ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
Мэдээлэл, Холбооны Технологийн Сургууль

F.CS213 Биоалгоритм

Multiple Sequence Alignment

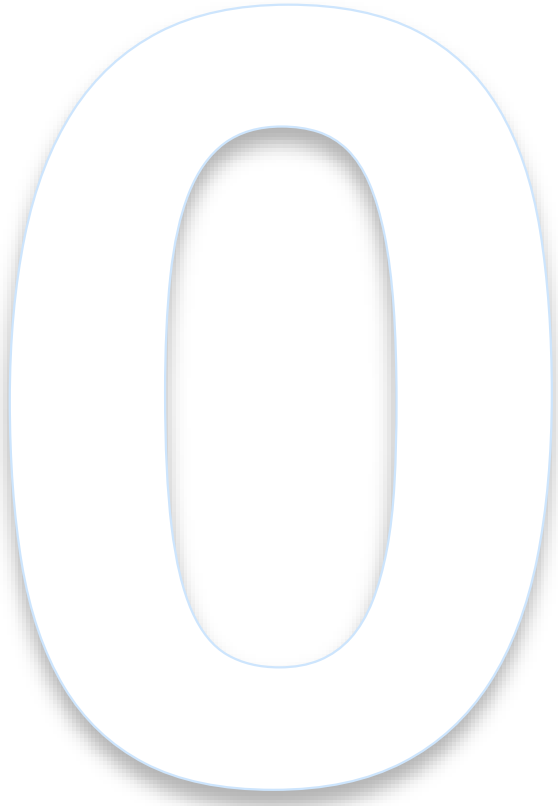
Олон дараалал дээрх зэрэгцүүлэлт

Лекц 7

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

Лекцийн агуулга

- Удиртгал:
  - Асуудал
  - Тооцооллын хүндрэл
- Олон дараалал зэрэгцүүлэлт ба DP
- Хьюристик алгоритмууд
- Прогрессив зэрэгцүүлэлтийг Python-аар хэрэгжүүлэх нь
- BioPython ашиглан зэрэгцүүлэлтийг хэрэгжүүлэх



Удиртгал

- Олон дарааллийг харьцуулах нь уургийн үүргийг тодорхойлоход чухал ач холбогдолтой :
  - Ороолтын дараалалтай гомолог байх боломжтой олон организмын дарааллуудаас бүрдэх бүлгээс уургуудын тухайн үүргийг илрүүлэх,
  - Үүргийн тайлбарт зориулсан таамаглал бий болгоход итгэл үнэмшлийг нэмэгдүүлдэг;
  - Тухайн үүргийг эдгээр уургууд дээр агуулж байж болох хэсгүүдийг илүү үндэслэлтэйгээр тодорхойлох
- Уургийн зэрэгцүүлэлт нь тэдний хоёрдогч/гуравдагч бүтэц (хувирал)-ийг илрүүлэхэд тохиромжтой.
- Эпидемиолог (өвчний тархалтын судалгаа)-ийн хувьд организм, төрөл зүйлийн ДНХ-ийн дарааллуудыг зэрэгцүүлэн авч үзсэнээр тэдний хувьсал, хоруу чанарт өртөх хэсгүүдийг ойлгоход хамгийн чухал.
- *Multiple Sequence Alignment (MSA)* асуудал нь Хоёр дараалал дээрх зэрэгцүүлэлтийг $N(N > 2)$ дараалал дээр буюу ерөнхий тохиолдолд нь авч үздэг.

УДИРТГАЛ Тооцооллын хүндрэл

MSA асуудал нь тооцооллын хүндрэлийн хувьд NP-hard ангид хамаардаг.
N нь их байхад асуудлыг шийдэх үр дүнтэй алгоритм байхгүй.

- Хоёр дараалал зэрэгцүүлэлттэй бараг адилхан ч дарааллын тоо өсөхөд хүндрэл нь нэмэгддэг.
- Хоёр дараалал зэрэгцүүлэлт
  - Эцсийн оноонд хүрэхийн тулд багануудын оноог нэмж цуглуулдаг (Багана бүр 2 ширхэг тэмдэгт).
  - Орлуулах матриц болон зай-торгууль ашигладаг
  - MSA удирдахад дээрхийг баримтлана
- MSA-ийн хувьд баганууд хоёроос илүү тэмдэгт агуулна.
  - Ерөнхийдөө *sum of pairs (SP)* аргыг хэрэглэнэ.
  - Багана бүрийн хувьд бүх боломжит хос тэмдэгтүүдийг авч үзэн оноонуудыг нэгтгэнэ.
  - Хэтэрхий олон зайн багана үүсэхээс сэргийлэн зайн торгуулийг тохируулж болно.

