Выравнивание: алгоритмы

Дополнительная лекция для I курса ФББ

С.А. Спирин, 12 мая 2020

Эволюционный домен

- Что это такое?

Эволюционный домен

- Что это такое?

  Мой ответ: максимальный участок цепи белка, эволюционирующий только локально, то есть не замеченный в перестановках, потерях начального или конечного участка, крупных инсерциях.

- Дана последовательность белка. Что нужно, чтобы найти в ней эволюционные домены?

  (Предположим, что Pfam куда-то исчез; какие ещё данные, кроме самой последовательности белка, нужны и что с ними делать?)

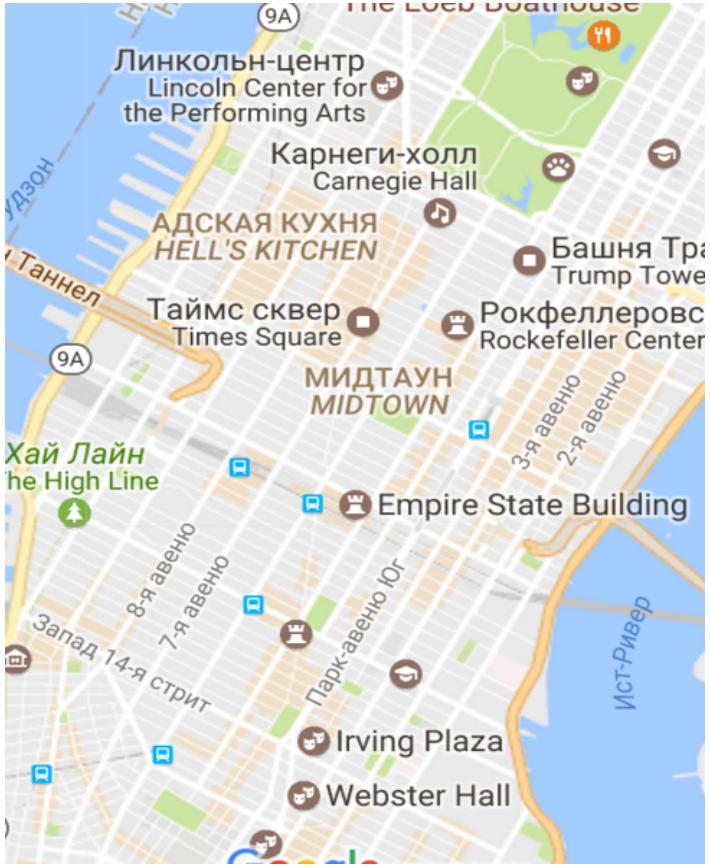
Динамическое программирование

- Как вы знаете, для двух последовательностей длин n и m существует C^{n}_{n+m} различных выравниваний.
  Для $n = m = 100$ это примерно 10^{60}.
- Как же алгоритм Нидлмана – Вунша справляется с этим за относительно короткое время?
- Ответ: динамическое программирование

