

# Ещё о выравнивании и BLAST

С.А. Спирин, 28 апреля 2020

# Локальное выравнивание



Seq1  
(длины  $L_1$ )



Seq2  
(длины  $L_2$ )

# Локальное выравнивание



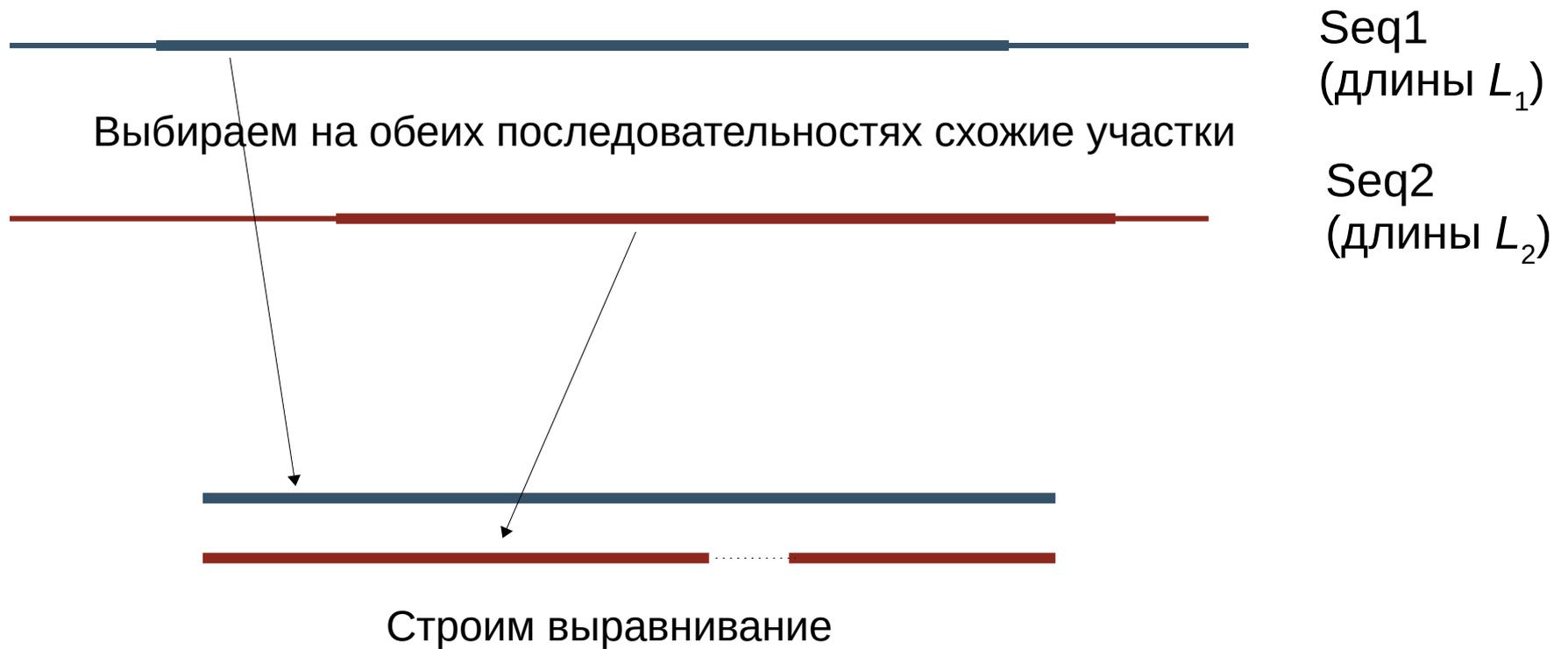
Выбираем на обеих последовательностях схожие участки



