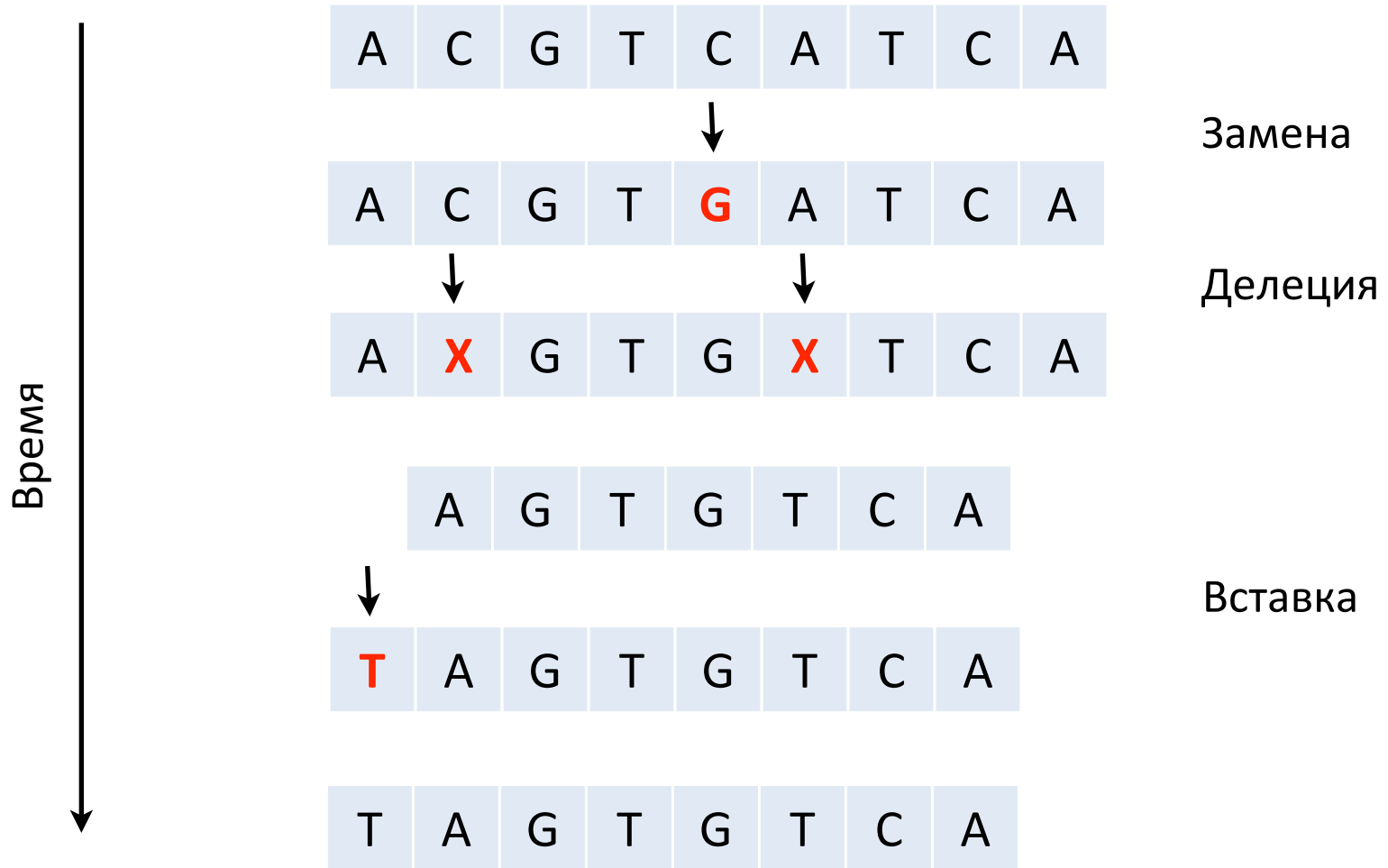


Выравнивание биологических последовательностей

Изменение геномов с течением времени



Задача выравнивания: определение редакционного расстояния

A	C	G	T	C	A	T	C	A
---	---	---	---	---	---	---	---	---

?

Формальная постановка задачи:
определение редакционного
расстояния - минимального
количества элементарных
преобразований (замен, вставок,
делеций), переводящих одну
последовательность в другую

T	A	G	T	G	T	C	A
---	---	---	---	---	---	---	---

Что необходимо для вычисления оптимального выравнивания?

S1 A C G T C A T C A

S2 T A G T G T C A

- Весовая функция (scoring function)
 - Вес выравнивания = стоимость редактирования S1 в S2
 - Стоимость замены, вставки, делеции
 - Бонус за совпадение букв
- Алгоритм нахождения оптимального выравнивания
 - Перебор?

