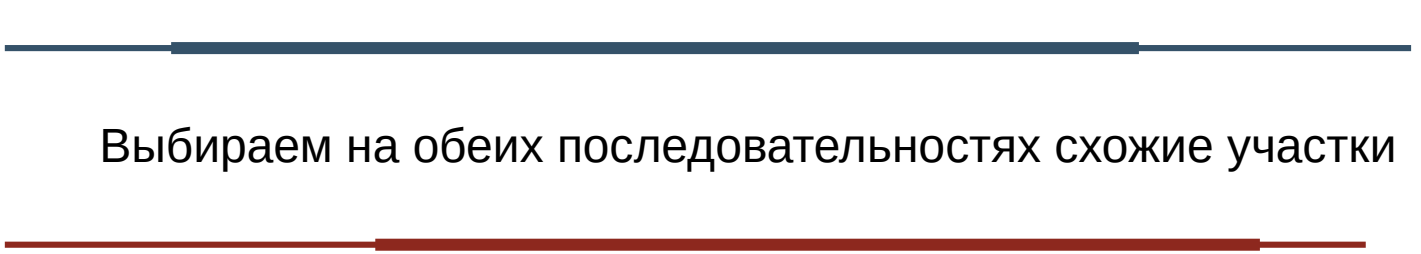
Ещё о выравнивании и BLAST

С.А. Спирин, 28 апреля 2020

Локальное выравнивание

Локальное выравнивание

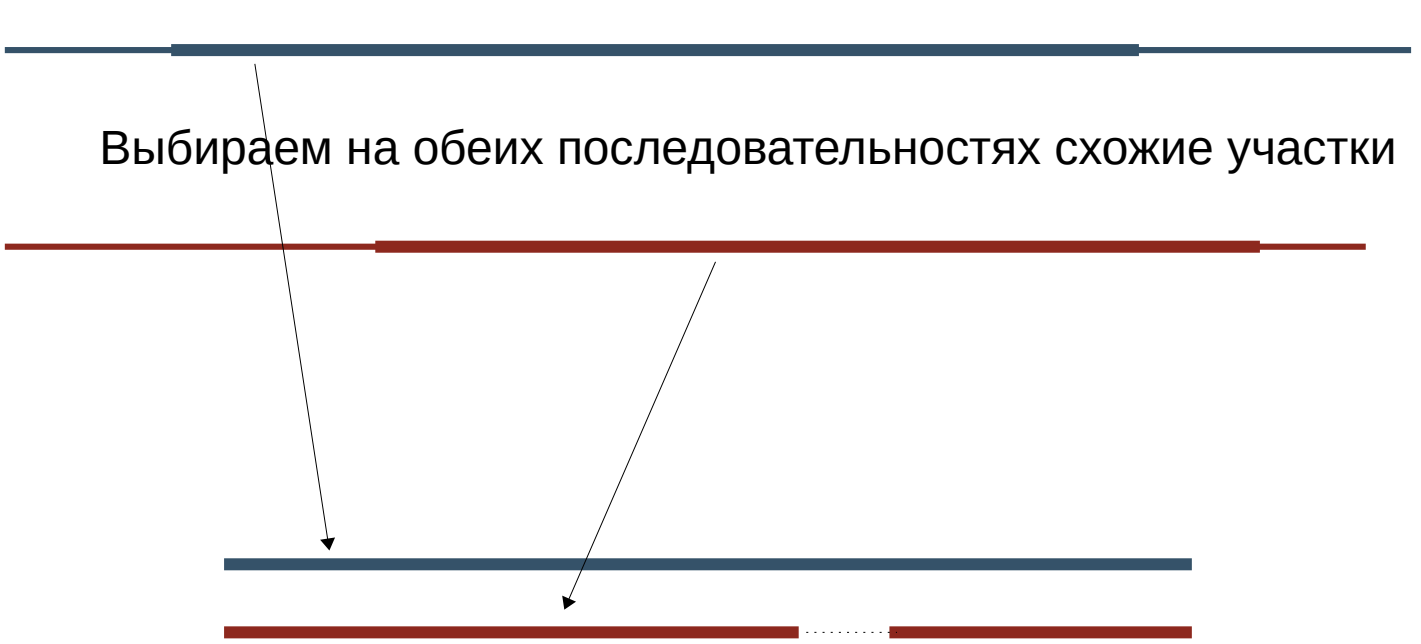


Seq1
(длины L_1)

Выбираем на обеих последовательностях схожие участки

Seq2
(длины L_2)

Локальное выравнивание



Локальное выравнивание

Покрытие 1 = l_1/L_1

Покрытие 2 = l_2/L_2

Алгоритмы Нидельмана – Вунша и Смита – Уотермена

На входе:
- Пара последовательностей
- Матрица замен
- Штрафы за открытие и удлинение гэпа

На выходе у алгоритма Нидельмана – Вунша (NW, Saul B. Needleman and Christian D. Wunsch, 1970):
- Оптимальное глобальное выравнивание
- Вес этого выравнивания

На выходе у алгоритма Смита – Уотермена (SW, Temple F. Smith and Michael S. Waterman, 1981):
- Оптимальное локальное выравнивание
- Вес этого выравнивания

Оптимальное — значит с наибольшим весом среди всех возможных выравниваний этих последовательностей

Число возможных выравниваний

```
TGGAGTAACCAT-
TGGGATAACCTTG
```

```
TGGAGTAACCAT-------------
------------TGGGATAACCTTG
```

```
-TGGAGTAACCAT
TGGGATAACCTTG
```

```
TGGA--GTAACCAT--
TGGGATAA---CCTTG
```

Теорема: даны последовательности длины *m* и *n*. Если выравнивания, в которых все сопоставления букв одинаковы, считать эквивалентными, то существует $C^{m}_{m+n} = C^{n}_{m+n}$ различных (неэквивалентных другу другу) выравниваний.