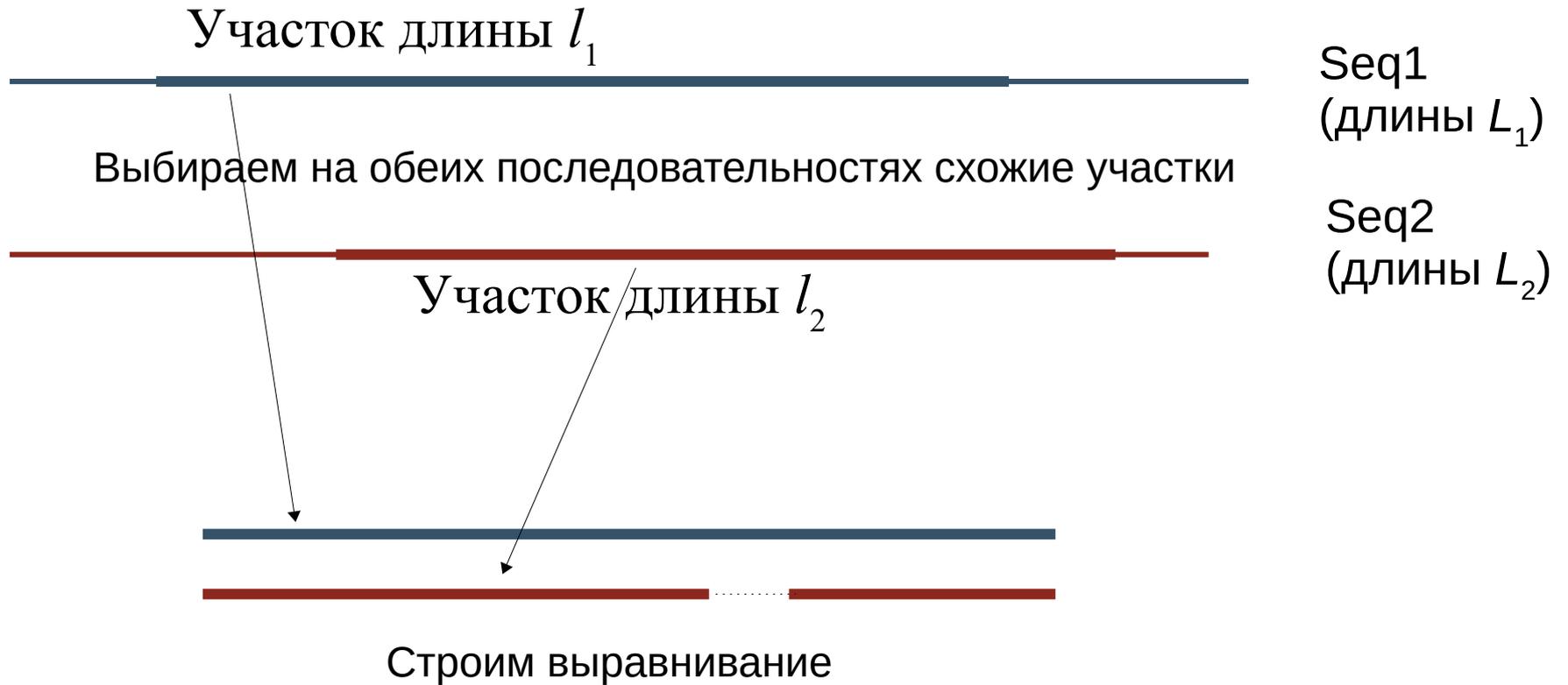
Seq1  
(длины  $L_1$ )

Seq2  
(длины  $L_2$ )

# Локальное выравнивание



# Локальное выравнивание



$$\text{Покрытие 1} = l_1 / L_1$$

$$\text{Покрытие 2} = l_2 / L_2$$

# 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  $C_{m+n}^m = C_{m+n}^n$  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  $C_{m+n}^m = C_{m+n}^n$  различных (неэквивалентных другу другу) выравниваний.

## Идея доказательства:

В сумме в двух последовательностях  $m + n$  букв.

Выберем из 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

$C_{m+n}^m$  — очень большое число, если  $m$  и  $n$  — типичные длины белков. При  $m = n$ ,  $C_{2n}^n$  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  $C_{m+n}^m$ )

# Про множественные выравнивания

Для множественных выравниваний (число последовательностей 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 kodo 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 последовательностях. 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.