Весовая функция (scoring function)

-	A	C	G	T	C	A	T	C	A
▲	■	▼	■	■	×	▼	■	■	■
T	A	-	G	T	G	-	T	C	A

Тип позиции		Стоимость
Совпадение	■	$+m$
Несовпадение	×	$-s$
Разрыв	▲ ▼	$-d$

Вес выравнивания (Score) =
(число совпадений) $\times m$ – (число несовпадений) $\times s$ – (число разрывов) $\times d$

Матричное представление выравнивания

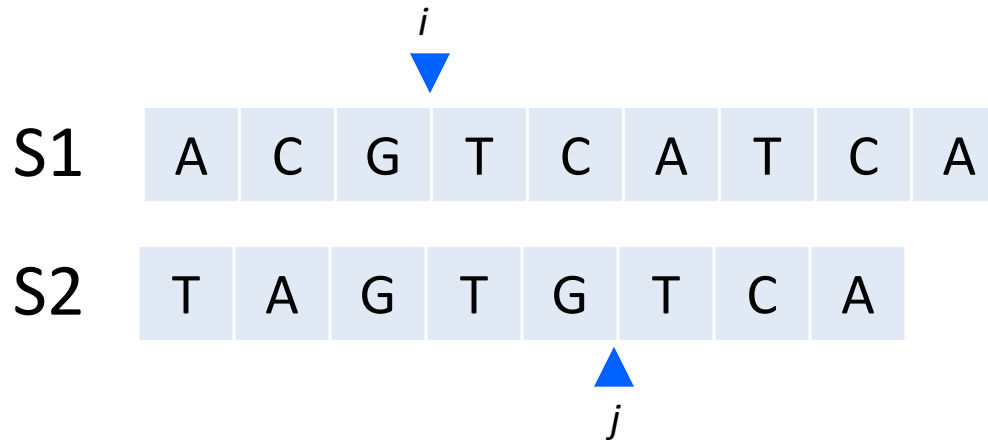
–	A	C	G	T	C	A	T	C	A
▲	■	▼	■	■	✗	▼	■	■	■
T	A	–	G	T	G	–	T	C	A

		A	C	G	T	C	A	T	C	A
	begin									
T	▲									
A		■	▼							
G				■						
T					■					
G						✗	▼			
T								■		
C									■	
A										■

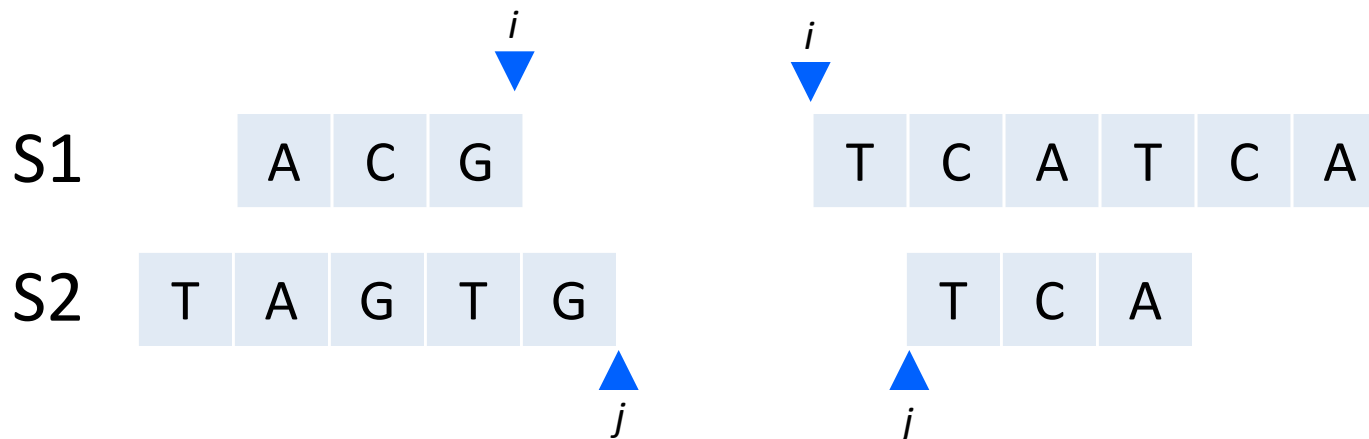
Цель: найти оптимальный путь по матрице от точки начала до точки окончания.

end

Вес (score) выравнивания аддитивен



Для заданного разделения (i,j) вес оптимального выравнивания есть:
вес оптимального выравнивания между $S1[1,i]$ и $S2[1,j]$ +
вес оптимального выравнивания между $S1[i,n]$ и $S2[j,m]$



Динамическое программирование

- Для данной задачи существует только конечное число подзадач?
 - Да. Имеем $n \times m$ подзадач выравниваний $S1[1,i]$ с $S2[1,j]$.
- Первоначальная задача является одной из подзадач?
 - Да. Выравнивание $S1[1,n]$ с $S2[1,m]$.
- Каждая подзадача решается на основе решений более мелких подзадач?
 - Да. (покажем далее).