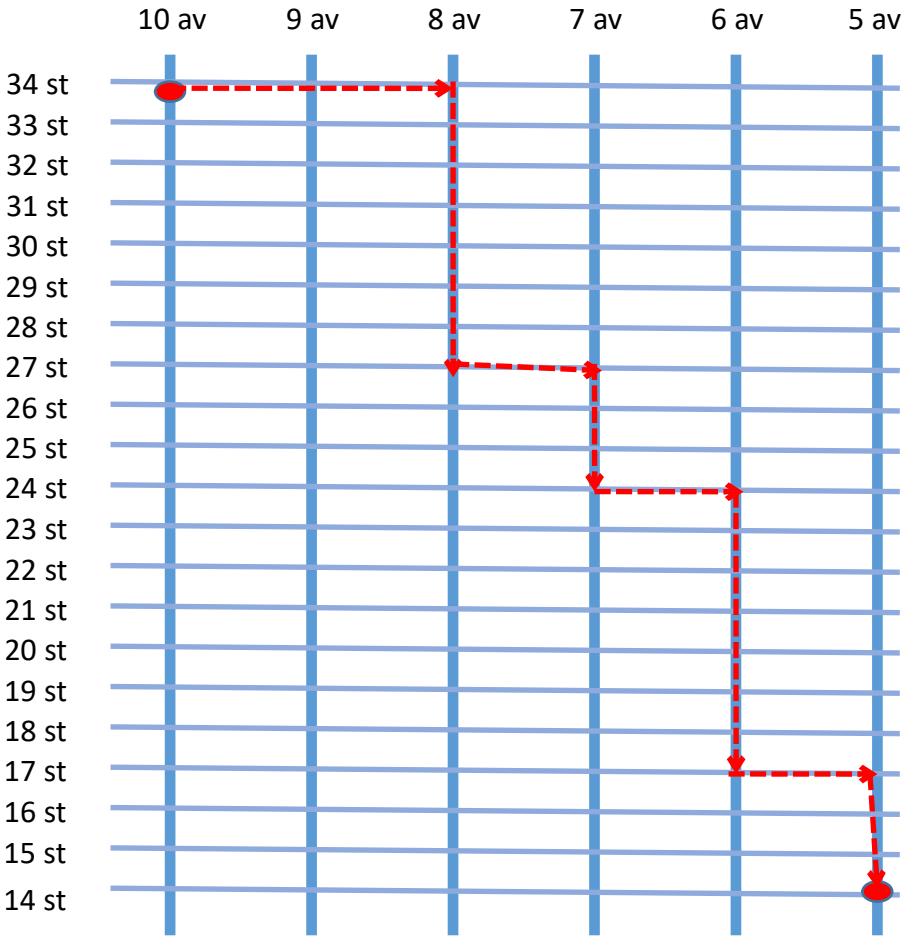
Динамическое программирование

- Идея динамического программирования состоит в сведении задачи к набору подзадач так, чтобы одни подзадачи использовали решения других.
  Ускорение достигается за счёт того, что каждая подзадача решается только один раз, в то время как при простом переборе их каждый раз приходится решать все.
- Простой пример — на следующем слайде

Манхэттен и его «формализация»



Мидтаун Манхэттен



Задача об оптимальном проезде

Пример задачи

- Пусть дана решётка размера $(n+1)\times(m+1)$, причём для всех соседних узлов решётки указана цена перехода

  (например, решётка — это улицы Манхэттена, а ценой может быть время с учётом пробок и светофоров)

- Задача: найти оптимальный путь из левого верхнего узла в правый нижний

  В случае времени «оптимальный» — значит с наименьшей суммой времён

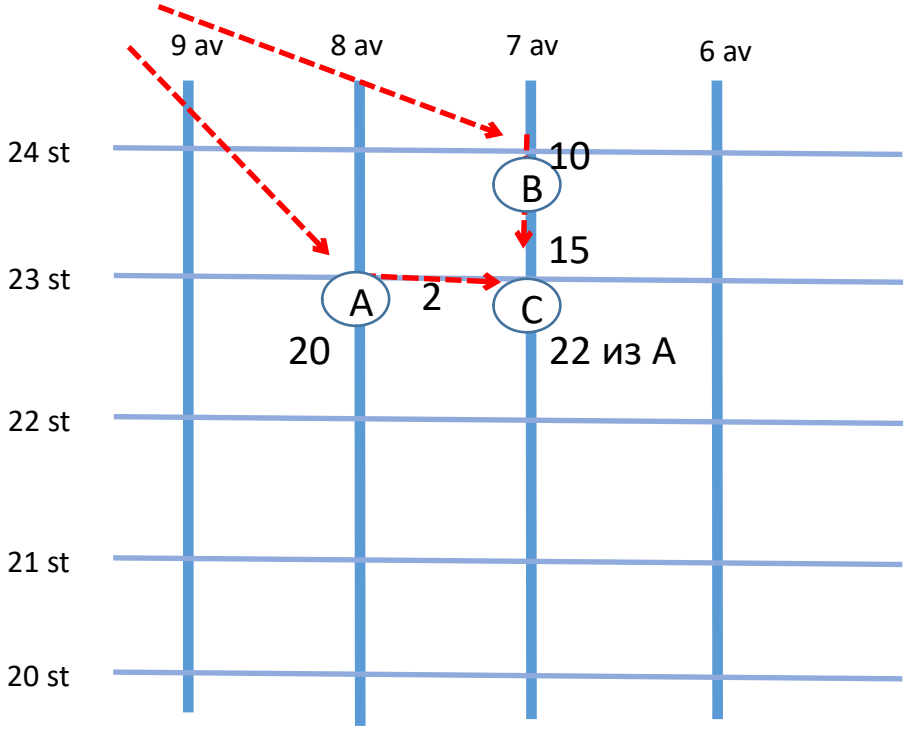
  Очень важно, что итоговая «цена» получается как сумма цен переходов!