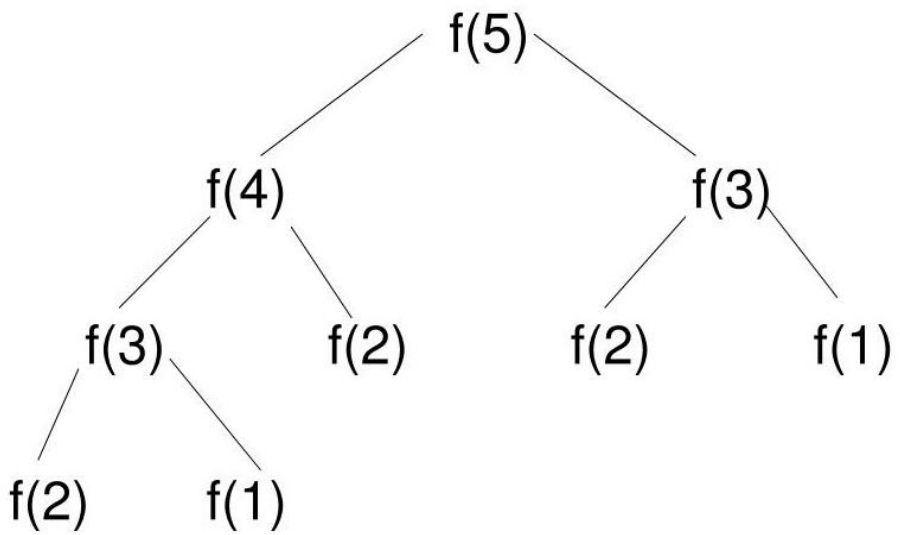
```
-RADNS
ARCD-A
AR-D-A
```

$$(sm(A,A)+g)+3\times sm(R,R)+(sm(A,C)-g)+3\times sm(D,D)+(2\times g)+(2\times sm(S,A)+sm(A,A)$$

$$Score=(4-8)+3\times 5+(0-8)+3\times 6+(-8\times 2)+(1+1+4)=11$$

MSA ба динамик программчлал

Хоёр дараалал дээр зэрэгцүүлэлт хийх Динамик програмчлал (DP)-ын үр ашигтай алгоритмууд нь дарааллын тоо N байх ерөнхийд тооцоололд ашиглах боломжтой юу?

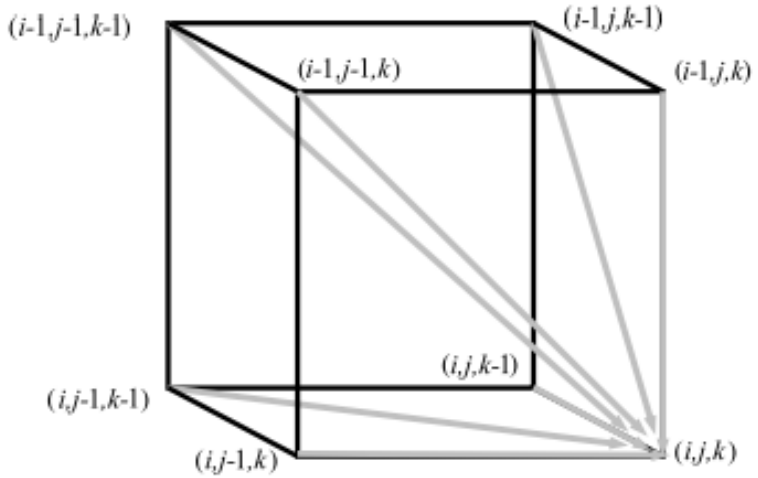


- DP аргуудаар MSA тооцоолж болох ч дарааллын тоо нэмэгдэхэд тооцооллын үр ашигтай байдаггүй.
- Тооцооллын хүндрэлийн хувьд, хоёр дараалал зэрэгцүүлэлт нь харьцуулах дарааллуудын дундаж урттай харьцангуй квадрат хүндрэлэй DP алгоритмтай байдаг.
- MSA-д зориулсан ерөнхийх тооцоололд хүндрэл нь *экспоненциал (exponential)* болж, N нь илтгэгч болдог
  - Биологийн судалгаан дахь дундажаас харьцангуй цөөн тооны дараалал дээр л ашиглах боломжтой.

Рекуррент харьцаа

Хос дараалал зэрэгцүүлэлт нь 2D матрицууд (оноо болон trace-back)-ыг байгуулдаг бол MSA-ын оновчтой шийдэд N хэмжээст бүтцүүд (hypercubes) шаардлагатай.