Вывод: мы можем использовать динамическое программирование

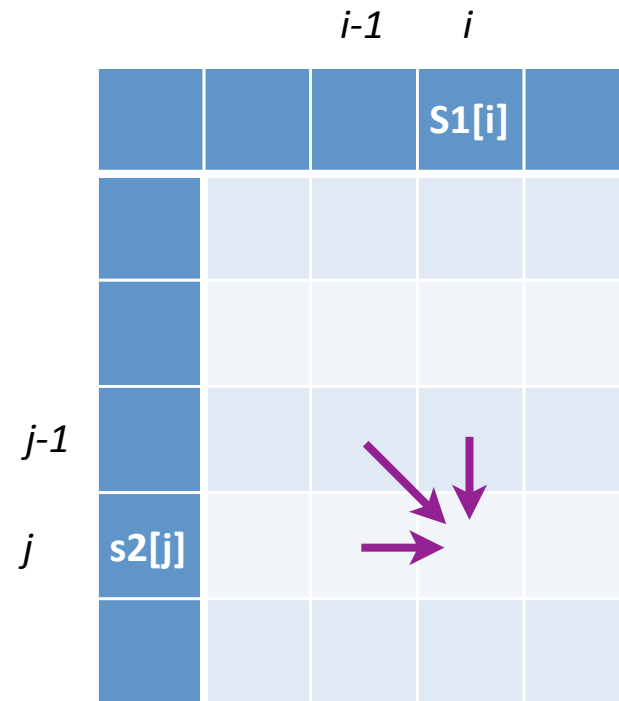
Поиск оптимального выравнивания $S1[1..l]$ с $S2[1..l]$

- Мы можем попасть в точку (i,j) только из трех позиций: $(i-1,j-1)$, $(i-1,j)$ и $(i,j-1)$.
- Зная веса F оптимальных выравниваний $F(i-1,j-1)$, $F(i-1,j)$ и $F(i,j-1)$, вес оптимального выравнивания $F(i,j)$ определяется как:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + g(i,j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

где

$$g(i,j) = \begin{cases} m, & \text{if } S1[i] = S2[j] \\ -s, & \text{if } S1[i] \neq S2[j] \end{cases}$$



Пример

S1 A G T A

$m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
A					
T					
A					

Пример

S1 A G T A $m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1				
T	-2				
A	-3				

Пример

S1 A G T A

$m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1				
T	-2				
A	-3				

Пример

S1 A G T A

$m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1				
T	-2				
A	-3				

$$F(1,1) = \max \begin{cases} F(0,0) + g(A,A) \\ F(0,1) - d \\ F(1,0) - d \end{cases} =$$

$$= \max \begin{cases} 0 + 1 \\ -1 - 1 = 1 \\ -1 - 1 \end{cases}$$

Пример

S1 A G T A $m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1	1			
T	-2				
A	-3				

Пример

S1 A G T A

$m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1	1			
T	-2				
A	-3				

Пример

S1 A G T A $m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1	1	0	-1	-2
T	-2	0	0	1	0
A	-3	-1	-1	0	2

Пример

S1 A G T A $m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1	1	0	-1	-2
T	-2	0	0	1	0
A	-3	-1	-1	0	2

A
A

Пример

S1 A G T A $m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1	1	0	-1	-2
T	-2	0	0	1	0
A	-3	-1	-1	0	2

T	A
T	A

Пример

S1 A G T A $m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1	1	0	-1	-2
T	-2	0	0	1	0
A	-3	-1	-1	0	2

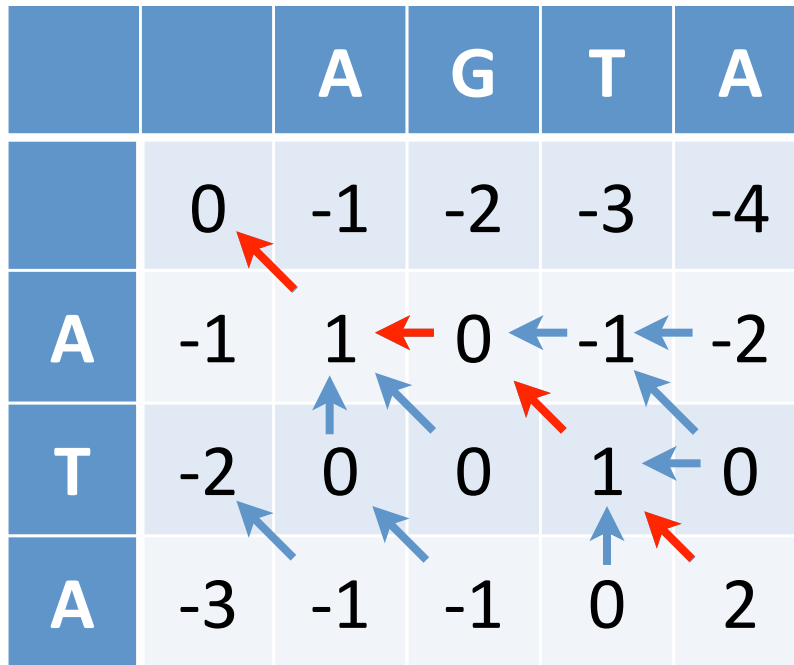
G	T	A
-	T	A

Пример

S1 A G T A $m=1, s=-1, d=-1$

S2 A T A

		A	G	T	A
	0	-1	-2	-3	-4
A	-1	1	0	-1	-2
T	-2	0	0	1	0
A	-3	-1	-1	0	2



A	G	T	A
A	-	T	A

Глобальное выравнивание (алгоритм Нидлмана-Вунша)

1. Инициализация.