Например, вот эти два выравнивания эквивалентны:

```
--TGGAGTAACCAT     =     T--GGAGTAACCAT
TG-GGATAACCTTG           -TGGGATAACCTTG
```

Число возможных выравниваний

```
TGGAGTAACCAT-
TGGGATAACCTTG
```

```
TGGAGTAACCAT-----------
-----------TGGGATAACCTTG
```

```
-TGGAGTAACCAT
TGGGATAACCTTG
```

```
TGGA--GTAACCAT--
TGGGATAA---CCTTG
```

Теорема: даны последовательности длины *m* и *n*. Если выравнивания, в которых все сопоставления букв одинаковы, считать эквивалентными, то существует $C^{m}_{m+n} = C^{n}_{m+n}$ различных (неэквивалентных другу другу) выравниваний.

Идея доказательства:
В сумме в двух последовательностях *m* + *n* букв.
Выбрем изо всех *m* + *n* какие-нибудь *m* букв.
Теперь сопоставим **выбранные** буквы второй последовательности (пусть их оказалось *k* штук) **невыбранным** буквам первой последовательности (в первой выбрано *m* – *k*, значит не выбрано *k*). Это задаёт выравнивание, взаимно однозначно.

Число возможных выравниваний

C^{m}_{m+n} — очень большое число, если *m* и *n* — типичные длины белков. При *m* = *n*, C^{n}_{2n} примерно равно 2^{2n}.

Для *n* = 100 это примерно 10^{60}.

Поэтому перебрать все возможные выравнивания, дабы найти оптимальное, нереально.

Тем не менее, алгоритмы NW и SW находят их за сравнительно короткое время.
Секрет в правильном применении принципа динамического программирования. Подробности — в курсе алгоритмов биоинформатики на III курсе.

Время работы обоих алгоритмов пропорционально произведению *mn* длин последовательностей
(эта величина растёт с ростом *m* и *n* намного медленнее, чем C^{m}_{m+n})

Про множественные выравнивания

Для множественных выравниваний (число последовательнсотей больше двух) алгоритмы, основанные на динамическом программировании, на практике не применяются.

Дело в том, что время работы таких алгоритмов по прежнему пропорционально произведению длин **всех** входных последовательностей, что, как правило, слишком много.

Поэтому существует множество **эвристических** алгоритмов множественного выравнивания

(Эвристических — значит не решающих никакой точно поставленной формальной задачи, вроде нахождения выравнивания с максимальным весом)

Самые популярные из них вы можете видеть в меню программы

Jalview: Web service → Alignment

На kodomo из командной строки доступны muscle, mafft, prank, edialign, emma.

(edialign — это "EMBOSS-обёртка" программы Dialign, а emma — программы ClustalW)

Идея алгоритма BLAST

Нам нужно найти в банке последовательности, хорошо (то есть с большим весом) выравнивающиеся с последовательностью запроса.

Можно было бы это делать алгоритмом SW, последовательно выравнивая каждую банковскую последовательность с запросом (и такие сервисы существуют, например ssearch на сайте ebi.ac.uk). Но при нынешних объёмах банков это работает слишком медленно.

Идея состоит в том, чтобы заранее **проиндексировать** банк.

Индексы вы видели в конце почти любой научной книги, там имеется **алфавитный** список терминов (или, например, латинских названий растений) с указанием страниц, на которых упоминается этот термин.

В случае BLAST индексами служат **слова** заданной длины из букв, встречающихся в наших последовательнсостях. Например, для белков и при длине слова 3 это AAA, AAC, AAD, …, YYY, всего $20^3 = 8000$ слов.

Перед тем, как запускать собственно поиск, создаётся таблица, в которой для каждого слова указано, в какой последовательности банка и в каком месте это слово встретилось.

Как работает BLAST?

Якорь (в примере —длины 3)

```
Query:   GSQSLAALLNKCKTPQGQRLVNQWIKQPLMDKNRIEERLNLVEAFVED
```

Сходные слова

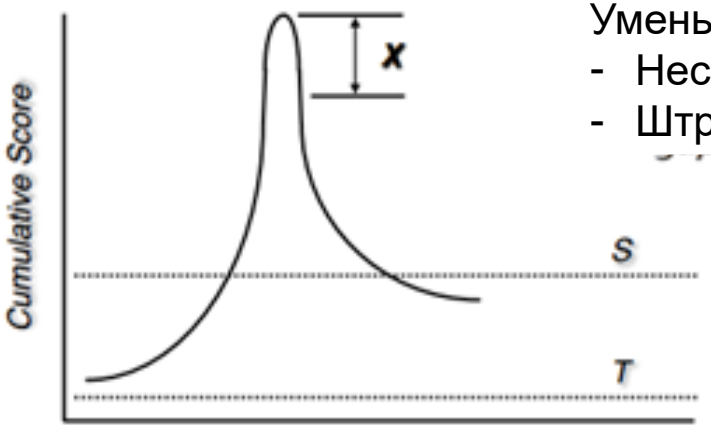
PQG	18
PEG	15
PRG	14
PKG	14
PNG	13
PDG	13
PHG	13
PMG	13
PSG	13
PQA	12
PQN	12
etc.	

=7+5+6

Порог на score

```
Query:   325  SLAALLNKCKTPQGQRLVNQWIKQPLMDKNRIEERLNLVEA  365
              +LA++L    TP+G R++ +W+ +P+ D   + ER   + A
Sbjct:   290  TLASVLDCTVTPMGSRMLKRWLHMPVRDTRVLLERQQTIGA  330
```

Поиск коротких сходных слов в банке (затравок для выравниваний)



Расширение затравки

Уменьшение за счёт:
- Несовпадений
- Штрафов за гэпы

Вопоросы и ответы про BLAST

За счёт чего BLAST работает быстро?

За счёт просмотра не всех возможных выравниваний, а только полученных расширением "затравок". Каждая "затравка" получается из слова длины k (k =2, 3, …, 6), встреченного в запросе, и очень сходного слова из какой-либо банковской последовательности.

"Затравки" находятся очень быстро благодаря предварительной индексации всех слов в банке. В результате индексации для каждого слова указано, в каких местах каких банковских последовательностей это слово встечается.