Якорь (в примере — длины 3)

# Как работает BLAST?

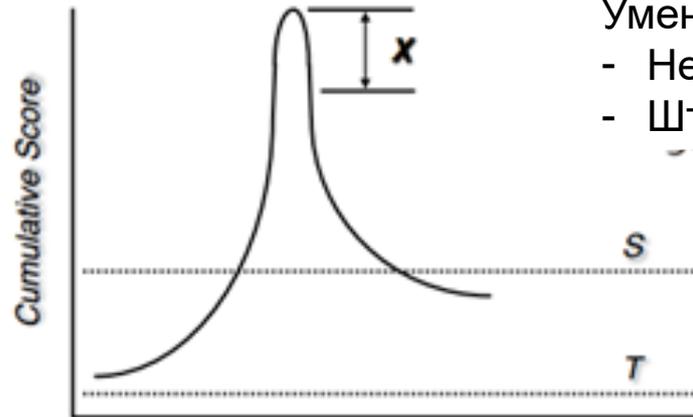
Query: GSQSLAALLNKCKT**PQG**QRLVNQWIKQPLMDKNRIEERLNLVEAFVED

Сходные слова

PQG	18	=7+5+6
PEG	15	
PRG	14	
PKG	14	
PNG	13	
PDG	13	
PHG	13	
PMG	13	
PSG	13	Порог на score
PQA	12	
PQN	12	
etc.		

Query: 325 SLAALLNKCKT**PQG**QRLVNQWIKQPLMDKNRIEERLNLVEA 365  
 +LA++L TP+G R++ +W+ +P+ D + ER + A  
 Sbjct: 290 TLASVLDCTVTPMGSRMLKRWLHMPVRDTRVLLERQQTIGA 330

Поиск коротких сходных слов в банке (затравок для выравниваний)



Уменьшение за счёт:

- Несовпадений
- Штрафов за гэпы

Расширение затравки

# Вопросы 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  $k$  ( $k = 2, 3, \dots, 6$ ), 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 встречается.

## **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,  $0 \leq q(a,b) \leq 1$ )

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 выравнивания?

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 результате 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

Protein BLAST: search protein da x +

blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastp&PAGE\_TYPE=BlastSearch&LINK\_LOC=blasthome

## Standard Protein BLAST

blastn blastp blastx tblastn tblastx

BLASTP programs search protein databases using a protein query. [more...](#) [Reset page](#) [Bookmark](#)

### Enter Query Sequence

Enter accession number(s), gi(s), or FASTA sequence(s) [Clear](#) Query subrange [Query subrange](#)

From

To

Or, upload file   [Clear](#)

Job Title

Enter a descriptive title for your BLAST search [Clear](#)

**Align two or more sequences** [Clear](#)

**BLAST results will be displayed in a new format by default**

You can always switch back to the Traditional Results page.



### Choose Search Set

Database  [Clear](#)

Organism   exclude

Enter organism common name, binomial, or tax id. Only 20 top taxa will be shown. [Clear](#)

Exclude  Models (XM/XP)  Non-redundant RefSeq proteins (WP)  Uncultured/environmental sample sequences

### Program Selection

Algorithm

- Quick BLASTP (Accelerated protein-protein BLAST)
- blastp (protein-protein BLAST)
- PSI-BLAST (Position-Specific Iterated BLAST)
- PHI-BLAST (Pattern Hit Initiated BLAST)
- DELTA-BLAST (Domain Enhanced Lookup Time Accelerated BLAST)

Choose a BLAST algorithm [Clear](#)

**BLAST** Search database nr using Blastp (protein-protein BLAST)

Show results in a new window

[+ Algorithm parameters](#)

20

# BLAST двух последовательностей

Protein BLAST: Align two or more

blastn blastp blastx tblastn tblastx

Align Sequences Protein BLAST

BLASTP programs search protein subjects using a protein query. [more...](#) [Reset page](#) [Bookmark](#)

**Enter Query Sequence**

Enter accession number(s), gi(s), or FASTA sequence(s) [Clear](#) Query subrange [?](#)