- a. $F(0, 0) = 0$
- b. $F(0, j) = -j \times d$
- c. $F(i, 0) = -i \times d$

2. Основной цикл. Заполнение матрицы

- a. For each $i = 1 \dots M$
For each $j = 1 \dots N$

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + g(s1[i], s2[j]) & \text{[case 1]} \\ F(i-1, j) - d & \text{[case 2]} \\ F(i, j-1) - d & \text{[case 3]} \end{cases}$$

$$Ptr(i, j) = \begin{cases} \text{DIAG,} & \text{if [case 1]} \\ \text{LEFT,} & \text{if [case 2]} \\ \text{UP,} & \text{if [case 3]} \end{cases}$$

- 3. Завершение. $F(M, N)$ - оптимальный вес, выравнивание извлекается из $Ptr(M, N)$ процедурой обратного прохода

Вычислительная сложность алгоритма Нидлмана-Вунша

- Временная сложность (количество операций) - $O(N \times M)$
- Пространственная сложность (объем памяти) - $O(N \times M)$

Локальное выравнивание

Даны две последовательности

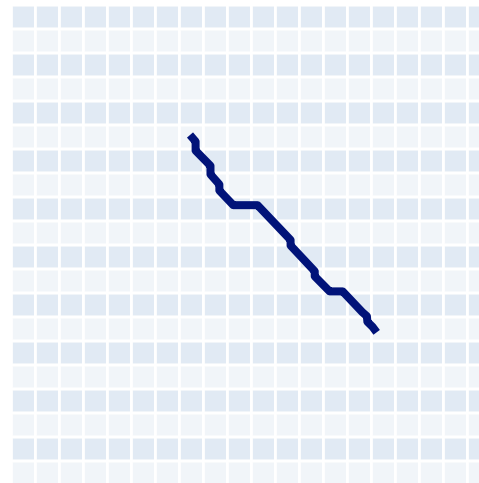
$$S1 = S1_1 \dots S1_M,$$

$$S2 = S2_1 \dots S2_N$$

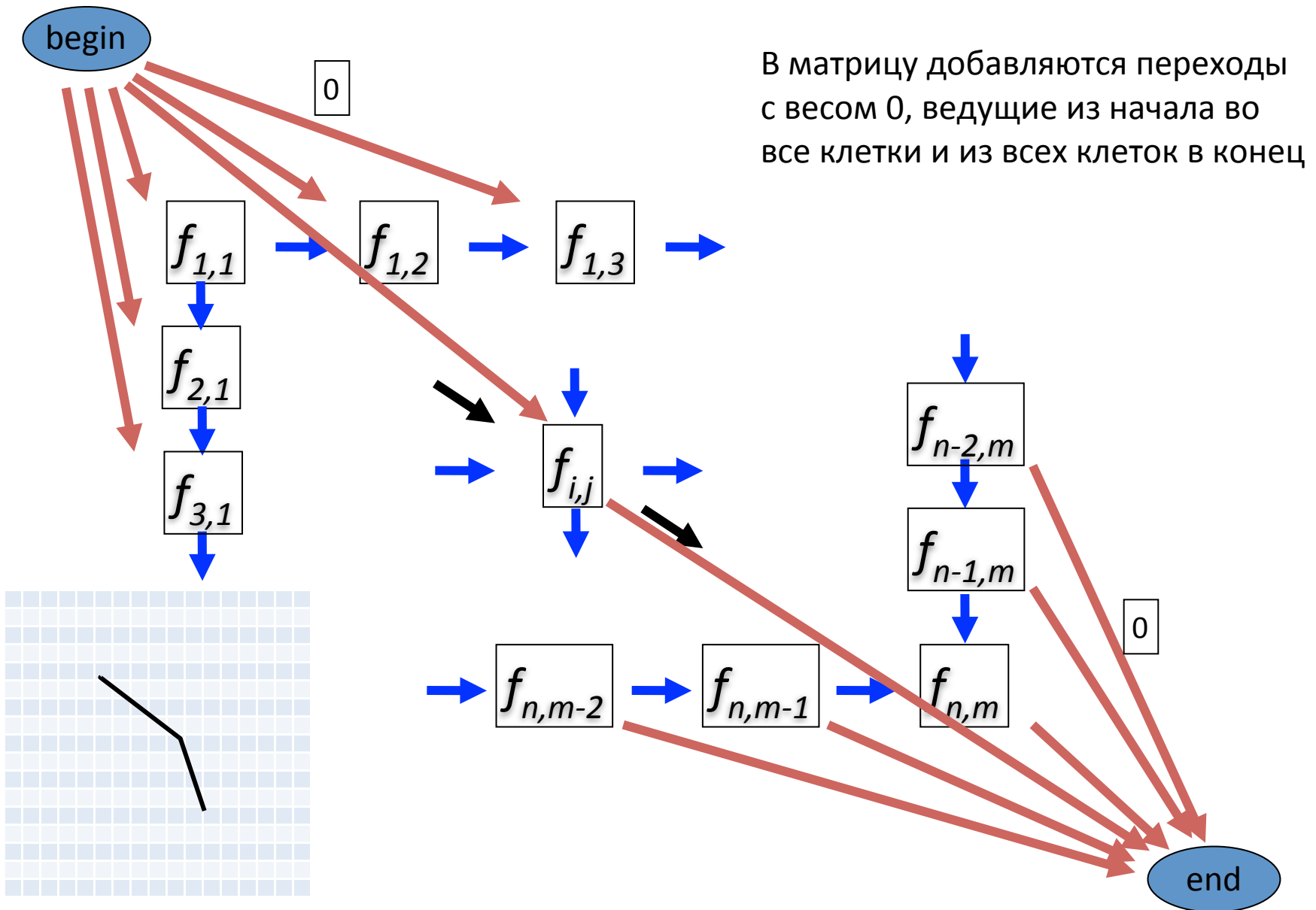
Найти подстроки $S1'$, $S2'$,
имеющие максимальный вес
выравнивания