  Это обстоятельство позволяет применить динамическое программирование.

Пример с решёткой: решение перебором

- Можно просто перебрать все возможные пути
- Сколько таких путей?
  - Мы должны проехать n+m отрезков, из них n с запада на восток и m с севера на юг
  - В последовательности проезжаемых отрезков любые n могут оказаться горизонтальными
  - Значит, различных путей C^{n}_{n+m} .

Основной шаг алгоритма

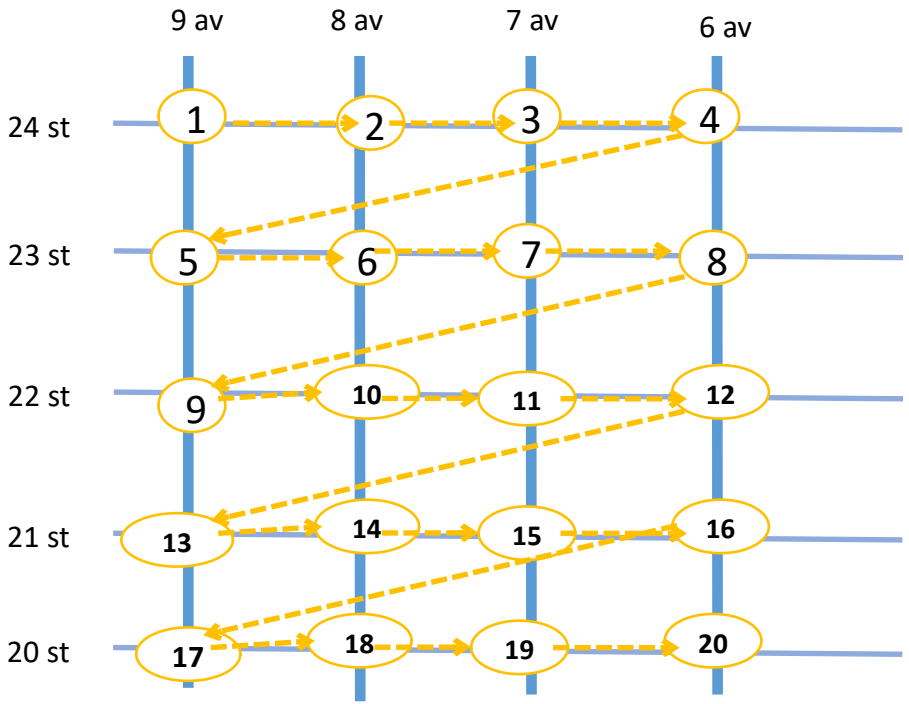


- На перекресток C можно попасть либо из A, либо из B.
- Если оптимальный путь в A занимает 20 мин, путь A-C – 2 мин, то путь в C через A занимает 22 мин
- Если оптимальный путь в B – 10 мин, B-C – 15 мин (пробка), то путь в C через B занимает 25 мин
- Значит, оптимальный путь до C занимает 22 мин, и это путь из A.
- Запомним это в C!

Идея в том, чтобы посчитать оптимальное время движения из точки старта **для всех** перекрёстков, которые могут попасться на пути!

Это легко сделать для самых северных перекрёстков.
Но и для остальных это занимает совсем немного времени!
В конце получаем оптимальное время для полного пути, всего за nm операций.

Оптимальный путь



Лучшее время вычислили.
А как ехать-то?

По ходу вычислений для каждого перекрестка C мы запоминали не только лучшее время, но и из какого соседнего перекрестка надо приехать: из A или из B (см. предыдущий слайд)

Оптимальный маршрут прокладывается от конечного пункта до начального по запомненным “стрелочкам”.

Сколько операций?

- Если надо проехать n кварталов направо и m – вниз, то имеем $(n+1)\times(m+1)$ перекрестков
- В каждом надо выполнить несколько операций: два сложения, одно сравнение, запомнить два числа. Требуется одно и то же число K операций в каждой вершине
- Значит, всего $K\cdot(n+1)\cdot(m+1)$ операций
- При $n = m = 50$ получаем $5 \cdot 51 \cdot 51 = 13\,005$ операций – пустяки для компьютера!
  (ср. с $C^{50}_{100} \approx 10^{30}$ — ноналлион вариантов, которые нужно было бы просмотреть при переборе)

А при чём тут выравнивание?