From

To

Or, upload file  Файл не выбран [?](#)

Job Title

Enter a descriptive title for your BLAST search [?](#)

Align two or more sequences [?](#)

**Enter Subject Sequence**

Enter accession number(s), gi(s), or FASTA sequence(s) [Clear](#) Subject subrange [?](#)

From

To

Or, upload file  Файл не выбран [?](#)

**Program Selection**

Algorithm  blastp (protein-protein BLAST)

Choose a BLAST algorithm [?](#)

**BLAST** Search protein sequence using Blastp (protein-protein BLAST)

Show results in a new window

[+ Algorithm parameters](#)

**BLAST results will be displayed in a new format by default**

You can always switch back to the Traditional Results page.

BLAST is a registered trademark of the National Library of Medicine

[Support center](#) [Mailing list](#)

21

# BLAST двух последовательностей

NCBI Blast: P0C6X7:RecName: Full x +

blast.ncbi.nlm.nih.gov/Blast.cgi

BLAST® » blastp suite-2sequences » results for RID-AFWJ6F46114

Home Recent Results Saved Strategies Help

[← Edit Search](#) Save Search Search Summary ▾

How to read this report? BLAST Help Videos Back to Traditional Results Page

**Job Title**  
P0C6X7:RecName: Full=Replicase polyprotein...  
RID  
[AFWJ6F46114](#) Search expires on 04-29 17:07 pm  
[Download All](#) ▾

**Program**  
Blast 2 sequences [Citation](#) ▾

**Query ID**  
[P0C6X7.1](#) (amino acid)

**Query Descr**  
RecName: Full=Replicase polyprotein 1ab; Short=pp1ab; A ...

**Query Length**  
7073

**Subject ID**  
[K9N7C7.1](#) (amino acid)

**Subject Descr**  
RecName: Full=Replicase polyprotein 1ab; Short=pp1ab; A ...

**Subject Length**  
7078

**Other reports**  
[Multiple alignment](#) [MSA viewer](#) ?

**Filter Results**

Percent Identity  to  E value  to  Query Coverage  to

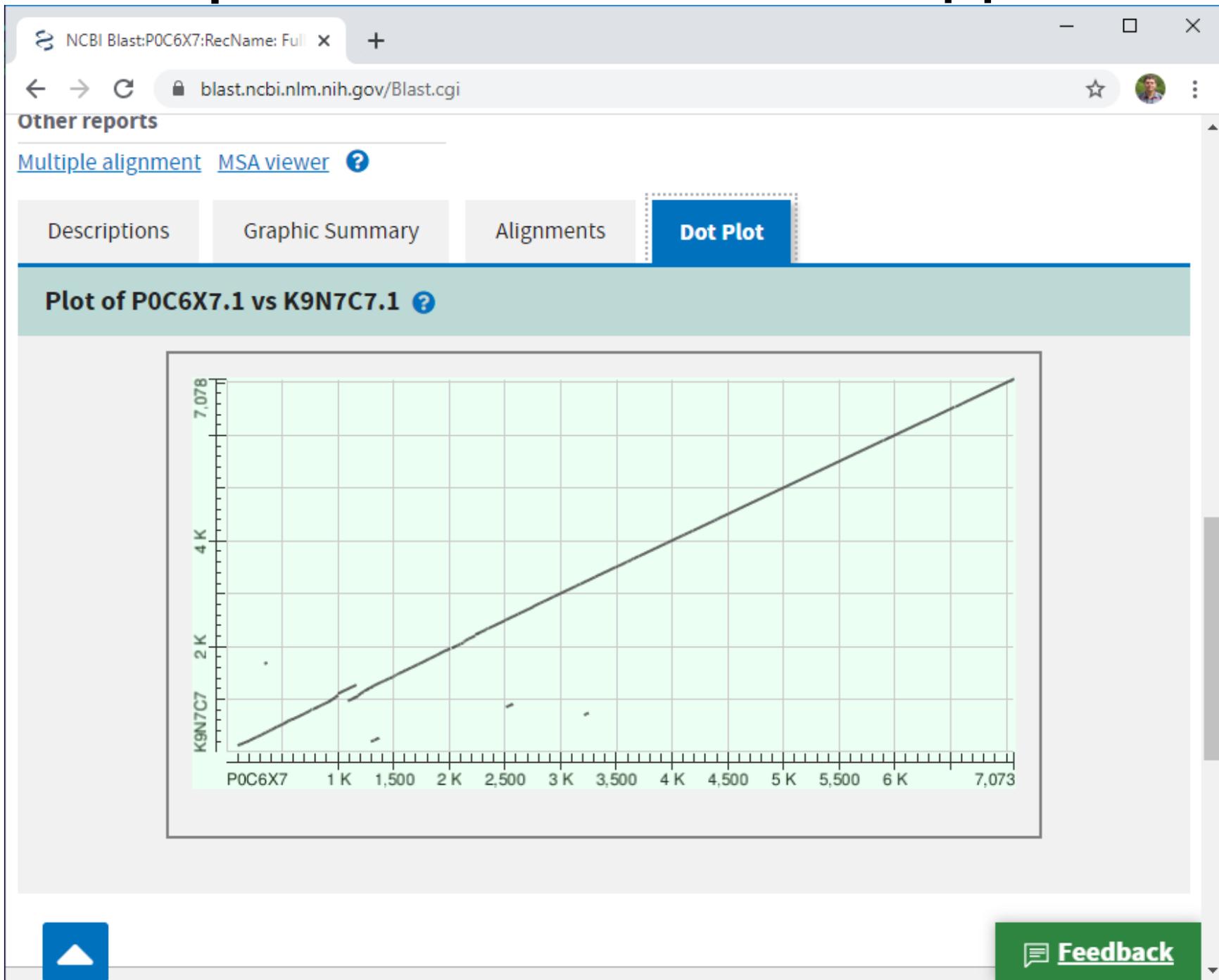
[Filter](#) [Reset](#)