$S1 = \text{aaaacc} \boxed{\text{cccgggg}} \text{tta}$

$S2 = \text{tt} \boxed{\text{cccgggg}} \text{aaccaacc}$



Алгоритм Смита-Ватермана



Алгоритм локального выравнивания Смита-Ватермана

По сути - модификация алгоритма Нидлмана-Вунша:

Инициализация: $F(0, j) = F(i, 0) = 0$

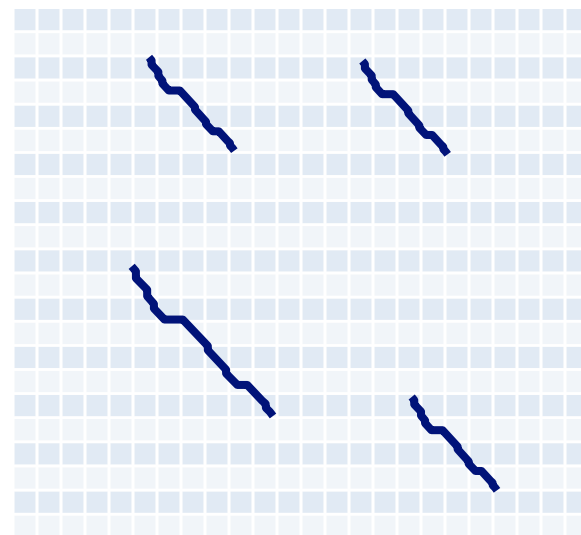
Итерация основного цикла:

$$F(i, j) = \max \begin{cases} 0 \\ F(i-1, j) - d \\ F(i, j-1) - d \\ F(i-1, j-1) + g(s1[i], s2[j]) \end{cases}$$

Завершение:

$$F_{\text{OPT}} = \max_{i,j} F(i, j)$$

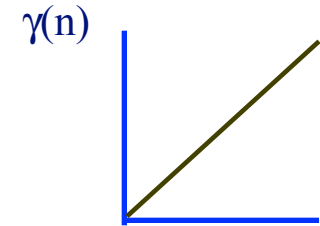
Найти F_{OPT} и выполнить процедуру обратного прохода



Оптимизация штрафов за разрывы

Текущая модель:

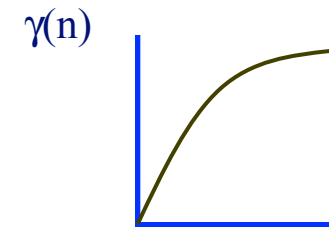
За разрыв длины n
следует штраф $n \times d$



Известно, что разрывы чаще всего встречаются сериями

Выпуклая штрафная функция:

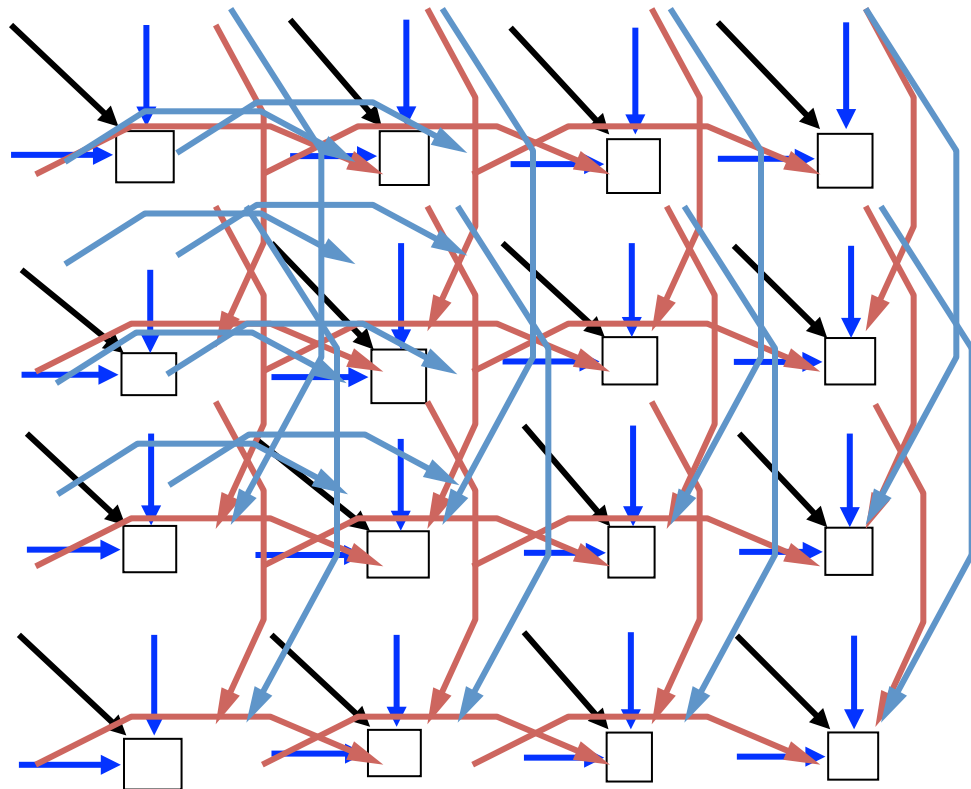
$\gamma(n)$:
for all n , $\gamma(n + 1) - \gamma(n) \leq \gamma(n) - \gamma(n - 1)$



Более общая зависимость штрафа за делецию от величины делеции

Теперь надо просматривать все возможные варианты делеций. Поэтому в каждую клетку возможно попасть не 3-мя типами переходов, а примерно $(n+m)/2$ - путями, где n, m – длины последовательностей

Поэтому время работы алгоритма становится кубическим: $O(nm(n+m))$

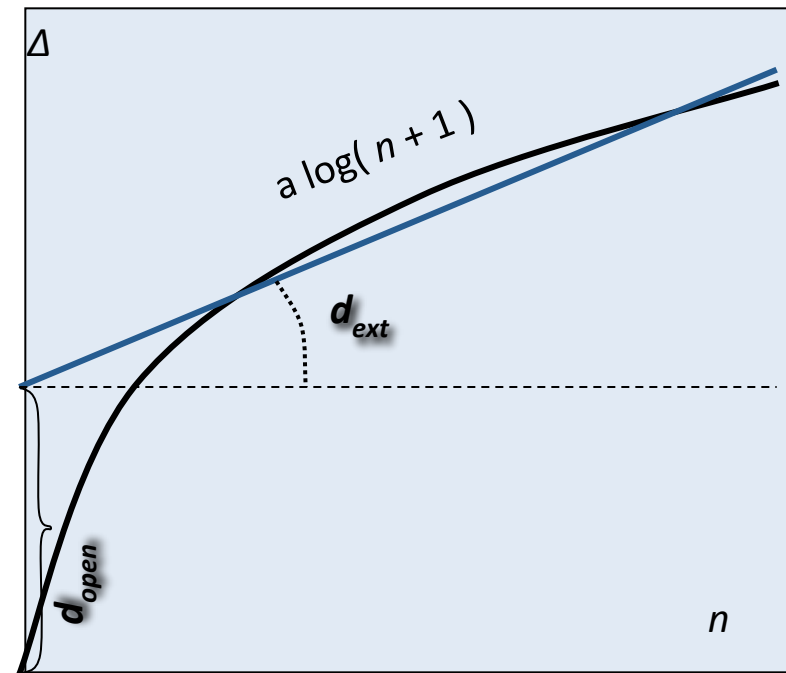


Аффинные штрафы за разрывы

- Вместо логарифмической зависимости используют зависимость вида:

$$\Delta (n) = d_{open} + (n-1) d_{ext}$$

- d_{open} – штраф за открытие делеции
- d_{ext} – штраф за удлинение делеции



Параметры выравнивания

$$\begin{array}{l} X = \{ \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline C & A & C & G & T & C & A & T & C & A \\ \hline \end{array} \} \\ Y = \{ \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline T & A & T & G & T & G & G & T & C & A \\ \hline \end{array} \} \end{array}$$