- Вместо пути теперь выравнивание
- Вместо времени — вес выравнивания
  - нужен не минимальный вес, а максимальный. Но главное осталось: вес выравнивания равен сумме весов по всем его позициям!
- Вместо перекрёстка у нас будет **выравнивание префиксов**

Префикс

- В языкознании это то же, что приставка:
  (в слове «пересдача» префикс «пере»)
- В комбинаторике (и теории алгоритмов) префиксом заданной последовательности длины n называется подпоследовательность от начала до k-ой буквы включительно (k может быть от нуля до n)
  (а последовательность обычно называется «слово»)
- Например, последовательность: ATGGCT
  Её префиксы: ∅, A, AT, ATG, ATGG, ATGGC, ATGGCT

Выравнивание префиксов

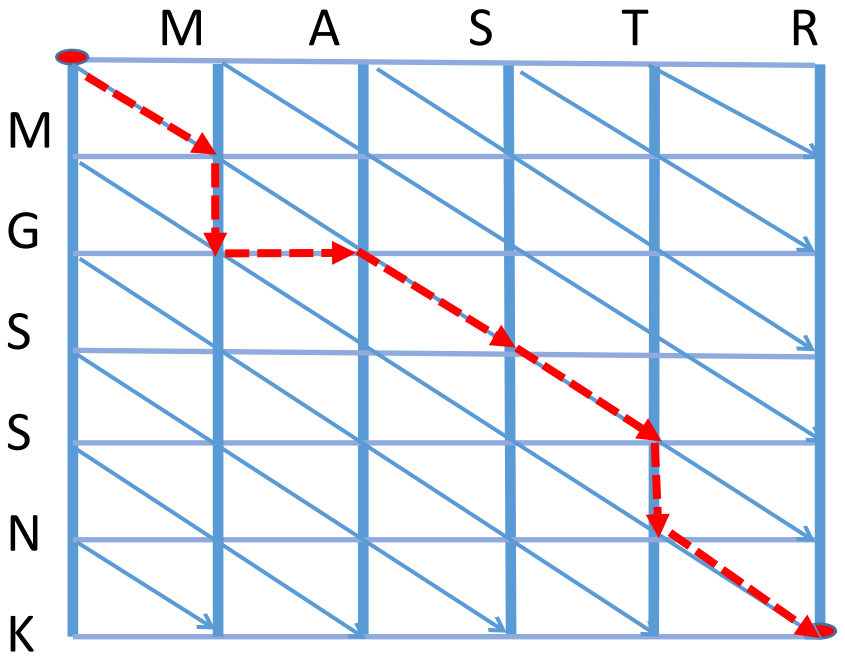
- Мы можем получить выравнивание k-го префикса первой последовательности с l-ым префиксом второй тремя способами:
  - взять **любое** выравнивание (k–1)-го префикса первой с (l–1)-ым префиксом второй и добавить колонку из k-ой буквы первой против l-ой буквы второй
  - взять любое выравнивание (k–1)-го префикса первой с l-ым префиксом второй и добавить колонку из k-ой буквы первой против **гэпа**
  - взять любое выравнивание k-го префикса первой с (l–1)-ым префиксом второй и добавить колонку из l-ой буквы второй против **гэпа**

```
M-AST-R     =   M-AST-  +  R
MG-SSNK         MG-SSN     K

M-AST--R    =   M-AST--  +  R
MG-SSNK-        MG-SSNK     -

M-AST-R-    =   M-AST-R  +  -
MG-SSN-K        MG-SSN-     K
```

Выравнивание префиксов

- Мы можем получить выравнивание k-го префикса первой последовательности с *l*-ым префиксом второй тремя способами:
  - взять **любое** выравнивание (k–1)-го префикса первой с (l–1)-ым префиксом второй и добавить колонку из k-ой буквы первой против l-ой буквы второй
  - взять любое выравнивание (k–1)-го префикса первой с l-ым префиксом второй и добавить колонку из k-ой буквы первой против **гэпа**
  - взять любое выравнивание k-го префикса первой с (l–1)-ым префиксом второй и добавить колонку из l-ой буквы второй против **гэпа**
- Вес каждого такого выравнивания получается из веса выравнивания меньших префиксов прибавлением веса сопоставления букв либо вычитанием штрафа за гэп.
  Из трёх вариантов можно выбрать оптимальный.
- Последовательно запомним для **всех** пар префиксов (начиная с пары пустых) оптимальный вес выравнивания и каким способом мы его получили. В конце получим оптимальный вес выравнивания последовательностей в целом.