- $N = 3$ байхад DP алгоритмын S ба T матрицууд нь 3 хэмжээстэй байна.
- *Needleman-Wunsch* алгоритмын зорилгын функцийн рекуррент харьцааны дагуу нийлбэрийг тооцоолно.
  - Дараалал нь $A = a_1 a_2 \dots a_n, B = b_1 b_2 \dots b_m$ ба $C = c_1 c_2 \dots c_p$
  - Зайн торгууль нь g (багана дахь зай тус бүрт торгууль нь тогтмол).
- DP-ын рекуррегт харьцааны жишээ :

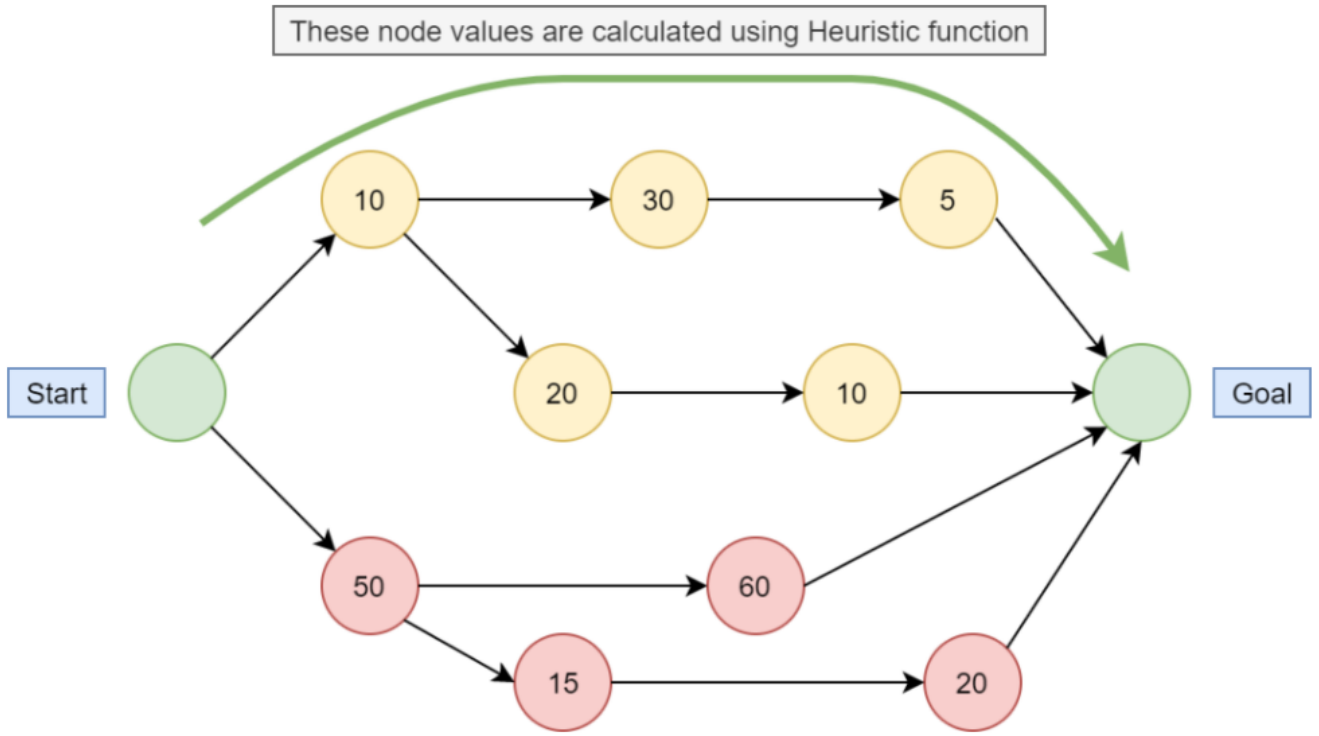


$$
\begin{aligned}
S_{i,j,k} = \max(& \\
& S_{i-1,j-1,k-1} + sm(a_i, b_j) + sm(a_i, c_k) + sm(b_j, c_k), \\
& S_{i-1,j-1,k} + sm(a_i, b_j) + g, \\
& S_{i-1,j,k-1} + sm(a_i, c_k) + g, \\
& S_{i,j-1,k-1} + sm(b_j, c_k) + g, \\
& S_{i-1,j,k} + 2g, \\
& S_{i,j-1,k} + 2g, \\
& S_{i,j,k-1} + 2g \\
), & \forall 0 < i \leq n, 0 < j \leq m, 0 < k \leq p
\end{aligned}
$$

Тооцооллын өртөг

- N хэмжээт гиперкуб дахь боломжит бүх замыг туршихаас зайлсхийсэн зарим оролдлого хийсэн.
  - Энд дарааллын хослол бүр дээрх тухайн хос зэрэгцүүлэлт нь *оновчтой (optimal) MSA* гэж үздэг.
  - Энэ нь хос зэрэгцүүлэлтийн матрицыг 2 хэмжээст орон зайд буулгасан зам байна.
- Дарааллын тухайн хослол бүрийн хувьд дээрх буулгалтын өртгийн дээд хязгаарыг тооцоолох боломжтой.
  - Энэ нь хос зэрэгцүүлэлтэнд авч үзэх боломжит замуудын олонлогыг хязгаарлана,
  - Үүнийг MSA-н хайлтанд шинжлэх ёстой таамаглалын олонлогийг хязгаарлахад ашиглаж болно.
- Гэсэн хэдий ч дарааллын тоо 10-аас их буюу практикт ашиглахад тохиромжгүй.