Случайная модель R

(независимые последовательности):

$$P(x, y | R) = \prod_i q_{x_i} \prod_i q_{y_i}$$

Родственная модель M:

$$P(x, y | M) = \prod_i p_{x_i y_i}$$

Определим вес выравнивания как отношение вероятностей двух моделей:

$$\frac{P(x, y | M)}{P(x, y | R)} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} \prod_i q_{y_i}} = \prod_i \frac{p_{x_i y_i}}{q_{x_i} q_{y_i}}$$

Для получения аддитивной весовой функции возьмем логарифм:

$$s(x_i, y_i) = \log \left(\frac{p_{x_i y_i}}{q_{x_i} q_{y_i}} \right)$$

Матрицы замен

- Элемент матрицы для пары остатков a и b определяется из вероятностной модели как:

$$s(a,b) = \log\left(\frac{p_{ab}}{q_a q_b}\right)$$

- Как вычислить p_{ab} - вероятность того, что остатки a и b происходят от общего предкового остатка? -
 - Можно использовать выравнивания составленные экспертами
- Популярные наборы матрицы замен для белковых последовательностей:
 - матрицы BLOSUM [Henikoff & Henikoff, 1992]
 - матрицы PAM [Dayhoff *et al.*, 1978]

Матрицы BLOSUM

- Источник - база данных BLOCKS (*Henikoff & Henikoff*) – безделеционные фрагменты множественных выравниваний (выравнивания получены *экспертом*)
- Блоки выравниваний разбивались на кластеры, соответствующие проценту консервативности:
 - 45% идентичности (матрица BLOSUM45)
 - 50% идентичности (матрица BLOSUM50)
 - 62% идентичности (матрица BLOSUM62)
- A_{ab} - частота выравнивания остатка a с остатком b в соответствующем кластере

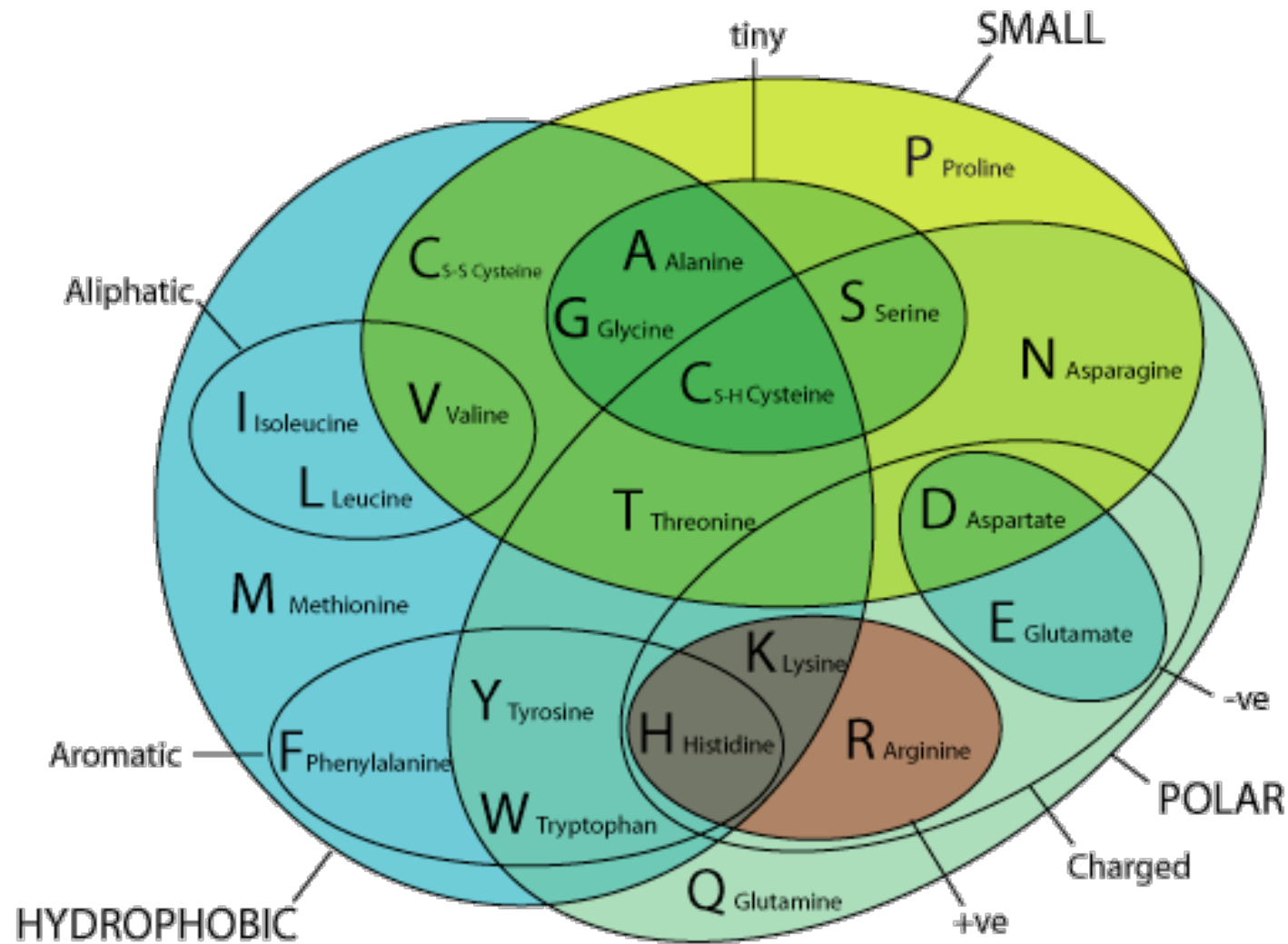
$$p_{ab} = \frac{A_{ab}}{\sum_{c,d} A_{cd}}$$

