Поиск и анализ последовательностей

Алексей Сальников

Факультет вычислительной математики и кибернетики
МГУ имени М. В. Ломоносова

13 апреля 2011 г.

Поиск последовательностей

Основные вопросы можно сформулировать следующим образом.

- Что это?
- Где такое можно встретить ещё?

```
1   cccacttttc ccaaacaaag ctcccggcaa ctttctccct cgcagcgccc cgcccgcccg
61  cggctcccca gccccaggcc gggaggccca gatgggggaa ccccgggctg gggccgccct
121 ggacgatggc agcggctgga cgggcagtga ggaaggcagt gaggagggta ccggcggcag
181 tgagggggct gggggtgacg ggggcccgga tgcagagggg gtgtggagcc cagacattga
241 gcagagcttc caggaggccc tggccatcta tccaccctgc ggccgccgga aaataatttt
301 gtctgatgaa ggcaagatgt atggtcggaa tgaactgatc gcccgctaca tcaagctgag
361 aacggggaag acccgaactc gaaaacaggt ttctagtcac atccaggttt tggcccgaag
```

Поиск последовательности в банках данных

Ответ на вопросы в предыдущем слайде: **организовать поиск в банках данных**.

Алгоритм BLAST

- Используется для поиска последовательностей похожих на запрашиваемую в банках данных (GenBank, UniProt).
- Ищутся последовательности похожие относительно оценочной функции точностью до некоторого порога.

Реализации алгоритма BLAST

- NCBI-BLAST – доступен как online сервис на сайте NCBI, Можно скачать себе с сайта `http://www.ncbi.nlm.nih.gov/BLAST/download.shtml`
- WU-BLAST – не поддерживается авторами но есть в работающем виде без исходников на многих сайтах
- mpiBLAST – MPI реализация для кластеров.
- ParallelBLAST – реализация для BLueGene/L.

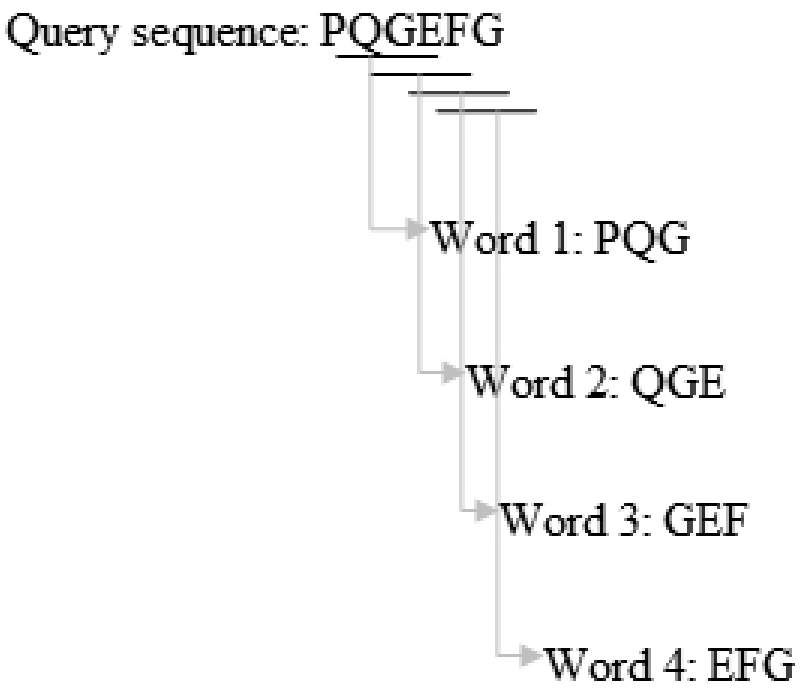
Бласт как набор программ

- blastn – искомая строка представлена нуклеотидами, поиск происходит только внутри нуклеотидной базы данных.
- blastp – поиск среди белковых последовательностей.
- blastx – строка транслируется в белковую
- tblastn – ищется в нуклеотидной базе данных, по транслированной в нуклеотидную белковой цепочки.
- tblastx – тоже самое, что tblastn но для белков.
- bl2seq – сравнение 2-х строк.