Хьюристик алгоритмууд



- DP алгоритмыг орлох нэг сонголт,
- Ерөнхийдөө практик агуулгаар илүү олон тооны дарааллын хувьд MSA-г өргөтгөдөг
- Энэ үед хьюристик (мөн ойролцоо гэж нэрлэдэг) алгоритмуудыг ашигладаг.
- Оновчтой шийдлийг баталгаатай өгч чадахгүй ч ажиллах хугацааны хувьд хүлээн зөвшөөрөгдөхүйц тооцооллын хүндрэлтэй байдаг.

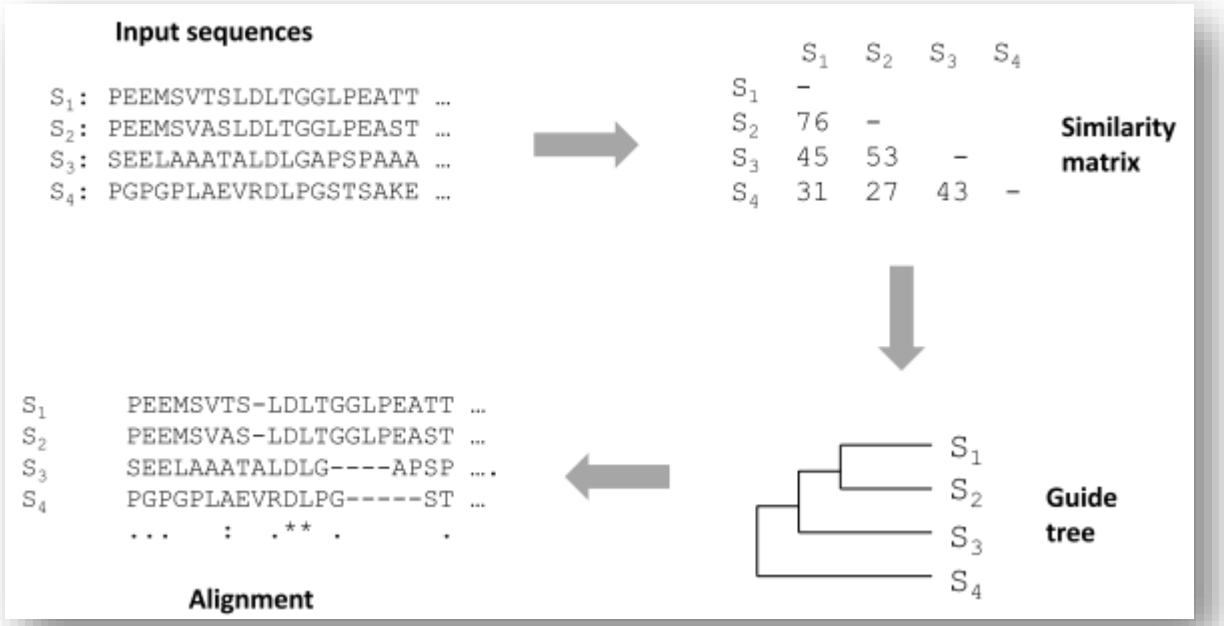
- MSA-д зориулсан хьюристик алгоритмуудыг ерөнхийд нь дараах байдлаар ангилж болно.
  - **Прогрессив (Progressive)** – эхлээд хоёр дарааллыг харьцуулж, дараа нь үлдсэн дарааллуудыг давтан нэмж харьцуулна;
  - **Давталттай (Iterative)** – зайг зөөх, нэмэх, устгах байдлаар эхний зэрэгцүүлэлтийг сайжруулахыг оролдоно;
  - **Гибрид (Hybrid)** – стратегиудыг хослуулах, гүйцээлт (complementary) мэдээлэл (уургийн бүтцийн мэдээлэл, жишиг локал зэрэгцүүлэлтийн сан г.м) хэрэглэх боломжтой.

Прогрессив зэрэгцүүлэлт

Прогрессив алгоритмын ерөнхий санаа нь хамгийн ойрхон хоёр дараалл дээр эхлэл зэрэгцүүлэлтийг үүсгэж, дараагийн давталтууд дээр арай хол дарааллыг нэмсээр бүх дарааллыг хамарсан зэрэгцүүлэлтийг гарган авна.