Выравнивание — это почти путь по Манхэттену



M-AST-R

MG-SSNK

- Каждому перекрёстку соответствует пара префиксов
- Путь превращается в выравнивание согласно примеру
- Вес диагонали берется из матрицы весов
- Вес горизонтальной и вертикальной стрелки равен штрафу за гэп
- Выравнивание восстанавливается обратным проходом из правого нижнего угла в левый верхний.

Аффинные штрафы за гэпы

- Предыдущая картинка — для **линейных** штрафов за гэпы
- Чтобы учесть аффинные штрафы, нужно в каждый перекрёсток поместить не одну сущность (оптимальное выравнивание префиксов), а три:
  - лучшее из выравниваний, кончающихся сопоставлением букв
  - лучшее из выравниваний, кончающихся гэпом в первой последовательности
  - лучшее из выравниваний, кончающихся гэпом во второй последовательности

Алгоритм Смита – Уотермэна

- Отличается от алгоритма Нидлмана – Вунша следующими особенностями:
  - в узел ставится только **неотрицательный** вес
    (вес оптимального локального выравнивания двух префиксов не может быть отрицательным, потому что вес пустого выравнивания равен 0);
  - если оптимальный вес оказался отрицательным, заменяем его на 0, а вместо стрелки запоминаем, что ничто хорошее здесь не кончается
    (лучшее из локальных выравниваний, кончающихся здесь, пустое)
  - начинаем обратный проход не с правого нижнего угла, а с **лучшего** из локальных выравниваний префиксов (сначала находим максимальный вес по всем парам префиксов);
  - заканчиваем обратный проход, когда доходим до узла без стрелки.

Множественное выравнивание

- Пусть нам надо выровнять $N > 2$ последовательностей
- Можно было бы повторить фокус с префиксами (то есть динамическое программирование). Но:
  - на каждом шаге 2^N вариантов
  - число шагов равно произведению длин последовательностей
- Вывод: это слишком долго

Множественное выравнивание

- Для $N > 5$ используются исключительно эвристические алгоритмы — не гарантируют нахождение оптимального выравнивания (как бы ни понимать оптимальность)
- Основной подход: «прогрессивное выравнивание» — сведение множественного к многим парным

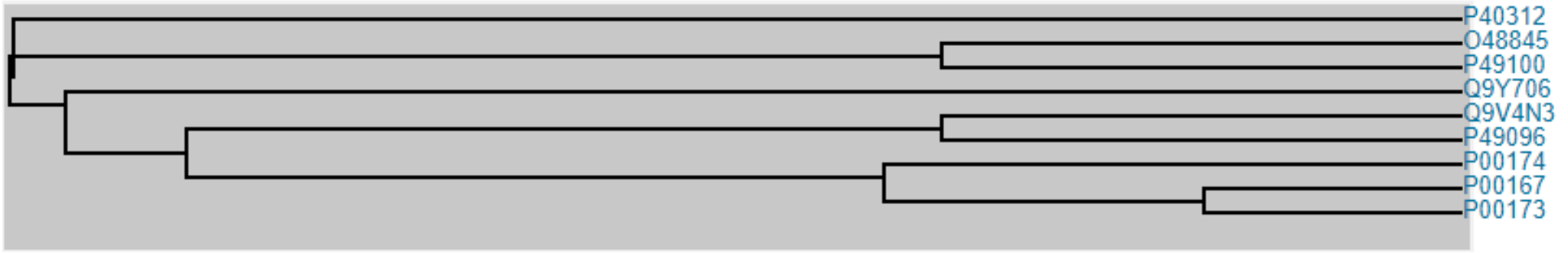
Прогрессивное множественное выравнивание

- Самый простой и наивный способ:
  - выровняем первую последовательность со второй
  - подравняем третью последовательность к готовому выравниванию (это можно сделать алгоритмом Нидлмана – Вунша, придумав разумный вес сопоставления колонки выравнивания и буквы)
  - затем четвёртую, пятую и т. д
- Это плохо тем, что ошибки начального этапа уже не исправить
- Можно по крайней мере их минимизировать, для этого первыми надо выравнивать самые близкие последовательности