Стадии алгоритма BLAST

1. Создание списка слов длинны k для строки запроса. Очычно 3 символа для белков и 11 – для нуклеотидов
2. Отбор наиболее представительных слов среди всех полученных
3. Поиск слов банке данных в порядке убывания значимости
4. Расширение последовательностей в обе стороны, до тех пор, пока степень совпадения не будет меньше порога, или одна из последовательностей не исчерпается.
5. Оценка правдоподобности найденых сопоставлений.

Создание слов длины k



Слова выбираются с шагом 1 со всеми возможными перекрытиями.

Отбор наиболее значимых слов

Значимость слова определяется следующим образом:

$$score(a) = \sum_{j=1}^{N} \sum_{i=1}^{k} substitute(w_{j,i}, a_i) \quad (1)$$

Здесь w_j – все возможные слова длинны k; a - слово, для которого определяется значимость; substitute - матрица замен одного символа на другой.

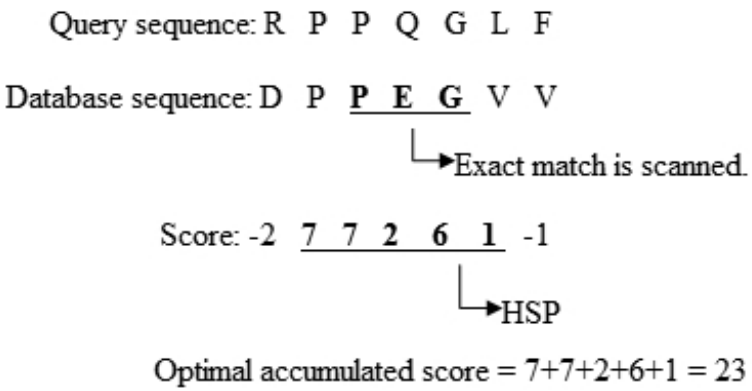
Расчёт матрицы замен

Функция подстановок может быть вычислена следующим образом: $substitute(a, b) = \ln(\frac{P_{a,b}}{P_a P_b})$, где a и b символы алфавита A.
Пусть P_a — вероятность встретить аминокислотный остаток a в соответствующей последовательности, $P_{a,b}$ — вероятность замены a на b в процессе эволюции.

Матрица замен BLOSUM50

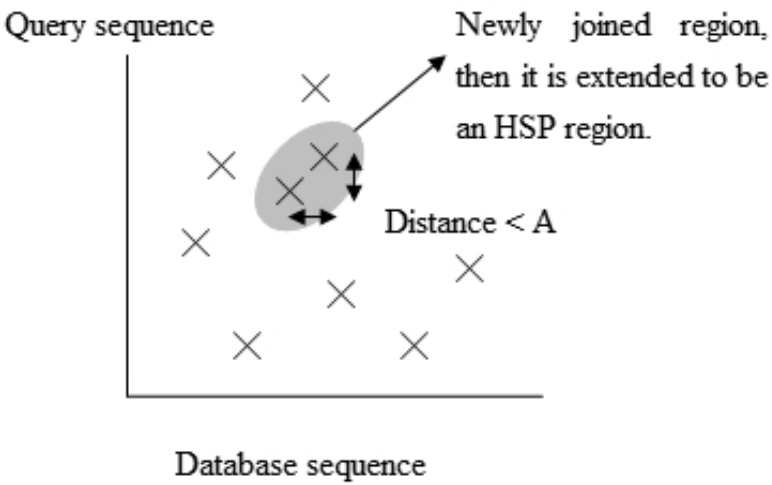
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-4	15	2	-3
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5

Поиск слов в банке и удлиннение



- Находится совпадение.
- слева и справа добавляют по символу, пока значение качества вычисляемое по формуле (1) не начнёт уменьшаться.

слияние удлиннений



Если процесс удленнения заходит на территорию соседей, то эти удлинения сливаются.