Что может поменяться при изменении параметра "Word size»?

Чем длиннее слово, тем меньше машинного времени займёт поиск.

Чем короче слово, тем чувствительнее поиск (меньше опасность пропустить хорошее выравнивание).

Standalone BLAST

BLAST можно установить на своём компьютере
(а на kodomo он уже установлен)

Предположим, вам нужно найти гомологи белка, чья последовательность — в файле myprot.fasta, в протеоме, содержащемся в файле proteom.fasta (всё в fasta-формате, BLAST других не понимает).

Придётся сначала проиндексировать ваш банк программой makeblastdb, подав ей на вход протеом (читайте makeblastdb -help)

Эта программа создаст несколько файлов, необходимых для поиска, в том числе тот самый индекс якорей (сразу для всех допустимых длин слов)

После этого можно искать программой blastp, указав ей имя файла с запросом и название проиндексированного банка
(читайте blastp -help, нужные опции: -query, -db, -out)

Standalone BLAST

Впрочем, можно использовать BLAST и для обычного локального выравнивания двух последовательностей, безо всякой индексации:

```
blastp -query seq1.fasta -subject seq2.fasta -out result.blastp
```

Но имейте в виду, что BLAST и в таком варианте не гарантирует оптимального выравнивания (это **эвристический** алгоритм)! Зато можно быстро выровнять очень длинные последовательности (команде water может не хватить памяти) и получить не одно, а много локальных выравниваний.

(На самом деле в этом варианте BLAST «на ходу» индексирует вторую последовательность)

Матрицы серии BLOSUM

Что мы хотим от матрицы аминокислотных замен?
Чтобы сопоставления букв, говорящие в пользу правильности выравнивания, вносили положительный вклад в вес, а говорящие против правильности выравнивания — отрицательный.

Матрицы серии BLOSUM

Что мы хотим от матрицы аминокислотных замен?
Чтобы сопоставления букв, говорящие в пользу правильности выравнивания, вносили положительный вклад в вес, а говорящие против правильности выравнивания — отрицательный.

Как этого добиться?
Для каждой пары букв посчитаем, как часто они встречаются в **достоверно правильных** выравниваниях в **одной позиции**, обозначим частоту пары через q(a,b) (где a,b — две буквы) (имеется в виду относительная частота, $0 \le q(a,b) \le 1$)
Теперь для каждой пары букв посчитаем, как часто они могут оказаться друг против друга **по случайным причинам**. Это просто произведение частот букв в банке, p(a)p(b).

Посчитаем **отношение правдоподобия** q(a,b)/p(a)p(b).

Если оно больше 1, то буквы чаще встречаются в гомологичных позициях, это аргумент в пользу их сопоставления. Если меньше 1, то наоборот.
Теперь элемент матрицы — это **логарифм** отношения правдоподобия**:** $\log(q(a,b)/p(a)p(b))$

Матрицы серии BLOSUM

Где взять достоверно правильные выраваниния?

Steven Henikoff & Jorja Henikoff в 1992 г. придумали использовать для этого базу данных BLOCKS, коллекцию множественных **локальных** выравниваний **без гэпов**, созданную из надёжно гомологичных белков.

(Сама база ещё жива, но сейчас редко используется)

Они поняли, что выравнивания из базы BLOCKS **целиком** использовать плохо: в большинстве выравниваний очень много совпадающих и очень близких последовательностей, и их использование приведёт к недооценке веса сопоставлений разных, но близких по свойствам аминокислотных остатков.

Матрицы серии BLOSUM

Henikoff & Henikoff придумали **проредить** каждое выравнивание из BLOCKS, оставив в каждом выравнивании лишь последовательности, **попарно** имеющие менее N% одинаковых букв.

В результаты получилась серия матриц BLOSUM (BLOcks SUbstitution Matrix): BLOSUM30, BLOSUM35, …, BLOSUM100.

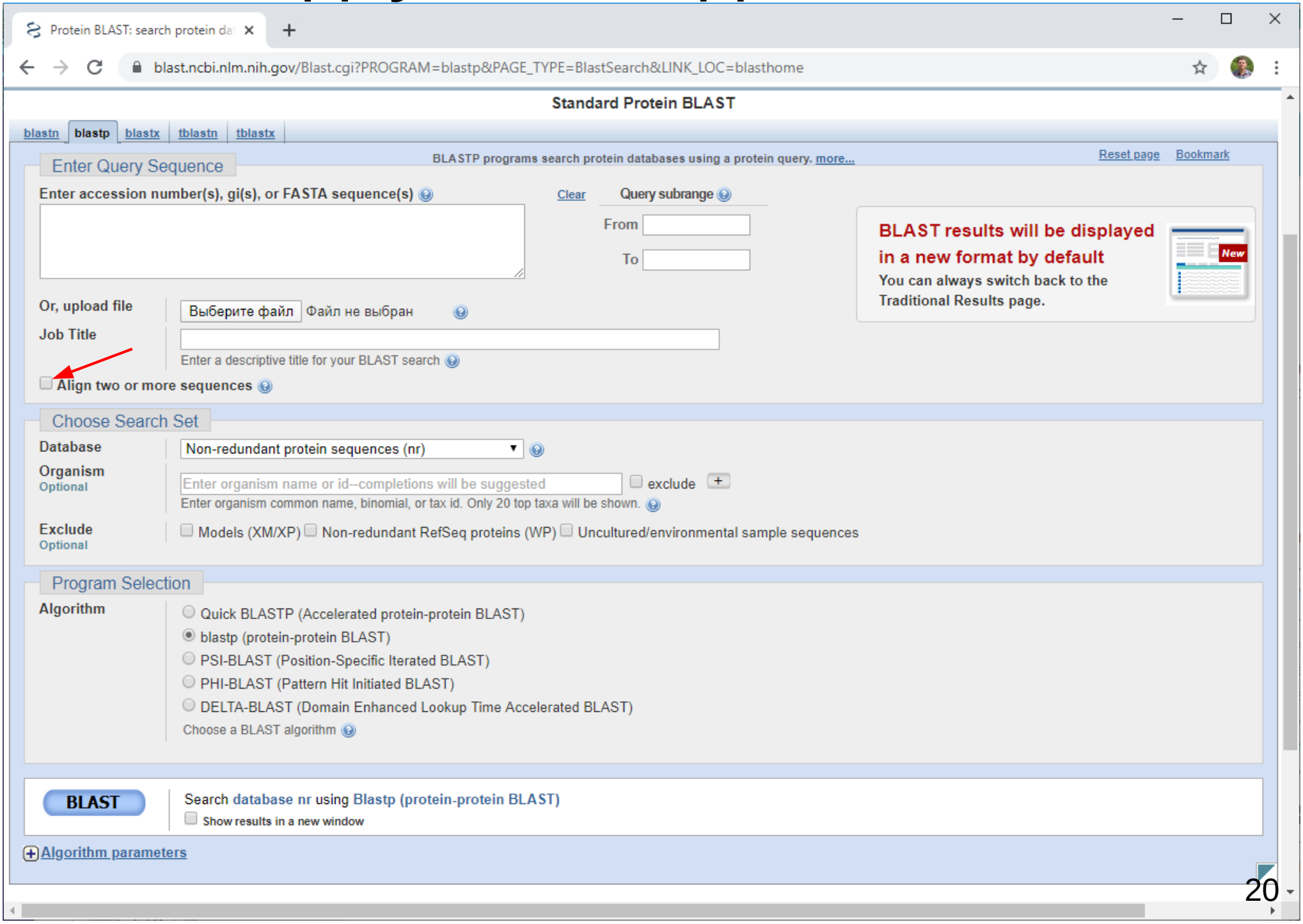
Число в конце названия матрицы означает порог на процент совпадений между парами последовательностей в «образцовых» выравниваниях.