Сонгодог MSA аргачлал:

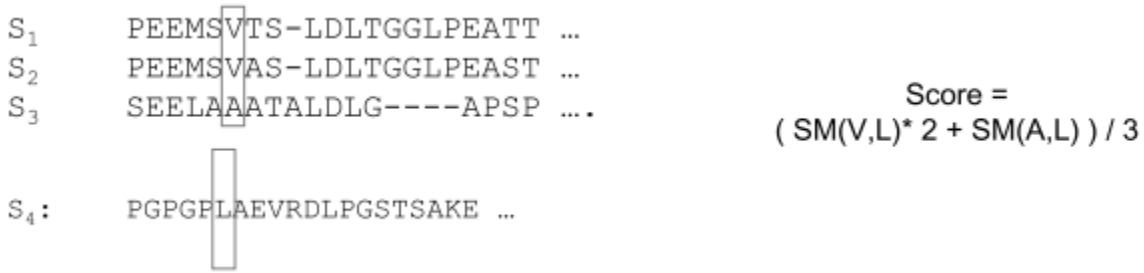
CLUSTAL -> CLUSTALW -> Clustal Omega



Overall workflow of the CLUSTAL, representing progressive MSA algorithms.

- Хос дараалал зэрэгцүүлэлтийг хоёр дарааллын бүрийн хувьд тооцоолж *ижилсэлтийн матриц (similarity matrix)*-ыг үүсгэнэ.
- Тус матриц нь *Чиглүүлэгч мод (guide tree)* үүсгэх алгоритмын оролт болно.
- Тухайн зэрэгцүүлэлтийн суурь болох учир эхний хоёр дарааллыг сонгох нь хамгийн чухал.
- Нэг чухал алхам бол зэрэгцүүлэлт дээр дараагийн дарааллыг нэмж шинэ дараалалыг тогтворжсон хэлбэр (consensus)-р үүсгэх :
  - Багана бүрийг хамгийн нийтлэг тэмдэгтээр илэрхийлж болно. (хэт явцуу)
  - Ерөнхийдөө тэмдэгтүүдийн давтамжийг авч үздэг (профайл).
- Давталт бүрт бид дарааллыг профайл хэлбэртэй өмнөх зэрэгцүүлэлттэй жишнэ.

Жигнэсэн оноо



Зэрэгцүүлэлтийг дараалалтай хослуулах үед баганын оноог тооцоолох.
SM нь ашигласан орлуулах матриц.

- Илрэл (Match)-ийн оноог тодорхойлох (DP алгоритм дахь S/T матрицын диагональ шилжилт) -ын тулд профайл дээр илрэх боломжтой тэмдэгтүүдийг давтамжаар нь жигнэж, нэмэх гэж буй дарааллын тэмдэгттэй нэгтгэнэ.
  - Тиймээс оноо нь боломжит хосуудын онооны жигнэсэн дундаж болно.
- Зэрэгцүүлэлт дээр шинэ дараалал нэмэх алхамд “once a gap, always a gap” хандлагатай.
  - Зэрэгцүүлэлт дээр гарч ирсэн зай дараагийн алхамд хадгалагдана
- Илүү ерөнхий тохиолдолд шинэ зэрэгцүүлэлт байгуулахын тулд чиглүүлэгч модны мөчрүүдэд харгалзах хоёр профайлын зэрэгцүүлэлтийг нэгтгэх хэрэгтэй байж болно.
  - Уг процессыг дээр дурьдсан аргын ерөнхий ойлголт гэж үзэхэд хялбар байдаг.

Бусад алгоритмууд

- CLUSTAL ангиллын CLUSTALW алгоритм нь олон жилийн турш нэлээд алдартай байсан.
  - ***Параметеруудын уялдаа:*** Орлуулах матриц болон зай-торгуулийг зэрэгцүүлэлтийн үе шат болон тухайн бүсийн дараалалд байгаа *үлдэгдэл (residue)*-үүдэд нийцүүлсэн: Жнь: гидрофилийн үлдэгдлийн хэсэг нь ихэвчлэн гогцоо эсвэл санамсаргүй ороомог бүтцийн бүсийг заадаг ба энэ нь эдгээр бүсүүдэд зай-нээх торгуулийг бууруулах чиглэл болдог.
  - ***Орлуулах матрицууд:*** Зэрэгцүүлэлт рүү нэмж буй дарааллын магадлалт зөрүүнээс хамаарч зэрэгцүүлэлтийн процессийн дагуу ялгаатай байна. Мөн профайлуудын зэрэгцүүлэлтэнд оноог тооцохдоо эдгээр багц бусадтайгаа илүү ижил төстэй бол бага жинтэй, эсрэг тохиолдолд их жинтэй байхаар авч үзнэ.

- Сүүлийн үед CLUSTAL ангиллын CLUSTAL Омега алгоритм хэрэглээнд шилжсэн.
  - Зэрэгцүүлэлт дэх дарааллуудын эрэмбэ тодорхойлдог чиглүүлэгч модыг үүсгэх алгоритмыг илүү сайжруулсан.
  - Хэдийгээр практикт өргөн хэрэглэгддэг боловч прогрессив алгоритмуудын хьюристик шунахай (greedy) хандлагатай нь холбоотойгоор зарим асуудал үүсдэг.
    - Зэрэгцүүлэлтийн эхний үе шатанд гаргасан буруу шийдвэр залруулагдахгүй.
    - Хамгийн муу үр дүн нь харьцуулах дарааллуудын ихэнх нь similarity багатай үед үүснэ.