Оценка правдоподобности сопоставления

Оценка правдоподобия вычисляется по следующей формуле:

$$P(x \leq S) = 1 - e^{-E}$$

Где x - значение качества сопоставления, S - порог.
E вычисляется следующим образом:

$$E = Kmne^{-\lambda S}$$

Где m - средняя длина последовательности в базе, n - длина запрашиваемой последовательности, K, λ - задаются алгоритмом исходя из матрицы подстановок.

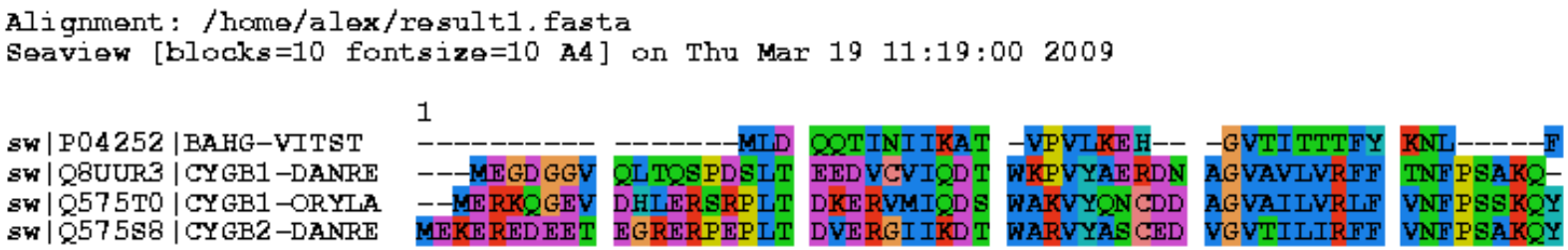
Формат файлов: последовательности белков в FASTA формате

```
>sw|P04252|BAHG-VITST Bacterial hemoglobin.
MLDQQTINIIKATVPVLKEHGVTITTTFYKNLF

>sw|Q8UUR3|CYGB1-DANRE Cytoglobin-1.
MEGDGGVQLTQSPDSLTEEDVCVIQDTWKPVYAERDNA
GVAVLVRFFTNFPSAKQ

>sw|Q575T0|CYGB1-ORYLA Cytoglobin-1.
MERKQGEVDHLERSRPLTDKERVMIQDSWAKVYQNCDD
AGVAILVRLFVNFPSSKQY

>sw|Q575S8|CYGB2-DANRE Cytoglobin-2.
MEKEREDEETEGRERPEPLTDVERGIIKDTWARVYASC
EDVGVTILIRFFVNFPSAKQY
```

Множественное выравнивание

```
----------------MLDQQTINIIKAT-VPVLKEH---GVTITTTFYKNL------F
---MEGDGGVQLTQSPDSLTEEDVCVIQDTWKPVYAERDNAGVAVLVRFFTNFPSAKQY
--MERKQGEVDHLERSRPLTDKERVMIQDSWAKVYQNCDDAGVAILVRLFVNFPSSKQY
MEKEREDEETEGRERPEPLTDVERGIIKDTWARVYASCEDVGVTILIRFFVNFPSAKQY
```

Множественное выравнивание (программа seaview)



Назначение множественных выравниваний

Исследование структуры аминокислотных и нуклеотидных последовательностей чрезвычайно важно для молекулярной биологии и биоинженерии:

- систематизация ДНК, РНК и белковых фрагментов;
- поиск участков схожести, которые могут соответствовать функциональным, структурным или эволюционным отношениям между данными фрагментами.

Определение парного выравнивания

1. Введём две последовательности символов
   - $S_1 = s_{1,1}s_{1,2}...s_{1,L_1}$
   - $S_2 = s_{2,1}s_{2,2}...s_{2,L_2}$

   алфавита $A = \{\alpha_1, \alpha_2, ..., \alpha_n\}$.
2. Определим матрицу $R = ((r_{i,j}))$ размерности $2 \times L$, которую назовём парным выравниванием последовательностей S_1 и S_2.
   Каждая позиция $r_{i,j}$ матрицы содержит: либо символ алфавита A, либо символ "-" (*indel*).