В результате испытаний наиболее подходящей для выравнивания и поиска по сходству была признана матрица BLOSUM62.

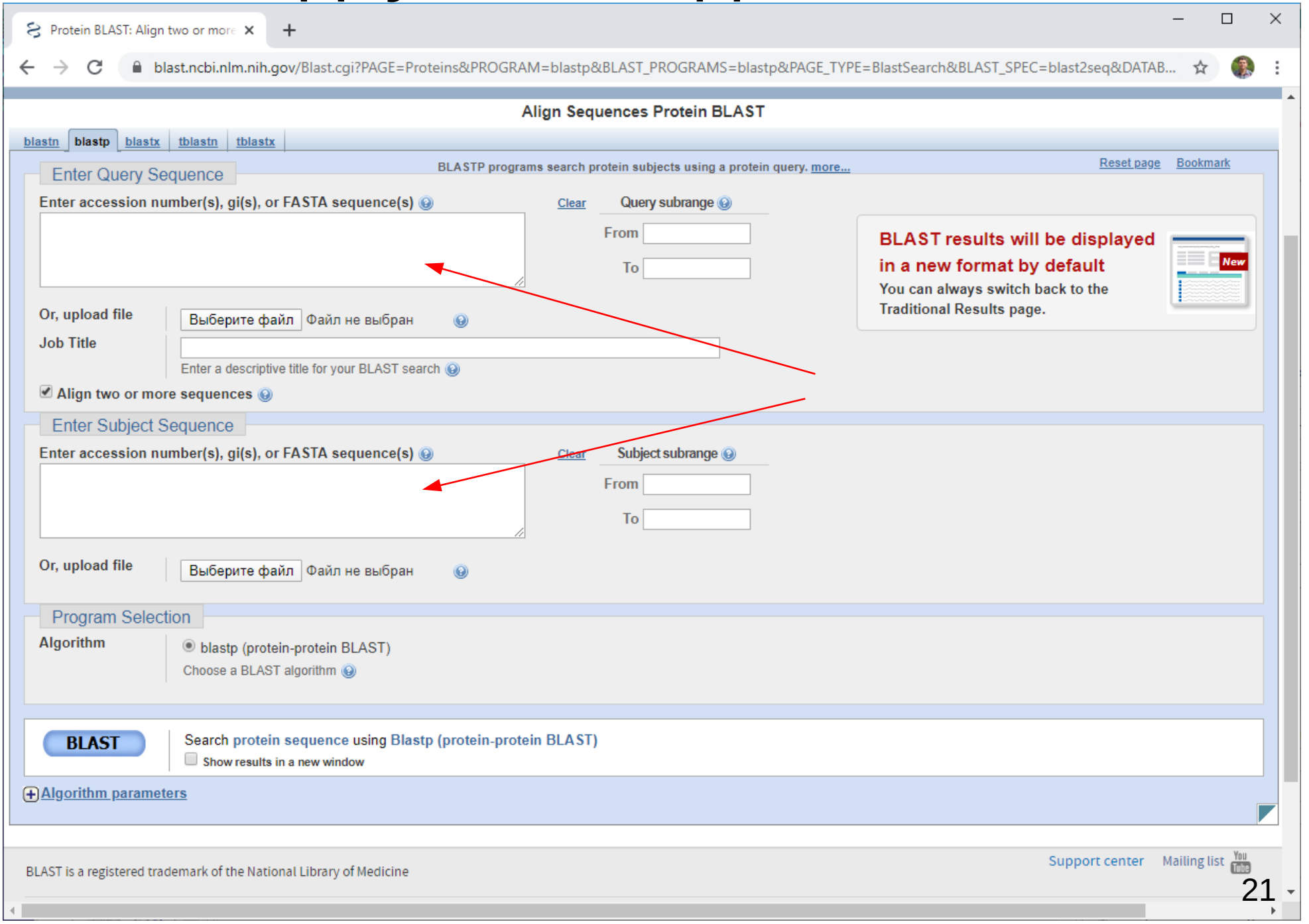
Все матрицы можно посмотреть здесь: ftp://ftp.ncbi.nlm.nih.gov/blast/matrices/