- T-coffee алгоритм нь тогворжилт (consistency)-д суурилсан аргачлалруу чиглэдэг
  - Хос зэрэгцүүлэлтийн agreement-ийг ихэсгэж, улмаар прогрессив алгоритмд алдаа үүсгэхгүй байхыг хичээнэ.
  - T-coffee сангууд нь локал ба глобал зэрэгцүүлэлтийн хувьд хос зэрэгцүүлэлтийн бүх байрлалд жинг тооцдог.
  - Дараа нь эдгээр жинг зэрэгцүүлэлт рүү шинэ дараалал нэмэх үед DP алгоритмын оноо болгон ашигладаг.
  - Энэ арга нь ихэвчлэн илүү нарийвчлалтай MSA шийдлүүдийг өгдөг боловч ClustalOmega зэрэг алгоритмтай харьцуулахад дарааллын тоо их байхад тохиромжгүй, тооцооллын үр ашгийг бууруулдаг.

Судлагдсан байдал

- Давталттай алгоритмууд нь өөр хувилбар байж болно.
  - Прогрессив зэрэг өгөгдсөн аргыг ашиглан зэрэгцүүлэлт үүсгэн эхэлдэг.
  - Дараа нь зарим өөрчлөлтүүдийг хийж, улмаар зорилгын функцэд хэрхэн нөлөөлж буйг үнэлэх замаар зэрэгцүүлэлтийг сайжруулахыг зорьдог.
  - Зэрэгцүүлэлтийг сайжруулах нэг хувилбар бол зайн байрлалыг өөрчлөх эсвэл зай нэмэх/ арилгах замаар дарааллын дэд бүлгүүд эсвэл баганын дэд бүлгүүдийг дахин дахин зэрэгцүүлэх явдал юм.
  - Генетикийн алгоритм зэрэг илүү хөгжсөн оновчлолын мета-хьюристикийг бас туршиж үзсэн бөгөөд зарим амжилтад хүрсэн.

- Гибрид алгоритмууд нь одоогоор хамгийн үр дүнтэй аргуудын нэг, Жнь: MUSCLE
  - Профайл дээр суурилсан прогрессив зэрэгцүүлэлтийг давталтын аргын техниктэй хослуулан хэрэглэдэг
  - Үр дүнгийн зэрэгцүүлэлтийг сайжруулахын тулд чиглүүлэгч модыг хөгжүүлдэг.

- *MAFFT* алгоритм нь
  - Амин хүчлийн тэмдэгтэн дарааллыг *хэмжээ ба далайц (volumes and polarities)-ын* цуваа болгон хөрвүүлэх ***Хурдан Фурье хувиргалт (Fast Fourier Transforms - FFT)***-ийн хэрэглээг MSA алгоритмуудад бий болгодог.
  - Прогрессив болон давталттай аргуудыг хослуулдаг
  - FFT нь ижил төстэй бүсийг хурдан илрүүлэх боломжийг олгодог

Прогрессив зэрэгцүүлэлт (ПЗ)-ийг хэрэгжүүлэх нь

- Объект хандалгат программчлал
- Үндсэн классууд.
  - Зэрэгцүүлэлтийн илэрхийлэх - **MyAlign**
  - Хос зэрэгцүүлэлт - **PairwiseAlignment**
  - Олон дараалал зэрэгцүүлэлт - **MultipleAlign**
- **MySeq** - Биологийн дараалал (ДНХ, РНХ эсвэл уураг) зориулсан. Өмнө нь үзсэн

ПЗ-ИЙГ ХЭРЭГЖҮҮЛЭХ НЬ

Зэрэгцүүлэлтийн илэрхийлэх - MyAlign

```
class MyAlign:

    def __init__(self, lseqs, al_type = "protein"):
        self.listseqs = lseqs
        self.al_type = al_type

    def __len__(self): # number of columns
        return len(self.listseqs[0])

    def __getitem__(self, n):
        if type(n) is tuple and len(n) ==2:
            i, j = n
            return self.listseqs[i][j]
        elif type(n) is int: return self.listseqs[n]
        return None

    def __str__(self):
        res = ""
        for seq in self.listseqs:
            res += "\n" + seq
        return res

    def num_seqs(self):
        return len(self.listseqs)

    def column (self, indice):
        res = []
        for k in range(len(self.listseqs)):
            res.append(self.listseqs[k][indice])
        return res

if __name__ == "__main__":
    alig = MyAlign(["ATGA-A","AA-AT-"], "dna")
    print(alig)
    print(len(alig))
    print(alig.column(2))
    print(alig[1,1])
    print(alig[0])
```

- al_type - төрөл (ДНХ, РНХ, уураг)
- listseqs - дарааллын жагсаалт
  - string, зайг "-"