Определение парного выравнивания

Матрица R должна удовлетворять следующим условиям:

1. строка $R_1 = r_{1,1} r_{1,2} \ldots r_{1,L}$ матрицы R получена из строки S_1, где в некоторых позициях вставлены символы indel, следовательно $L_1 \leqslant L$.
2. строка $R_2 = r_{2,1} r_{2,2} \ldots r_{2,L}$ матрицы R получена из строки S_2, где в некоторых позициях вставлены символы indel, следовательно $L_2 \leqslant L$.
3. В матрице R отсутствуют столбцы, состоящие целиком из символов indel.

Поиск парного выпавнивания

Среди всех возможных парных выравниваний ищется выравнивание, оптимальное относительно некоторой оценочной функции. Оценочная функция обычно отражает биологический смысл выравнивания.

Биологический смысл выравнивания

Предположим, что выравниваемые последовательности имеют общую последовательность предка. В процессе эволюции наблюдаемые последовательности эволюционировали посредством следующих операций (мутаций):

1. замена символа,
2. удаление символа,
3. вставка символа.

Биологический смысл выравнивания

Не все подстановки символов равновероятны: аминокислотные остатки подразделяются по степени схожести в зависимости от своих физико-химических свойств (заряду, гидрофильности/гидрофобности, размеру и т.п.).
Вероятности замены определены опытным путём и собраны в одну из матриц:

- BLOSUM50
- BLOSUM62
- BLOSUM80
- PAM

Расчёт матрицы замен (повторение)

Функция подстановок может быть вычислена следующим образом: $substitute(a, b) = \ln(\frac{P_{a,b}}{P_a P_b})$, где a и b символы алфавита A.
Пусть P_a — вероятность встретить аминокислотный остаток a в соответствующей последовательности, $P_{a,b}$ — вероятность замены a на b в процессе эволюции.

Матрица замен BLOSUM50 (повторение)

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-4	15	2	-3
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5

Оценочная функция

$$score(R_1, R_2) = \sum_{i=1}^{L} substitute(r_{1,i}, r_{2,i}) - \sum_{j=1}^{G} gap\_penalty(g_j),$$

где g_j – один из гэпов. (2)

Штрафы за вставку гэпов

Функция штрафов $gap\_penalty(g_j)$ задаёт размер штрафа за вставленный гэп. Эта функция вычисляется для всех гэпов в обоих строках выравнивания (G — множество всех гэпов выравнивания, а g_j — один из них) следующим образом:

$$gap\_penalty(g) = cost_{begin} + length(g) \cdot cost_{extension}.$$

Здесь: $cost_{begin}$ — штраф за инициацию гэпа, $cost_{extension}$ — штраф за удлинение на один символ.

Алгоритм динамического программирования (Нидельман - Вунш)

Оценочной функции сопоставляется матрица score размерности $(L_1 + 1) \times (L_2 + 1)$:

$$score(s_i, s_j) = max \begin{cases} score(s_{i-1}, s_{j-1}) + substitute(s_i, s_j), \\ score(s_{i-1}, s_j) - gap\_penalty(g), \\ score(s_i, s_{j-1}) - gap\_penalty(g). \end{cases} \quad (3)$$

Алгоритм динамического программирования

1. Прямой ход. По формуле (3) рекурсивно заполняется матрица score с первых элементов к последним, при этом с числом дополнительно сохраняется действие.
2. Обратный ход. В каждой строке производятся действия, в обратном порядке к заполнению матрицы — с конца к началу.