Прогрессивное множественное выравнивание

- Сначала оцениваем расстояния между последовательностями
  - в старой программе ClustalW это делалось попарными выравниваниями всех со всеми
  - в более новых программах (Muscle, ClustalO, ...) это делается отдельными быстрыми алгоритмами
- Строим т. н. «направляющее дерево» — guide tree
- Выравниваем сначала пару самых близких последовательностей, а затем, шаг за шагом — пару самых близких из уже сделанных выравниваний

Направляющее дерево



Улучшение созданного выравнивания

- Делим множество последовательностей готового выравнивание на два подмножества
- Не меняя соответствие букв внутри частей, перевыравниваем между собой эти две части алгоритмом Нидлмана – Вунша
  (в каждую часть может быть вставлен только общий для всей части гэп)
- Повторяем, пока общее качество выравнивания не перестаёт улучшаться

Выдача программы Muscle

```
MUSCLE v3.8.31 by Robert C. Edgar
http://www.drive5.com/muscle
This software is donated to the public domain.
Please cite: Edgar, R.C. Nucleic Acids Res 32(5), 1792-97.

tmp 12 seqs, max length 335, avg  length 312
00:00:00      5 MB(0%)  Iter   1  100.00%  K-mer dist pass 1
00:00:00      5 MB(0%)  Iter   1  100.00%  K-mer dist pass 2
00:00:00      9 MB(0%)  Iter   1  100.00%  Align node
00:00:00      9 MB(0%)  Iter   1  100.00%  Root alignment
00:00:00      9 MB(0%)  Iter   2  100.00%  Root alignment
00:00:00      9 MB(0%)  Iter   3  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   4  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   5  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   5  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   6  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   7  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   8  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   8  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter   9  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  10  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  11  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  12  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  13  100.00%  Refine biparts
00:00:00      9 MB(0%)  Iter  13  100.00%  Refine biparts
```

Ещё про BLAST

- Отбор слов для поиска в индексной таблице
- Получение локального выравнивания из «якоря»
- Альтернативные форматы выдачи

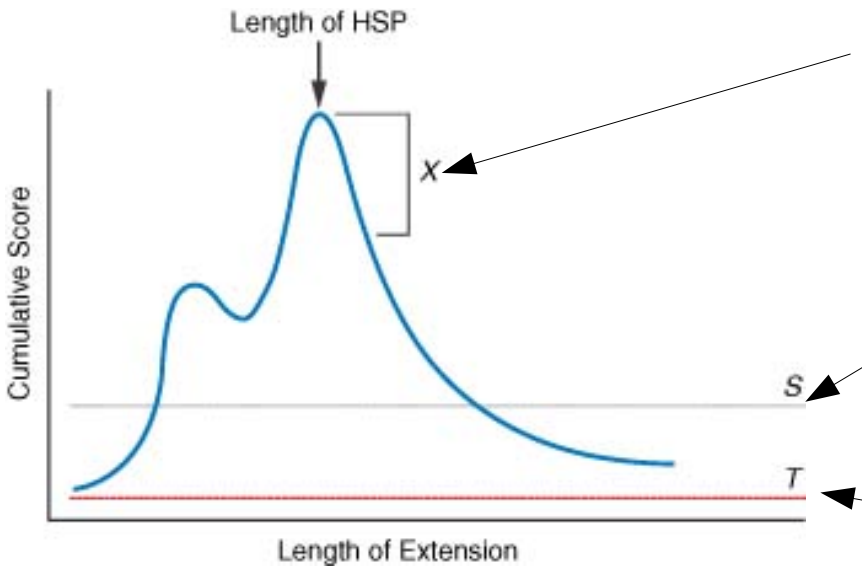
BLAST: отбор слов

- Два параметра:
  - длина слова
    (word_size, ≥2, по умолчанию 3)
  - порог на сходство слов
    (threshold, ≥0, по умолчанию 11)
- Берутся все слова из запроса (query)
  например, из aacddefg будут взяты (при длине слова 3):
  aac, acd, cdd, dde, def, dfg
- В индексах ищутся слова, имеющие сходство со словами из запроса на уровне не менее threshold