Зэрэгцүүлэлтийн тогторжсон хувилбар

```
class MyAlign:
                                (...)
    def consensus (self):
        cons = ""
        for i in range(len(self)):
            cont = {}
            for k in range(len(self.listseqs)):
                c = self.listseqs[k][i]
                if c in cont:
                    cont[c] = cont[c] + 1
                else:
                    cont[c] = 1
            maximum = 0
            cmax = None
            for ke in cont.keys():
                if ke != "-" and cont[ke] > maximum:
                    maximum = cont[ke]
                    cmax = ke
            cons = cons + cmax
        return cons

if __name__ == "__main__":
    alig = MyAlign(["ATGA-A","AA-AT-"], "dna")
    print(alig.consensus())
```

- Бас нэг чухал метод бол Зэрэгцүүлэлтийн тогтворжон хэлбэрийг үүсгэх тооцоолол
- Тогтворжсон хэлбэрийг зэрэгцүүлэлтийн багана бүрийн хувьд зайг тооцоогүй хамгийн өндөр давтамжтай тэмдэгтүүдийн дараалалаар илэрхийлнэ.
- Энэ метод нь dictionary ашиглан багана бүрийн тэмдэгтүүдийн давтамжийг тоолж, хамгийн олныг сонгоно.

ПЗ-ИЙГ ХЭРЭГЖҮҮЛЭХ НЬ

Хос зэрэгцүүлэлт - PairwiseAlignment

```
from MyAlign import MyAlign
from MySeq import MySeq
from SubstMatrix import SubstMatrix

class PairwiseAlignment:

    def __init__(self, sm, g):
        self.g = g
        self.sm = sm
        self.S = None
        self.T = None
        self.seq1 = None
        self.seq2 = None

    def score_pos (self, c1, c2):
        (...)

    def score_alin (self, alin):
        (...)
```

- орлуулах матриц ба зай-торгууль
- зэрэгцүүлэх дарааллууд,
- DP алгоритмын S ба T матрицууд.

```
    def needleman_Wunsch (self, seq1, seq2):
        if (seq1.seq_type != seq2.seq_type): return None
        (...)
        return self.S[len(seq1)][len(seq2)]

    def recover_align (self):
        (...)
        return MyAlign(res, self.seq1.seq_type)

    def smith_Waterman (self, seq1, seq2):
        if (seq1.seq_type != seq2.seq_type): return None
        (...)
        return maxscore

    def recover_align_local (self):
        (...)
        return MyAlign(res, self.seq1.seq_type)
```

SubstMatrix – Орлуулах матриц

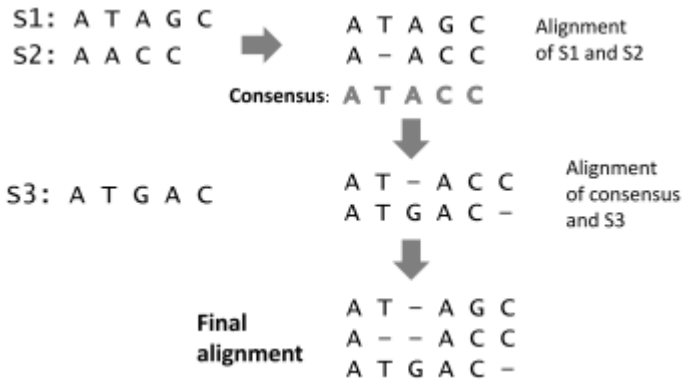
```
class SubstMatrix:

    def __init__(self):
        self.alphabet = ""
        self.sm = {}

    def __getitem__(self, ij):
        i, j = ij
        return self.score_pair(i, j)

    def score_pair(self, c1, c2):
        if c1 not in self.alphabet or c2 not in self.alphabet:
            return None
        return self.sm[c1+c2]

    def read_submat_file(self, filename, sep):
        (...)

    def create_submat(self, match, mismatch, alphabet):
        (...)
```

- Dictionary бүтцэд суурилан дүрсэлсэн орлуулах матрицыг ашиглана.
- цагаан толгойг хадгалдаг аттрибут
- хос тэмдэгтүүдийн оноотой dictionary
- Өмнө нь үзсэн кодтой ижил төстэй гүйцээж бичнэч

ПЗ-ИЙГ ХЭРЭГЖҮҮЛЭХ НЬ Олон дараалал зэрэгцүүлэлт

Оролт: Өгөгдсөн дарааллын багц, Зэрэгцүүлэлтийн зорилгын функцийн параметрүүд (орлуулах матриц ба зай-торгууль)
Гаралт: Дарааллын хамгийн боломжит зэрэгцүүлэлт.