Динамическое программирование: прямой ход

Начальный этап. Первая строка и первый столбец заполняются нулями. (За совпадение – 1, за гэп и ошибку – -1)

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

Динамическое программирование: прямой ход

Рекурсивное заполнение матрицы от элементов с меньшим номером к правому нижнему углу матрицы.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1									
A	0	1	2									
T	0	1	2									
C	0	1	2									
G	0	1	2									
A	0	1	2									

Динамическое программирование: прямой ход

Рекурсивное заполнение матрицы от элементов с меньшим номером к правому нижнему углу матрицы.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1	2	2	2	2
A	0	1	2	2	2	2	2	2	2	2	2	3
T	0	1	2	2	3	3	3	3	3	3	3	3
C	0	1	2	2	3	3	4	4	4	4	4	4
G	0	1	2	2	3	3	4	4	5	5	5	5
A	0	1	2	3	3	3	4	5	5	5	5	6

Динамическое программирование: обратный ход

Начинаем с последней строчки или последнего столбца.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1	2	2	2	2
A	0	1	2	2	2	2	2	2	2	2	2	3
T	0	1	2	2	3	3	3	3	3	3	3	3
C	0	1	2	2	3	3	4	4	4	4	4	4
G	0	1	2	2	3	3	4	4	5	5	5	5
A	0	1	2	3	3	3	4	5	5	5	5	6

a
|
a

Динамическое программирование: обратный ход

От выбранной позиции просматриваем соседние клетки (i,j-1), (i-1,j), (i-1,j-1) выбираем одну из тех по которой могла получится клетка (i,j). Если вариантов несколько, выбирается диагональ.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1	2	2	2	2
A	0	1	2	2	2	2	2	2	2	2	2	3
T	0	1	2	2	3	3	3	3	3	3	3	3
C	0	1	2	2	3	3	4	4	4	4	4	4
G	0	1	2	2	3	3	4	4	5	5	5	5
A	0	1	2	3	3	3	4	5	5	5	5	6

a
|
a

Динамическое программирование: обратный ход

Следующая позиция выбирается однозначно. В процессе выбора восстанавливается действие, произведённое в ячейке.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	-	-
G	0	1	1	1	1	1	1	1	1	1	-	-
G	0	1	1	1	1	1	1	1	2	2	-	-
A	0	1	2	2	2	2	2	2	2	2	-	-
T	0	1	2	2	3	3	3	3	3	3	-	-
C	0	1	2	2	3	3	4	4	4	4	4	-
G	0	1	2	2	3	3	4	4	5	5	5	-
A	-	-	-	-	-	-	-	-	-	-	-	6

```
ta
 |
-a
```

Динамическое программирование: обратный ход

Очередное действие, выбирается диагональ.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	-	-	-
G	0	1	1	1	1	1	1	1	1	-	-	-
G	0	1	1	1	1	1	1	1	2	-	-	-
A	0	1	2	2	2	2	2	2	2	-	-	-
T	0	1	2	2	3	3	3	3	3	-	-	-
C	0	1	2	2	3	3	4	4	4	-	-	-
G	0	1	2	2	3	3	4	4	5	5	5	-
A	-	-	-	-	-	-	-	-	-	-	-	6

```
tta
  |
--a
```

Динамическое программирование: обратный ход

Результирующее выравнивание. (Возможны были и другие варианты выравнивания с тем же значением качества.)

		G	A	A	T	T	C	A	G	T	T	A
	0	-	-	-	-	-	-	-	-	-	-	-
G	-	1	-	-	-	-	-	-	-	-	-	-
G	-	-	1	-	-	-	-	-	-	-	-	-
A	-	-	-	2	2	-	-	-	-	-	-	-
T	-	-	-	-	-	3	-	-	-	-	-	-
C	-	-	-	-	-	-	4	4	-	-	-	-
G	-	-	-	-	-	-	-	-	5	5	5	-
A	-	-	-	-	-	-	-	-	-	-	-	6

```
gaattcagtta
| | || |   |
gga-tc-g--a
```

Алгоритм динамического программирования (Смит - Уотерман)

Оценочной функции сопоставляется матрица score размерности $(L_1 + 1) \times (L_2 + 1)$, которая является модификацией формулы (3):

$$score(s_i, s_j) = max \begin{cases} 0, \\ score(s_{i-1}, s_{j-1}) + substitute(s_i, s_j), \\ score(s_{i-1}, s_j) - gap\_penalty(g), \\ score(s_i, s_{j-1}) - gap\_penalty(g). \end{cases}$$

Особенности алгоритма Смита - Уотермана

- Не допускаются отрицательные значения в матрице. Считается, что выгоднее прервать выравнивание и получить неотрицательное значение score.
- Обратный ход начинается с максимального значения в матрице.
- При обратном ходе, если алгоритм натыкается на 0 значение, выравнивание прекращается. Тем самым алгоритм ищет наилучшее локальное выравнивание. Тоесть выравниваются две подстроки.

