMENTATION > UPGMA

Хуримтлан шаталсан кластер алгоритм

- HierarchicalClustering.
  - Оролт болон зайн матрицын аттрибуттай.
- execute_clustering: Алгоритмыг ажиллуулж, үр дүнд нь хоёртын модыг буцаадаг үндсэн метод.
  - Модны олонлогийг эхлүүлэх
  - Навчнууд болон оролтын матрицыг үүсгэдэг.
- Үндсэн for цикл нь нэгтгэх кластеруудыг тодорхойлохоор матриц дахь хамгийн бага зайны индексүүдийг илрүүлдэг.
- Эдгээр хоёр кластертай харгалзсан мөчрүүдийг холбосон шинэ мод бий болно. Хэрэв энэ нь сүүлчийн давталт бол энэ модыг эцсийн үр дүн болгон буцаана.
- Үгүй бол алгоритм нь дараах процессуудыг хэрэгжүүлнэ.
  i. холбосон мөчрүүдийг модны жагсаалтаас устгана,
  ii. зайны матрицыг шинэчилнэ: холбосон кластеруудын мөр, баганыг арилгаж, шинэ кластерт шинийг мөр, баганыг нэмнэ
  iii. шинэ модыг давталтын дараагийн алхамд ашиглах идэвхтэй модны багцад нэмсэн.

```
def test():
    m = NumMatrix(5,5)
    m.set_value(0, 1, 2)
    m.set_value(0, 2, 5)
    m.set_value(0, 3, 7)
    m.set_value(0, 4, 9)
    m.set_value(1, 2, 4)
    m.set_value(1, 3, 6)
    m.set_value(1, 4, 7)
    m.set_value(2, 3, 4)
    m.set_value(2, 4, 6)
    m.set_value(3, 4, 3)
    hc = HierarchicalClustering(m)
    arv = hc.execute_clustering()
    arv.print_tree()
```

HierarchicalClustering

```
from BinaryTree import BinaryTree
from NumMatrix import NumMatrix

class HierarchicalClustering:

    def __init__(self, matdists):
        self.matdists = matdists

    def execute_clustering(self):
            ## initialization of the tree leaves and matrix
        trees = []
        for i in range(self.matdists.num_rows()):
            t = BinaryTree(i)
            trees.append(t)
        tableDist = self.matdists.copy()
        ## iterations
        for k in range(self.matdists.num_rows(), 1, -1):
            mins = tableDist.min_dist_indexes() ## minimum distan
   in D
            i,j = mins[0], mins[1]
            ## create new tree joining clusters
            n = BinaryTree(-1, tableDist.get_value(i, j)/2.0, trees[i
   ], trees[j])
            if k>2:
                ## remove trees being joined from the list
                ti = trees.pop(i)
                tj = trees.pop(j)
                ## calculating distances for new cluster
                dists = []
                for x in range(tableDist.num_rows()):
                    if x != i and x != j:
                        si = len(ti.get_cluster())
                        sj = len(tj.get_cluster())
                        d = (si*tableDist.get_value(i,x) + sj*
tableDist.get_value(j,x)) / (si+sj)
                        dists.append(d)
                ## updating the matrix
                tableDist.remove_row(i)
                tableDist.remove_row(j)
                tableDist.remove_col(i)
                tableDist.remove_col(j)
                tableDist.add_row(dists)
                tableDist.add_col([0] * (len(dists)+1))
                ## add the new tree to the set to handle
                trees.append(n)
            else: return n
```

Main Class

- Эцэст нь биологийн дараалалд дээр тодорхойлсон ерөнхий шаталсан кластерийн алгоритмыг хэрэглэх UPGMA ангиллыг тодорхойлох болно.
- Энэ анги нь модны навч (өмнө нь тодорхойлсон MySeq ангийн объектууд), зэрэгцүүлэх параметрүүд (PairwiseAlignment ангийн объект) болон зайны матриц (NumMatrix ангийн объект) зэрэг олон дарааллыг хадгалах шинж чанаруудтай байх болно.
- Доорх кодонд бид дэлхийн хэмжээнд тохируулсны дараа (Needleman-Wunsch аргатай) хоёр дарааллын хоорондох ялгаатай тэмдэгтүүдийн тооноос бүрдэх зайны хэмжүүрийг харгалзан энэ ангийн хэрэгжилтийг харуулж байна.
- Үүнийг matdist ангиллын хувьсагчийг дүүргэх create_mat_dist аргаар тооцдог.
- Энэ функцийг өөрчлөх эсвэл солих замаар бид бусад зайны хэмжүүрүүдийг хялбархан үүсгэж болохыг анхаарна уу.
- Run аргыг HierarchicalClustering ангийн объектыг үүсгэж, кластерын алгоритмыг гүйцэтгэж, үүссэн модыг буцаахад ашигладаг.

```
from NumMatrix import NumMatrix
from HierarchicalClustering import HierarchicalClustering
from MySeq import MySeq
from PairwiseAlignment import PairwiseAlignment
from SubstMatrix import SubstMatrix

class UPGMA:

    def __init__(self, seqs, alseq):
```