[Descriptions](#) [Graphic Summary](#) [Alignments](#) [Dot Plot](#)

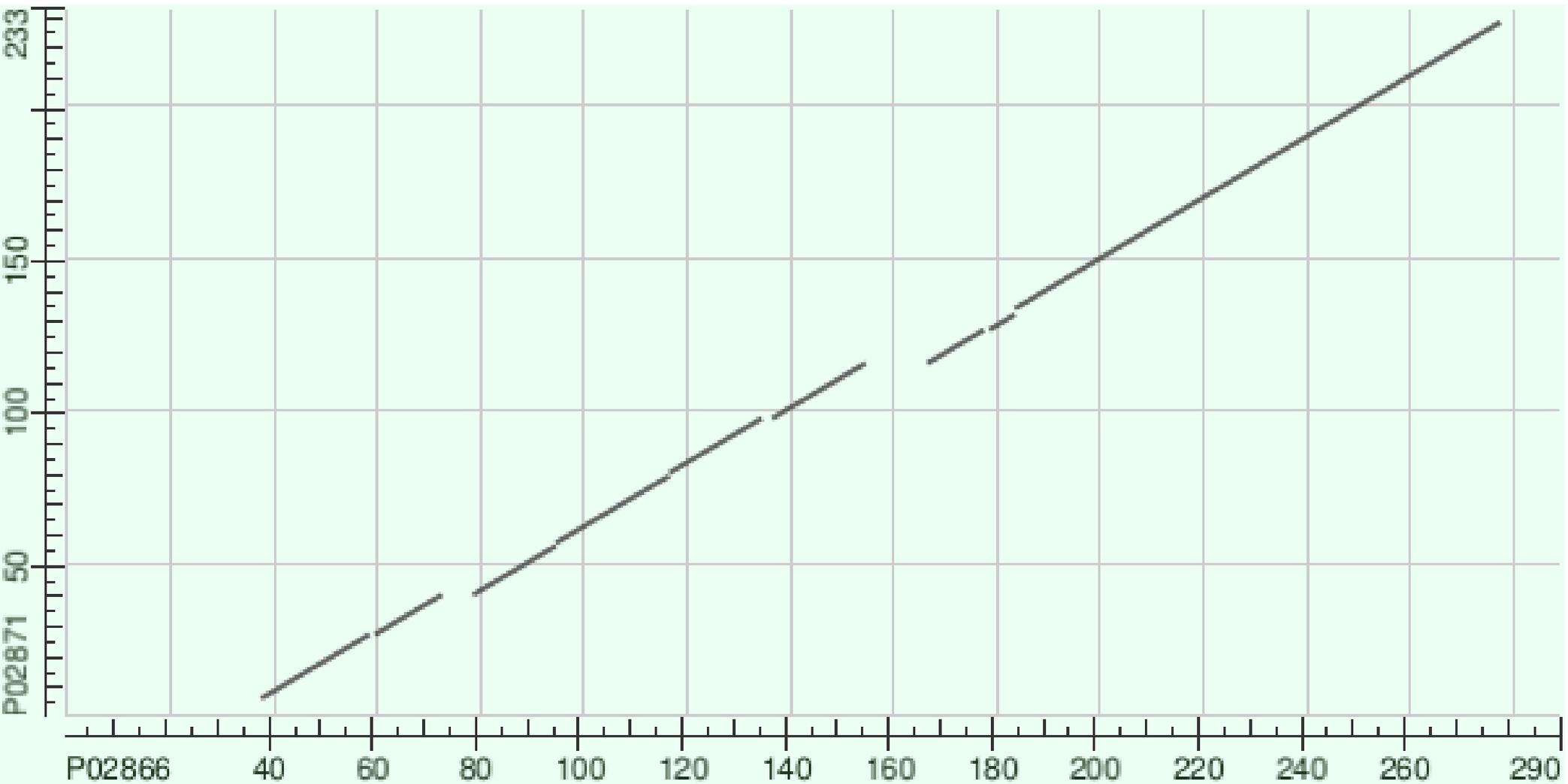
ences producing significant alignments

Download ▾ Manage Columns ▾ Show [Feedback](#)