Множественное выравнивание

Множественное выравнивание (выравнивание более чем двух последовательностей) определяется по аналогии с парным выравниванием как матрица $R = ((r_{i,j}))$ размерности $M \times L$, где M — число выравниваемых последовательностей.

Реализации алгоритмов

- MAFFT
- T-COFFE
- ClustalW
- MUSCLE
- Dialign

Этапы построения множественного выравнивания (прогрессивное выравнивание)

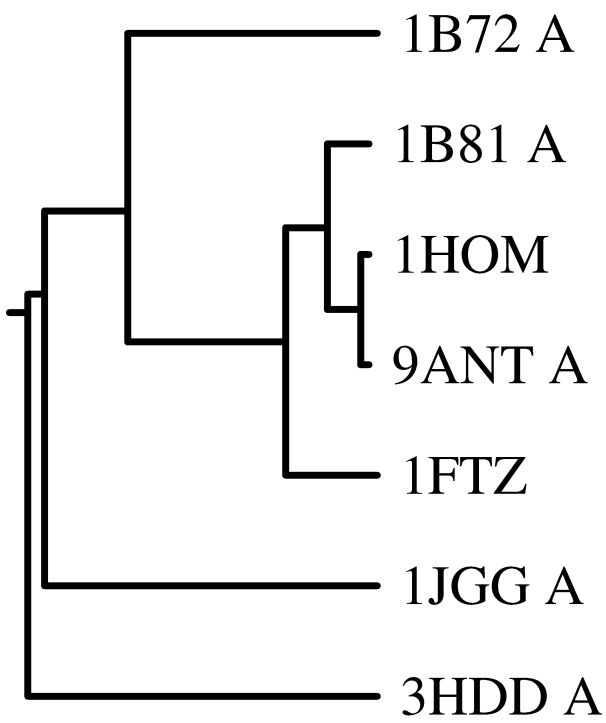
1. построение матрицы схожести
2. построение кластерного дерева алгоритмами UPGMA или присоединения ближайшего соседа (Neighbor Joining)
3. построение выравнивания по дереву от листьев к корню.

Матрица схожести в алгоритме MUSCLE

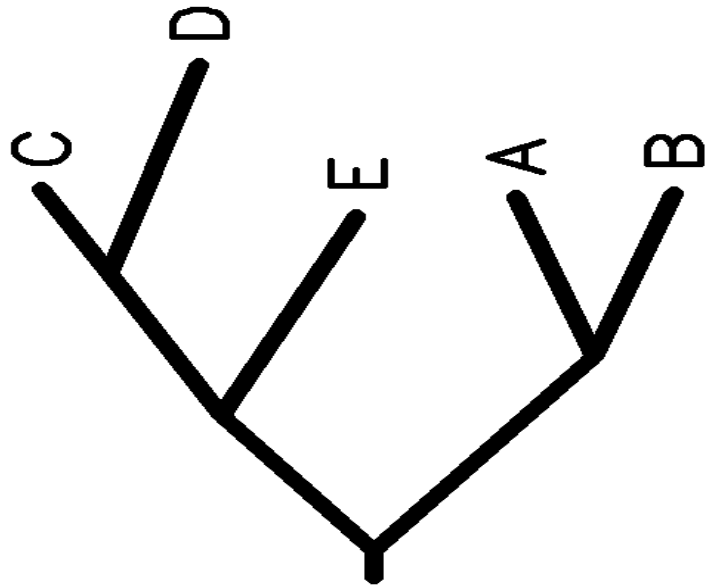
$$similarity(S_i, S_j) = \sum_{m=1}^{M} \frac{min(frequency(S_i, \tau_m), frequency(S_j, \tau_m))}{min(length(S_i), length(S_j)) - k + 1}.$$

Здесь τ_m один из различных k-меров, присутствующих в последовательности;
$frequency(S_i, \tau_m)$ — число раз, которое τ_m встретился в последовательности S_i; M — общее число различных k-меров, присутствующих в последовательности.