В программах needle, water, stretcher, matcher можно задать матрицу, отличную от BLOSUM62, опцией -datafile, прибавив к названию матрицы спереди "E", например -datafile EBLOSUM50
В программе blastp это опция -matrix, E не нужно, например -matrix BLOSUM45

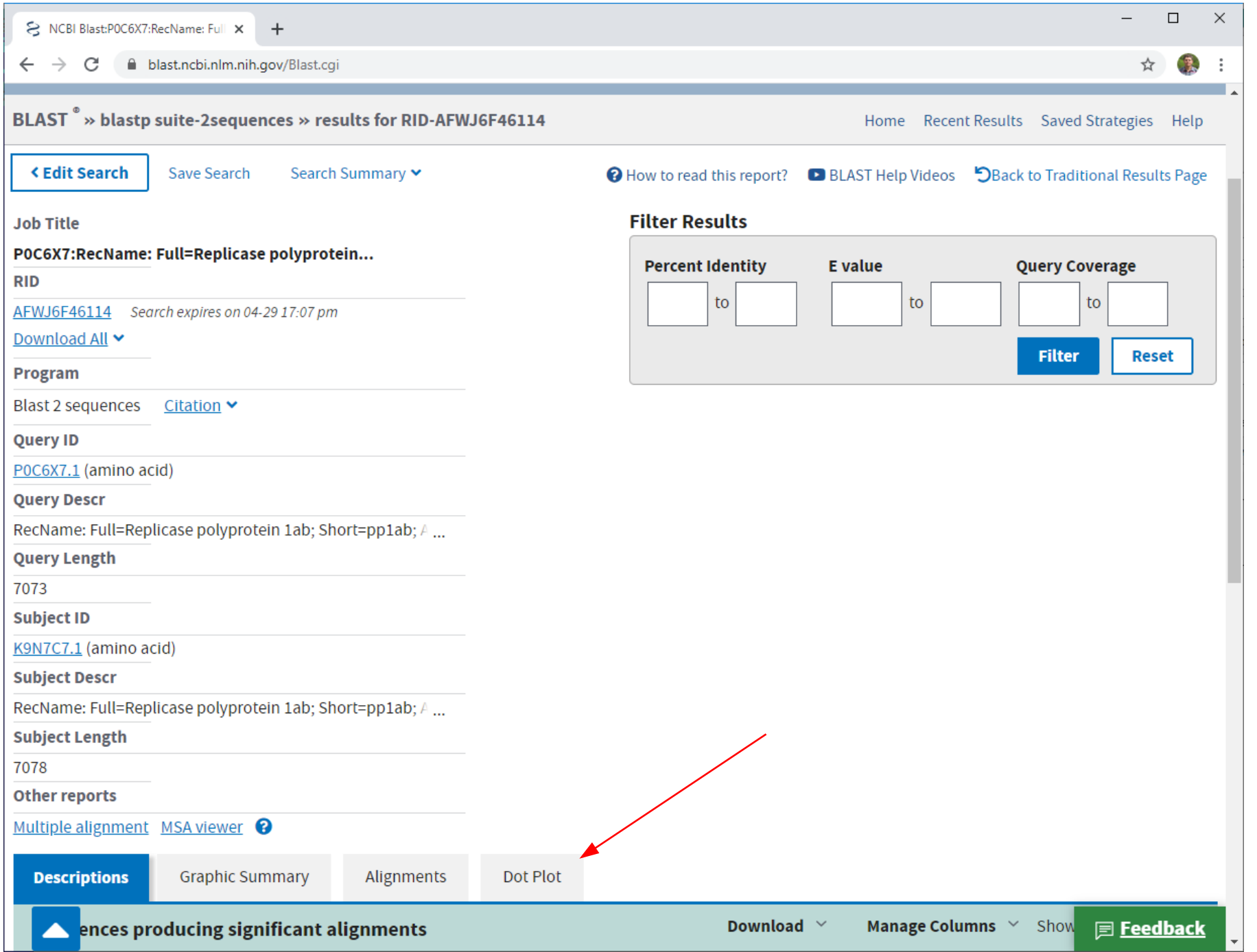
BLAST двух последовательностей



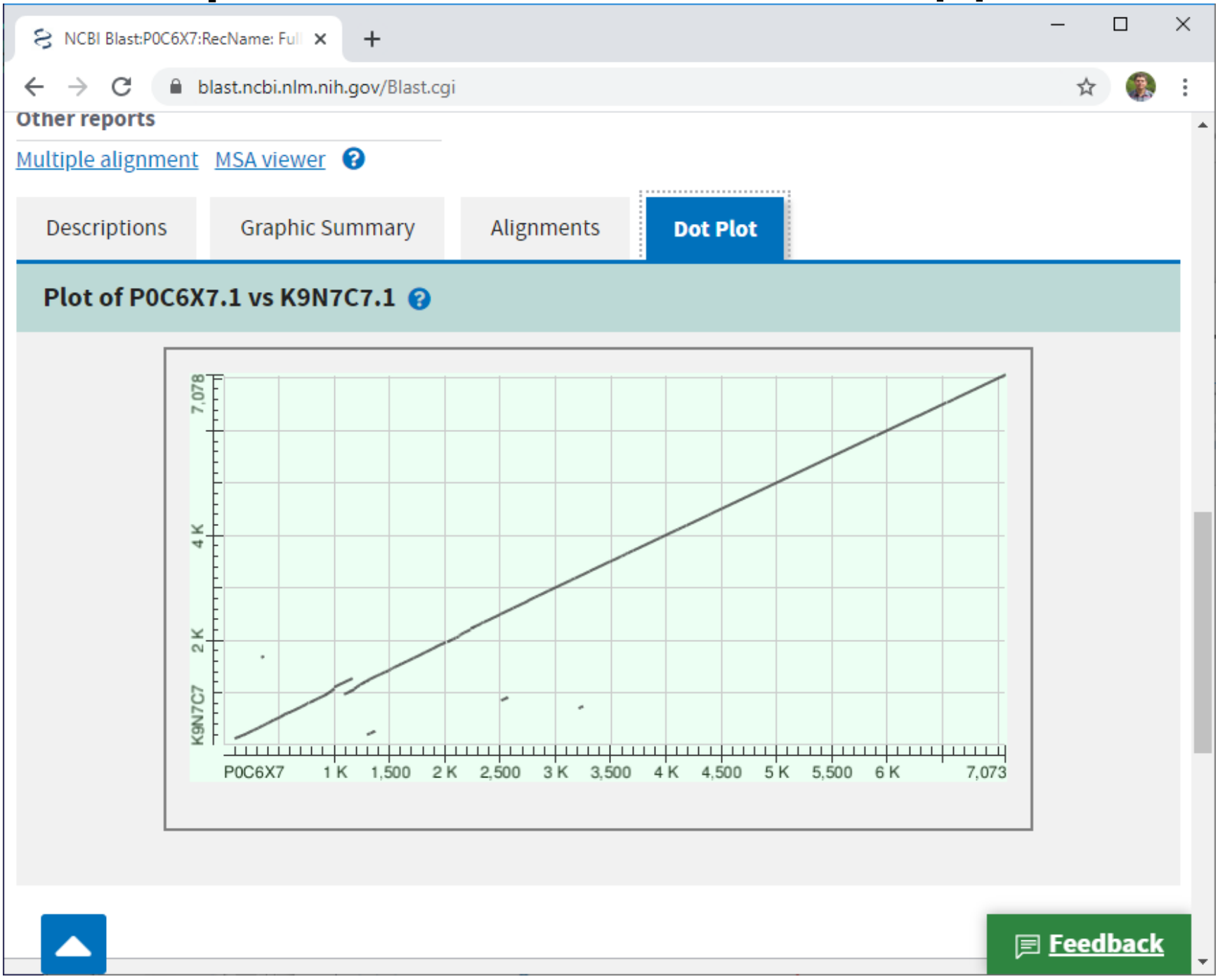
BLAST двух последовательностей



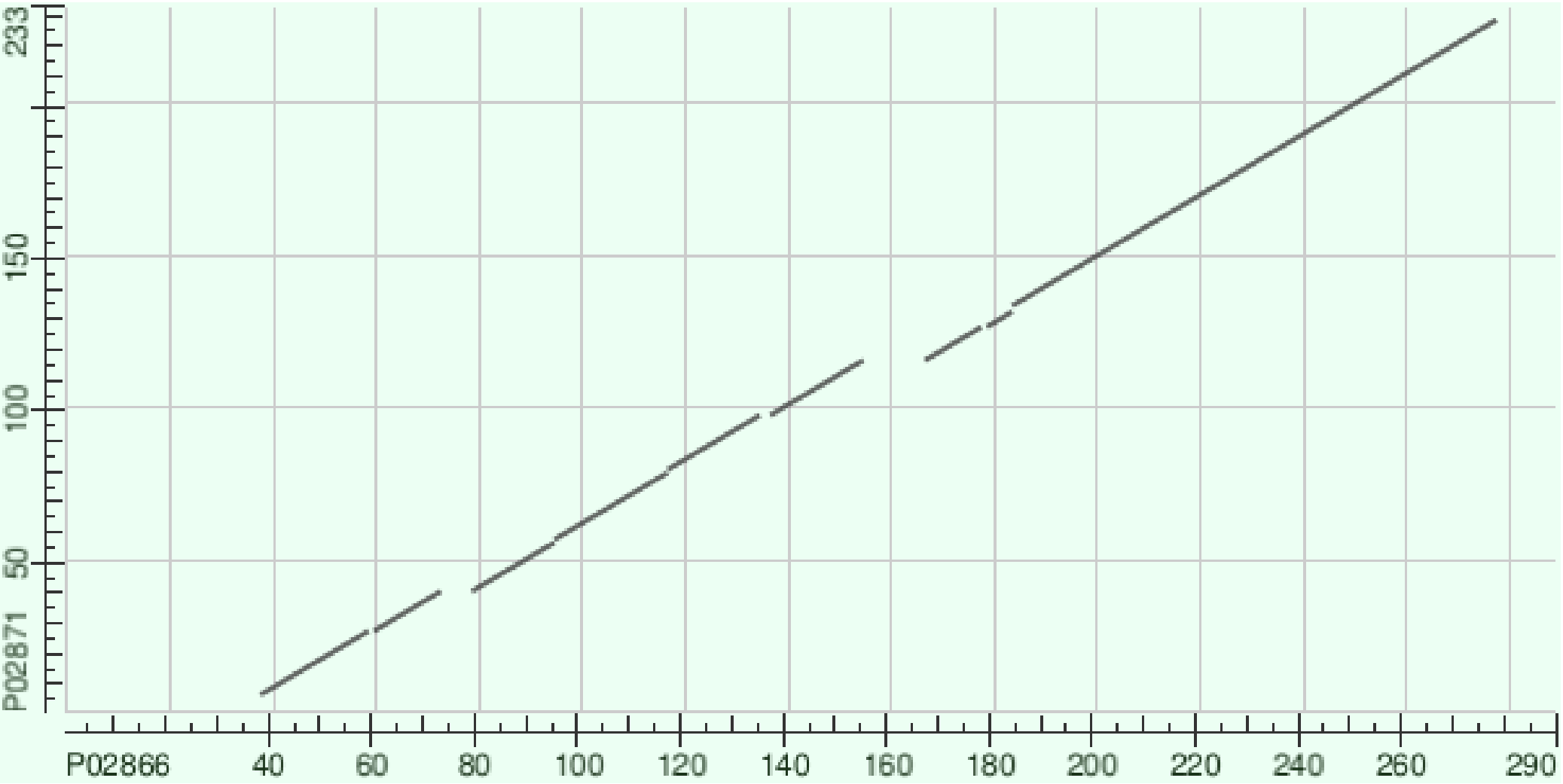
BLAST двух последовательностей



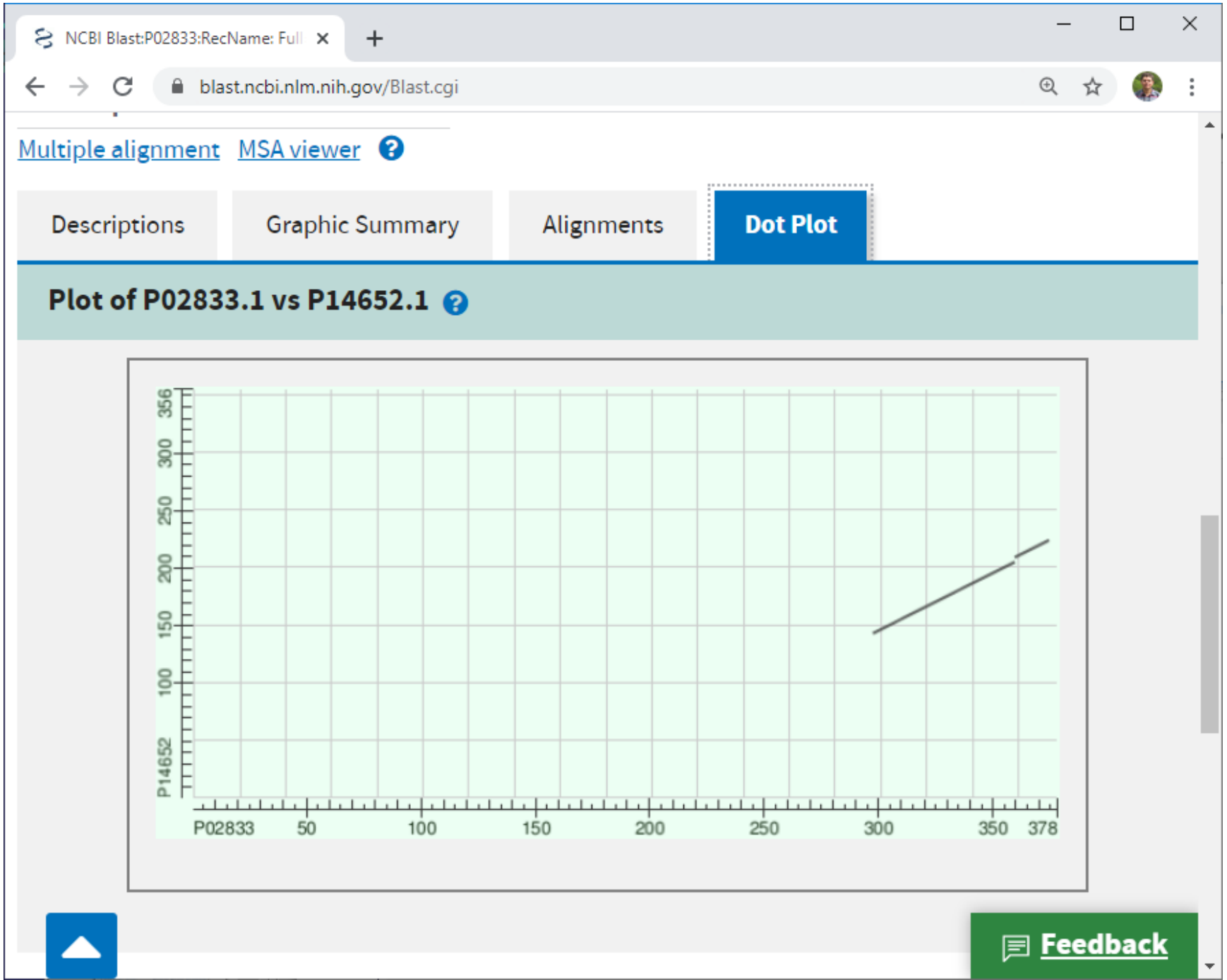
Карта локального сходства