Main Class

- Биологийн дараалалд дээр тодорхойлсон шаталсан кластерийн генетик алгоритмыг хэрэглэх UPGMA класс.
  - модны навч (өмнө нь тодорхойлсон MySeq объектууд),
  - зэрэгцүүлэлтийн параметрүүд (PairwiseAlignment объект)
  - зайны матриц (NumMatrix объект)
- Глобал зэрэгцүүлэлтийн дараа (Needleman-Wunsch) хоёр дарааллын хоорондох ялгаатай тэмдэгтүүдийн тооноос бүрдэх зайн хэмжигдэхүүний хэрэгжилтыг харуулна
  - Үүнийг create_mat_dist методоор тооцоолж matdist класс хувьсагчид хадгална.
  - Функцийг өөрчлөх/солих замаар бусад зайны хэмжигдэхүүнийг хялбар үүсгэж болно.

```
from NumMatrix import NumMatrix
from HierarchicalClustering import HierarchicalClustering
from MySeq import MySeq
from PairwiseAlignment import PairwiseAlignment
from SubstMatrix import SubstMatrix

class UPGMA:

    def __init__(self, seqs, alseq):
        self.seqs = seqs
        self.alseq = alseq
        self.create_mat_dist()

    def create_mat_dist(self):
        self.matdist = NumMatrix(len(self.seqs), len(self.seqs))
        for i in range(len(self.seqs)):
            for j in range(i, len(self.seqs)):
                s1 = self.seqs[i]
                s2 = self.seqs[j]
                self.alseq.needleman_Wunsch(s1, s2)
                alin = self.alseq.recover_align()
```

Main Class

- run метод нь HierarchicalClustering классын объектыг үүсгэж, кластерын алгоритмыг гүйцэтгэж, үүссэн модыг буцаана.

```
                ncd = 0
                for k in range(len(alin)):
                    col = alin.column(k)
                    if (col[0] != col[1]): ncd += 1
                self.matdist.set_value(i, j, ncd)

    def run(self):
        ch = HierarchicalClustering(self.matdist)
        t = ch.execute_clustering()
        return t

def test():
    seq1 = MySeq("ATAGCGAT")
    seq2 = MySeq("ATAGGCCT")
    seq3 = MySeq("CTAGGCCC")
    seq4 = MySeq("CTAGGCCT")
    sm = SubstMatrix()
    sm.create_submat(1, -1, "ACGT")
    alseq = PairwiseAlignment(sm, -2)
    up  = UPGMA([seq1, seq2, seq3, seq4], alseq)
    arv = up.run()
    arv.print_tree()
```

BioPython Functions for Phylogenetic Analysis

- Bio.Phylo модуль.
- Ерөнхийдөө филогенетик модыг төлөөлөх өгөгдлийн бүтцийг хэрэгжүүлдэг бөгөөд модыг ачаалах, хадгалах, тойрох методуудтай.
  - **read**, **parse** функцийг файлаас нэг буюу хэд хэдэн модыг унших
  - **write** функцийг файл руу мод бичих,
  - **convert** функцийг файлын форматыг шууд хөрвүүлэх.

Newick форматтай модыг "simple.dnd" файлд хадгалсан байг

$$(((A,B),(C,D)),(E,F,G));$$

```
tree = Phylo.read("simple.dnd", "newick")
print(tree)
Phylo.draw_ascii(tree)
```

- эхнийх нь модны агуулгыг хэвлэх,
- хоёр дахь нь модны энгийн график дүрслэл.

```
tree2 = Phylo.read("int_node_labels.nwk", "newick")
Phylo.draw_ascii(tree2)

Phylo.convert("int_node_labels.nwk", "newick", "tree.xml", "phyloxml"
    )
trees = Phylo.parse("tree.xml", "phyloxml")
for t in trees: print(t)
```

BioPython Functions for Phylogenetic Analysis

- Энэ модуль нь дараагийн жишээнд үзүүлсэн шиг модны мөчрүүдийг өөр өөр өнгөөр будаж болно.
- E, F, G навч, тэдгээрийн нийтлэг өвөг хулд загас, C, D навчтай мөчрийг цэнхэр өнгөөр будна.

```
from Bio.Phylo.PhyloXML import Phylogeny

treep = Phylogeny.from_tree(tree)
Phylo.draw(treep)

treep.root.color = "gray"
mrca = treep.common_ancestor({"name": "E"}, {"name": "F"})
mrca.color = "salmon"
treep.clade[0, 1].color = "blue"
Phylo.draw(treep)
```

АНХААРАЛ ТАВЬСАНД БАЯРЛАЛАА