Кластерное дерево



Формат представления дерева (Newick format)



Дерево представляется скобочной структурой:
(((C:3.2,D:8.0):5.5,E:7.7):5.2,(A:6.1,B:6.3):7.5);
Затем при помощи программы phylip можно построить изображение.

Алгоритм кластеризации UPGMA

UPGMA - Unweighted Pair Group Method using arifmetic Averages.

- Кластеризует элементы множества, с заданными растояниями между всеми элементами путём построения бинарного кластерного дерева.
- Построение ведётся на основе минимизации среднего расстояния между кластерами.

Алгоритм кластеризации UPGMA: расстояния между кластерами

Пусть имеется 2-кластера C_f и C_l включающие m и n элементов соответственно. Пусть $d_{i,j}$ – расстояние между 2-мя элементами. Тогда расстояние между двумя кластерами определяется так:

$$\rho(C_f, C_l) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} d_{i,j}}{\mu(C_f)\mu(C_l)} \tag{4}$$

Где $\mu(C_f) = m$ – число элементов кластера.

Алгоритм UPGMA: расстояние до объединения кластеров

Пусть $C_k = C_i \cup C_j$, а C_l – другой кластер, тогда расстояние от C_l до C_k вычисляется следующим образом:

$$\rho(C_l, C_k) = \frac{\rho(C_i, C_l)\mu(C_i) + \rho(C_j, C_l)\mu(C_j)}{\mu(C_i) + \mu(C_j)} \tag{5}$$

Здесь $\rho(C_i, C_l)$ и $\mu(C_i)$ те же, что и в формуле (4).

Алгоритм UPGMA: начальная стадия

1. Строим множество кластеров. Каждому элементу i сопоставляется кластер из одного элемента C_i. Каждому такому элементу ставится в соответствие лист дерева.

Алгоритм UPGMA: стадия построения ветвей дерева

1. Ищется пара кластеров для которых $\rho(C_i, C_j)$ минимально. Если несколько таких, то выбирается произвольный.
2. Построим объединение этих 2-х кластеров $C_k = C_i \cup C_j$ и для всех остальных кластеров C_l по формуле (5) персчитывается $\rho(C_l, C_k)$
3. В дерево помещается новый узел, где к листьям C_i и C_j ведут 2 ребра длинны $\rho(C_i, C_j)/2$. Кластеры C_i и C_j могут соответствовать промежуточным узлам дерева.
4. Кластеры C_i и C_j удаляются из множества кластеров.

Алгоритм UPGMA: правила остановки и итерации

1. С новым набором кластеров стадия построения ветвей дерева повторяется.
2. Алгоритм останавливается с исчерпанием множества кластеров. На последнем шаге 2 кластера объединятся и получится корень дерева.

Алгоритм кластеризации Neighbor Joining (NJ)

Метод присоединения ближайшего соседа.

- Кластеризует элементы множества, с заданными растояниями между всеми элементами путём построения бинарного кластерного дерева.
- Построение ведётся на основе слияния 2-х кластеров близких друг к другу и наиболее удалённых от остальных кластеров.
- Строит дерево без корня.

Алгоритм кластеризации NJ: расстояния между кластерами

Пусть имеется 2-элемента i, j множества C. Всего элементов в множестве m. Пусть $d_{i,j}$ – расстояние между 2-мя элементами. Тогда расстояние между двумя кластерами, составляющими эти 2 элемента, по отношению к остальным элементам множества определяется так:

$$\rho(C_i, C_j) = d_{i,j} - (r_i + r_j)$$

$$r_i = \frac{\sum_{k \neq i,j}^{m} d_{i,k}}{\mu(C \setminus (C_i \cup C_j))} \quad (6)$$

Где $\mu(C) = m$ – число элементов кластера.