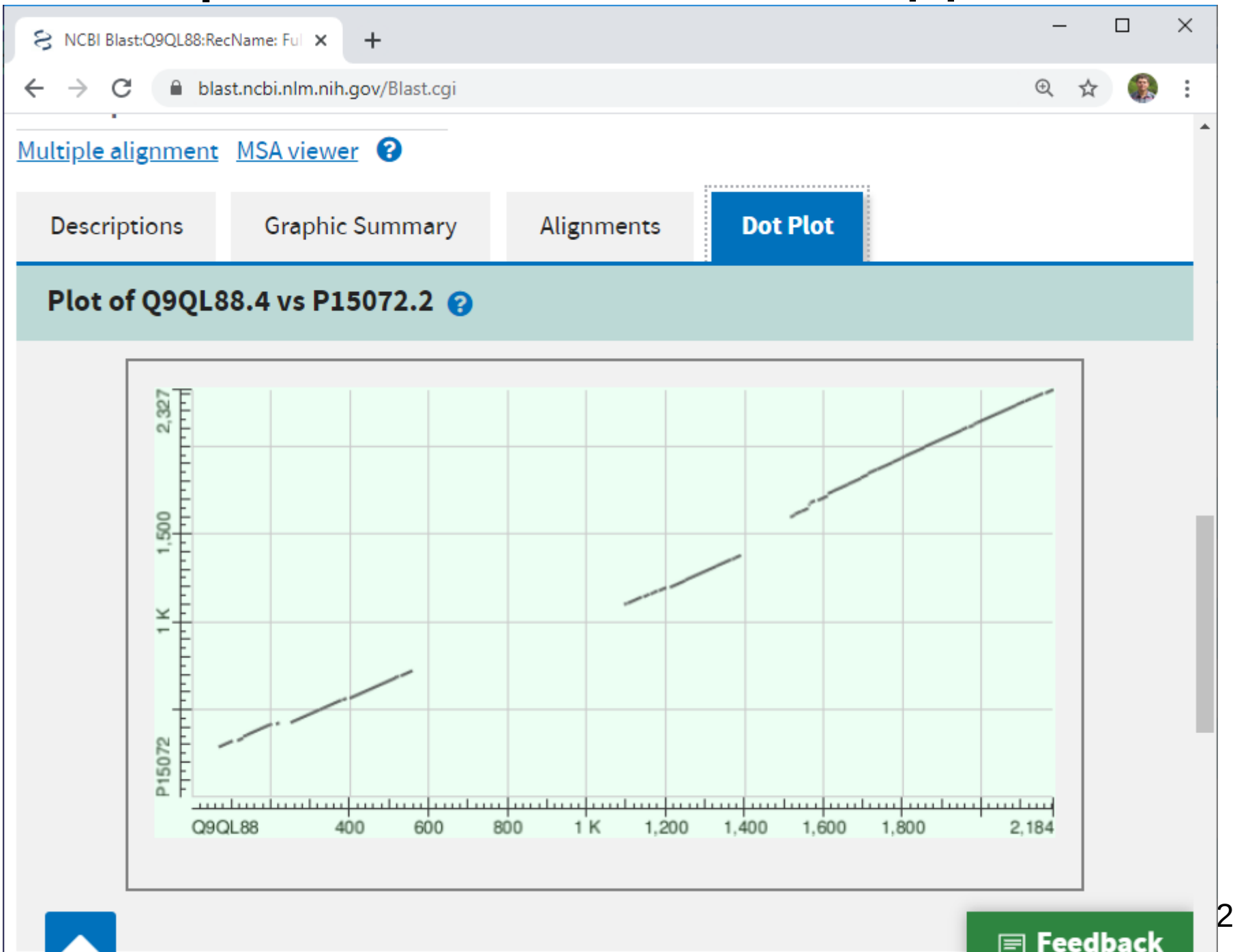
Карта локального сходства



Карта локального сходства



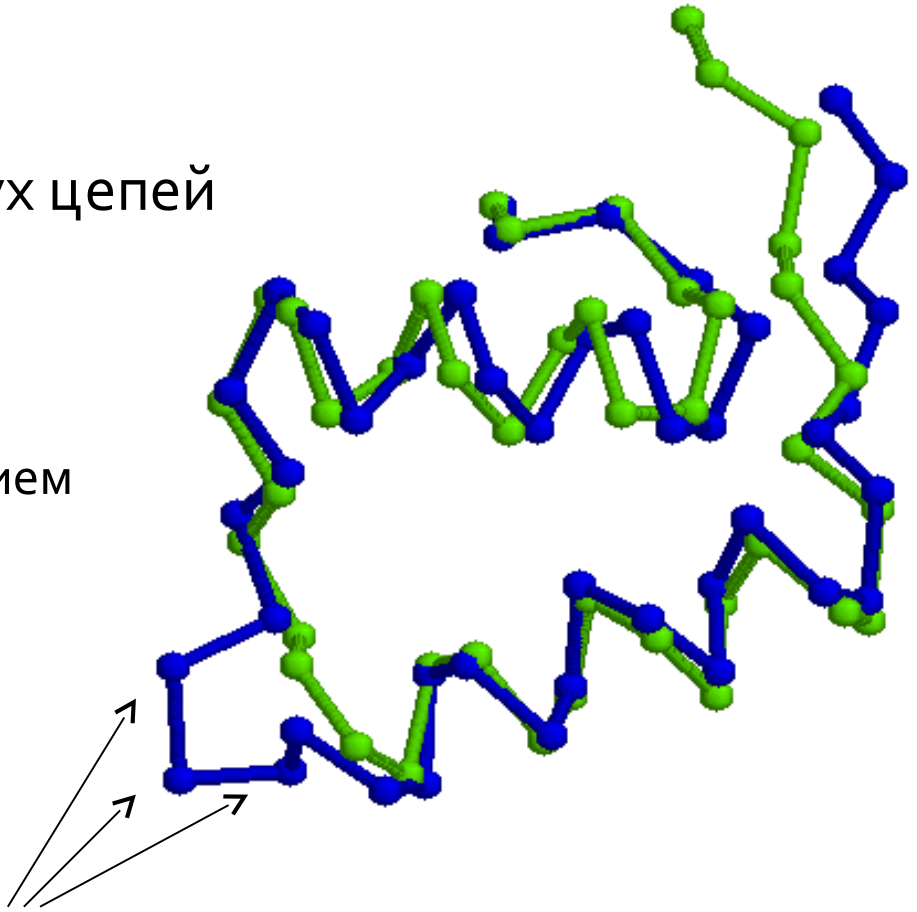
Карта локального сходства



Выравнивание на основе сопоставления пространственных структур

Большая часть остатков двух цепей соответствуют друг другу.

Соответствие в данном случае определяется хорошим наложением структур в пространстве.



Этим Cα-атомам в «синей» структуре ничего не соответствует в «зелёной»

Сравнение выравниваний

Пусть есть набор последовательностей и пусть есть **два** выравнивания этого набора последовательностей.

Например, для двух последовательностей: оптимальное глобальное и оптимальное локальное.
Или оба оптимальные глобальные, но при разных параметрах вычисления веса.
Или оптимальное и неоптимальное:

- выданное BLAST“ом;
- полученное из множественного выравнивания путём ограничения на эти две последовательности;
- полученное путём анализа пространственных структур.

Для многих последовательностей: полученные разными программами множественного выравнивания (Muscle, MAFFT, Prank, Dialign, …).

Сравнение выравниваний

Пусть есть набор последовательностей и пусть есть **два** выравнивания этого набора последовательностей.

Сравнить выравнивания — значит найти сопоставления аминокислотных остатков (нахождение их в одной колонке), которые имеются в одном выравнивании, но отсутствуют в другом.

```
AFTGAHAYL          AFTGAHAYL
AYS---AYM          AY---SAYM
```

Например, в правом выравнивании есть сопоставление 6-ого H первой последовательности 3-му S второй, а в левом нет.
Вместо этого в левом выравнивании 3-й S второй последовательности сопоставлен 3-му T первой.