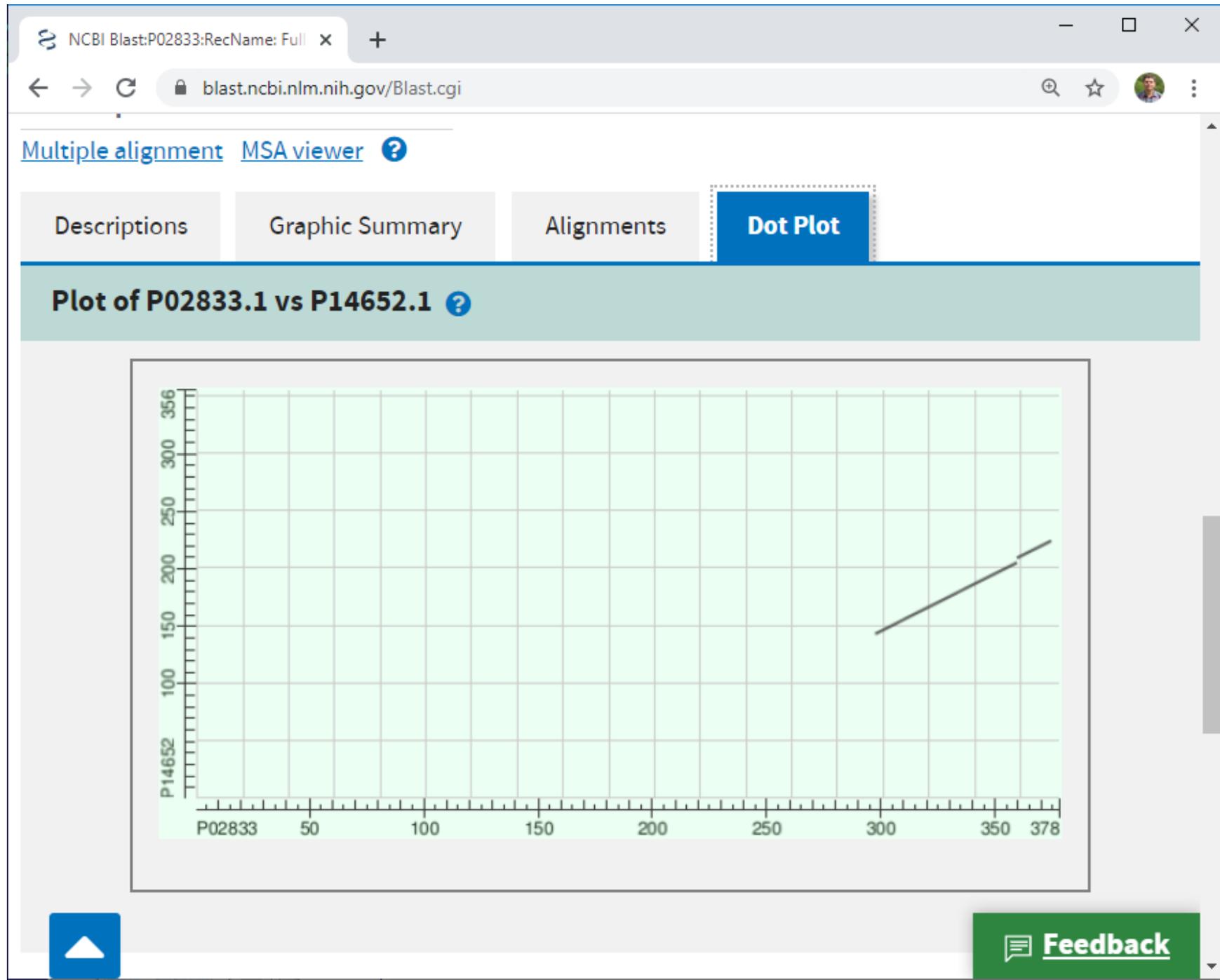
# Карта локального сходства



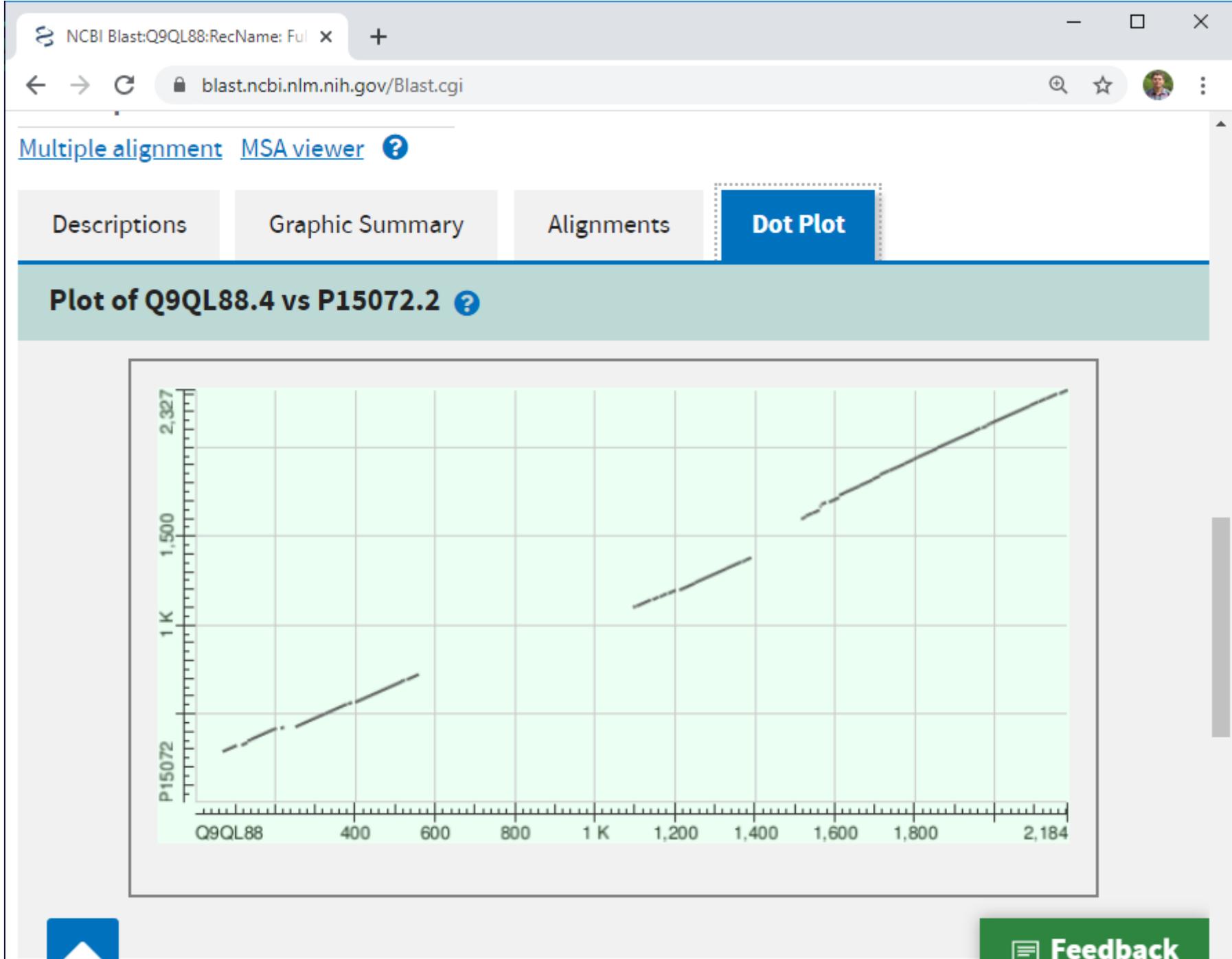
# Карта локального сходства



# Карта локального