- Эхлээд эхний хоёр дараалал дээр *Needleman-Wunsch* алгоритмыг ашиглан зэрэгцүүлэлт байгуулна.
- Дараа нь давталт бүрт дараагийн дарааллыг зэрэгцүүлэлтрүү нэмнэ. Энд тухайн зохицуулалтыг авч, тогтворжсон хэлбэрийг тооцдог.
- Улмаар уг тогворжсон хэлбэр рүү *Needleman-Wunsch* алгоритмыг ашиглан шинэ дараалал зэрэгцүүлнэ.
- Үүссэн зэрэгцүүлэлтийг тогтвожсон хэлбэрийн баганууд дээр үндэслэн дахин байгуулна. Ингэхдээ хэрэв зэрэгцүүлэлт нь тухайн байрлалд зай байвал уг баганад зай авна.



Scores: match 1, mismatch −1, gap penalty −1

```
def test():
    s1 = MySeq("ATAGC")
    s2 = MySeq("AACC")
    s3 = MySeq("ATGAC")
    sm = SubstMatrix()
    sm.create_submat(1,-1,"ACGT")
    aseq = PairwiseAlignment(sm,-1)
    ma = MultipleAlignment([s1,s2,s3], aseq)
    al = ma.align_consensus()
    print(al)

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

MultipleAlign

```
from PairwiseAlignment import PairwiseAlignment
from MyAlign import MyAlign
from MySeq import MySeq
from SubstMatrix import SubstMatrix

class MultipleAlignment():

    def __init__(self, seqs, alignseq):
        self.seqs = seqs
        self.alignpars = alignseq

    def add_seq_alignment (self, alignment, seq):
        res = []
        for i in range(len(alignment.listseqs)+1):
            res.append("")
        cons = MySeq(alignment.consensus(),alignment.al_type)
        self.alignpars.needleman_Wunsch(cons, seq)
        align2 = self.alignpars.recover_align()
        orig = 0
        for i in range(len(align2)):
            if align2[0,i]== '—':
                for k in range(len(alignment.listseqs)):
                    res[k] += "—"
            else:
                for k in range(len(alignment.listseqs)):
                    res[k] += alignment[k,orig]
                orig+=1
        res[len(alignment.listseqs)] = align2.listseqs[1]
        return MyAlign(res, alignment.al_type)

    def align_consensus(self):
        self.alignpars.needleman_Wunsch(self.seqs[0], self.seqs[1])
        res = self.alignpars.recover_align()

        for i in range(2, len(self.seqs)):
            res = self.add_seq_alignment(res, self.seqs[i])
        return res
```

- **seqs** – зэрэгцүүлсэн дарааллууд
- **alignpars** – Needleman-Wunsch алгоритмд хэрэглэгдэх зэрэгцүүлэлтийн параметерүүд

BioPython зэрэгцүүлэлт

MultipleSeqAlignment: Хос эсвэл олон дараалал зэрэгцүүлэлтийг үндсэн класс
(Хос зэрэгцүүлэлт нь зөвхөн тухайн тохиолдол болно).

```
from Bio import Alphabet
from Bio.SeqRecord import SeqRecord
from Bio.Align import MultipleSeqAlignment
from Bio.Alphabet import IUPAC
from Bio.Seq import Seq

seq1 = "MHQAIFIYQIGYPLKSGYIQSIRSPEYDNW"
seq2 = "MH--IFIYQIGYALKSGYIQSIRSPEY-NW"
seq3 = "MHQAIFI-QIGYALKSGY-QSIRSPEYDNW"

seqr1 = SeqRecord(Seq(seq1,Alphabet.Gapped(IUPAC.protein)),id="seq1")
seqr2 = SeqRecord(Seq(seq2,Alphabet.Gapped(IUPAC.protein)),id="seq2")
seqr3 = SeqRecord(Seq(seq3,Alphabet.Gapped(IUPAC.protein)),id="seq3")

alin = MultipleSeqAlignment([seqr1, seqr2, seqr3])
print(alin)
```

```
print(alin[1]) # 2nd sequence
print(alin[:,2]) # 3rd column
print(alin[:,3:7])  # 4th to 7th columns (all sequences)
print(alin[0].seq[:3]) # first 3 columns of seq1
print(alin[1:3,5:12]) # sequences 2 and 3; 4th to 10th column
```

Оролт/гаралт

- AlignIO: Мөн дараалалтай адил янз бүрийн форматтай зэрэгцүүлэлтийн оролт/гаралтын үйлдлийн анги.
- Зэрэгцүүлэлтийг нэг нэгээр нь read, олноор нь parse методоор уншина.
- Мөн write методыг өөр өөр форматын хооронд шууд хөрвүүлэлт хийхэд ашиглаж болно.

```
from Bio import AlignIO
alin2 = AlignIO.read("PF05371_seed.aln", "clustal")

print("Size:", alin2.get_alignment_length())
for record in alin2:
    print (record.seq, record.id)
```

```
AlignIO.write(alin2, "example_alin.fasta", "fasta")

AlignIO.convert("PF05371_seed.aln", "clustal", "example_alin.fasta",
   "fasta")
```

АНХААРАЛ ТАВЬСАНД БАЯРЛАЛАА