BLAST: от якоря к выравниванию

- Выравнивание начинает строиться, если в запросе есть пара слов на расстоянии, меньшем параметра window_size (по умолчанию 40), для которых нашлась пара сходных слов в одной банковской последовательности на том же расстоянии.
  В результате получаем два якоря — выравнивания длины word_size.
- Второй якорь расширяется без гэпов в обе стороны, пока вес не упадёт на заданную величину от максимально достигнутого (по умолчанию этот параметр xdrop_ungap = 7 бит)
- Если максимально достигнутый вес больше 22 бит, то соответствующее выравнивание расширяется уже с гэпами (аналогично алгоритму Нидлмана – Вунша). Расширение продолжается, пока вес не упадёт ниже максимально достигнутого на величину, большую xdrop_gap, по умолчанию 15 бит

BLAST: расширение якоря



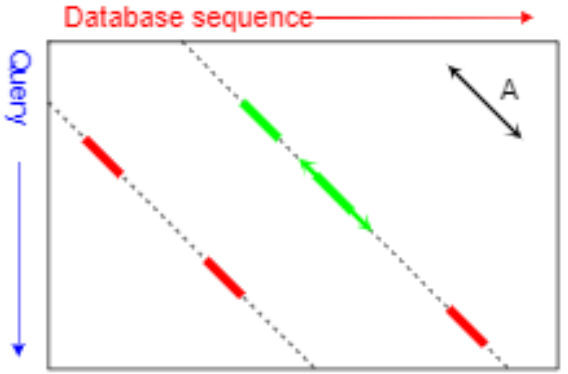
xdrop_ungap, по умолчанию X =7

Минимальный вес хита, по умолчанию S = 22

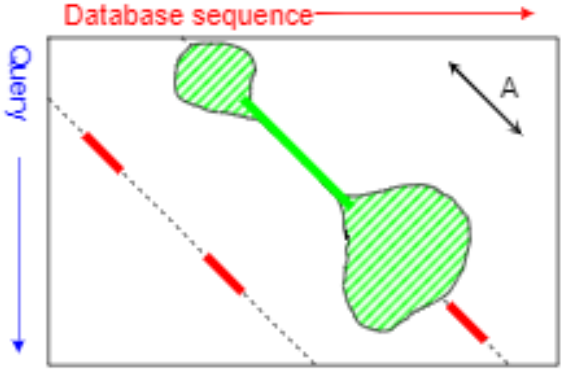
Минимальный вес якоря, по умолчанию T = 11

Это схема расширения в одну сторону; после того, как максимальное значение найдено, точно так же расширяем в другую.

Indexing for Blast (3)



Ungapped extension if:
2 "Hits" are on the same diagonal but
at a distance less than A



Extension using dynamic programming
limited to a restricted region
limited through a score drop-off
threshold

LR, Bazel October 2006

https://docplayer.net/15013198-Databases-indexation.html
Автор: Laurent Falquet, SIB

BLAST: роль длины слова
(мой эксперимент)

- Вход: последовательность из 466 остатков
- NCBI BLAST (https://blast.ncbi.nlm.nih.gov/)
- Область поиска: Swiss-Prot, белки из бактерий
- Параметры, кроме "Word Size", по умолчанию.
  В частности, порог E-value = 10
- W = 6
  - Найдено 16 последовательностей, в них 18 находок
  - 8 находок с E < 0,001
  - Время работы сервиса NCBI – менее одной минуты
- W = 2
  - Найдено 69 последовательностей, в них 75 находок
  - 12 находок с E < 0,001
  - Время работы сервиса NCBI – около 35 мин

BLAST: варианты формата выходного файла

```
-outfmt <String>
  alignment view options:
    0 = Pairwise,
    1 = Query-anchored showing identities,
    2 = Query-anchored no identities,
    3 = Flat query-anchored showing identities,
    4 = Flat query-anchored no identities,
    5 = BLAST XML,
    6 = Tabular,
    7 = Tabular with comment lines,
    8 = Seqalign (Text ASN.1),
    9 = Seqalign (Binary ASN.1),
   10 = Comma-separated values,
   11 = BLAST archive (ASN.1),
   12 = Seqalign (JSON),
   13 = Multiple-file BLAST JSON,
   14 = Multiple-file BLAST XML2,
   15 = Single-file BLAST JSON,
   16 = Single-file BLAST XML2,
   18 = Organism Report
```

0–4 — чтобы смотреть глазами

5–12 — чтобы парсить программами.

6, 7 и 10 можно импортировать в электронные таблицы