сходства



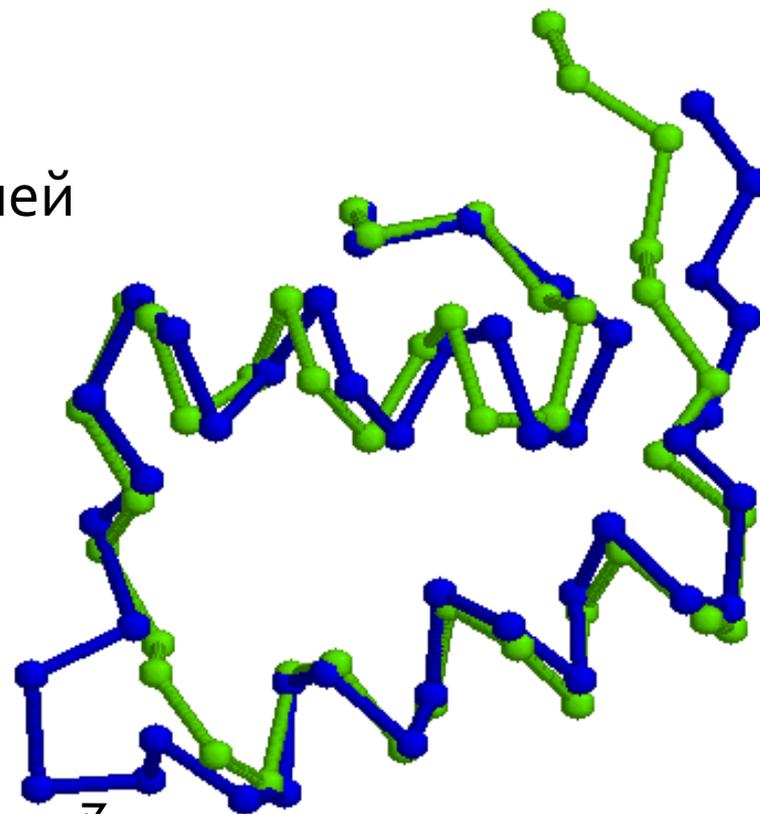
# Карта локального сходства



# 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 C $\alpha$ -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.

```
AFTGАНAYL  
AYS---AYM
```

```
AFTGАНAYL  
AY---SAYM
```

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