Алгоритм NJ: начальная стадия

1. Строим множество кластеров. Каждому элементу i сопоставляется кластер из одного элемента C_i. Каждому такому элементу ставится в соответствие лист дерева.

Алгоритм NJ: стадия построения ветвей дерева

1. Ищется пара кластеров для которых $\rho(C_i, C_j)$ минимально. Если несколько таких, то выбирается произвольный. Величина $\rho(C_i, C_j)$ определяется по формуле (6)
2. В дерево помещается новый узел h, где к узлам C_i и C_j от него ведут рёбра, где длинна определяется как сумма расстояний до узла за вычетом суммы расстяний до другой ветки дерева.

$$b_{i,h} = \frac{1}{2}\left(d_{i,j} + \frac{(\sum_{k\neq i,j}^{m} d_{i,k} - \sum_{k\neq i,j}^{m} d_{j,k})}{\mu(C \setminus (C_i \cup C_j)}\right)$$

Или для экономии

$$b_{i,h} = \frac{1}{2}\rho(i,j) + \frac{\sum_{k\neq i,j}^{m} d_{i,k}}{\mu(C \setminus (C_i \cup C_j)} \qquad (7)$$

Алгоритм NJ: стадия построения ветвей дерева (продолжение)

1. Построим объединение этих 2-х кластеров $C_h = C_i \cup C_j$ и для всех остальных кластеров C_l по формуле персчитывается:

$$d_{l,h} = \frac{1}{2}(d_{i,l} - b_{i,h}) + \frac{1}{2}(d_{j,l} - b_{j,h}).$$

   Здесь $b_{i,h}, b_{j,h}$ вычисляются по формуле (7).
2. Кластеры C_i и C_j удаляются из множества кластеров, а кластер C_h добавляется.

Алгоритм NJ: правила остановки и итерации

1. С новым набором кластеров стадия построения ветвей дерева повторяется.
2. Алгоритм останавливается по исчерпанию множества кластеров. На последнем шаге 2 кластера объединятся ребром длины $d_{i,j}$

Построение выравнивания по кластерному дереву

1. Пара листьев выравнивается, с помощью динамического программирования.
2. Выровненная пара последовательностей становится профилем выравнивания. С профилем происходит работа как с одной строкой. символы '-' можно вставлять только целиком в весь столбец.
3. По модифицированному алгоритму динамического программирования строится выравнивание профилей. В результате получаем новый профиль.
4. Так продолжается до тех пор пока не дойдём до корня дерева.

Функция оценки качества профилей выравнивания

$$score(R_x, R_y) = \sum_{i=1}^{L} \sum_{\alpha} \sum_{\beta} f(\alpha, R_{x,i}) f(\beta, R_{y,i}) substitute(\alpha, \beta) - \sum_{j=1}^{G} gap\_penalty(g_j),$$

$$\alpha \in column\_symbols(R_{x,i}),$$
$$\beta \in column\_symbols(R_{y,i}).$$

Функция $column\_symbols(R_{x,i})$ вычисляет множество символов, которые встречаются в столбце i промежуточного выравнивания x. Функция $f(\alpha, R_{x,i})$ определяет частоту встречаемости символа α в столбце i промежуточного выравнивания x.

Параллельные алгоритмы множественного выравнивания

Существует некоторое количество параллельных алгоритмов множественного выравнивания, реализованных на MPI:

- ClustalW-MPI
- PDIALIGN

Эти алгоритмы обладают некоторыми недостатками, главный из которых — серьёзные требования к объёму памяти на узле.

Список литературы


Гасфилд Д. “Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология”.


Р. Дурбин, Ш. Эдди, А. Крог, Г. Матчисон “Анализ биологических последовательностей”.