$$q_a = \frac{\sum_b A_{ab}}{\sum_{c,d} A_{cd}}$$

Матрица BLOSUM62

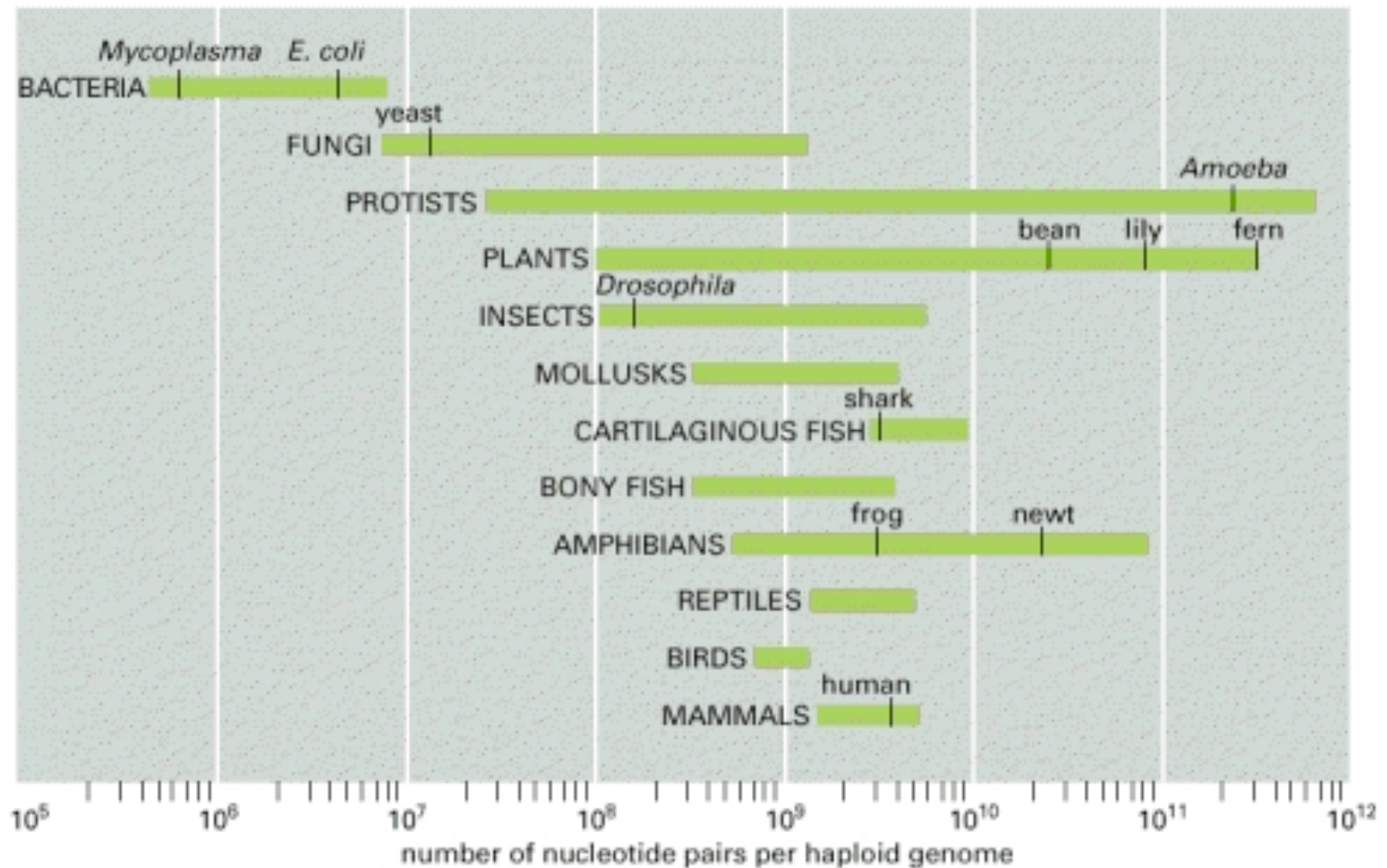
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

Свойства аминокислот

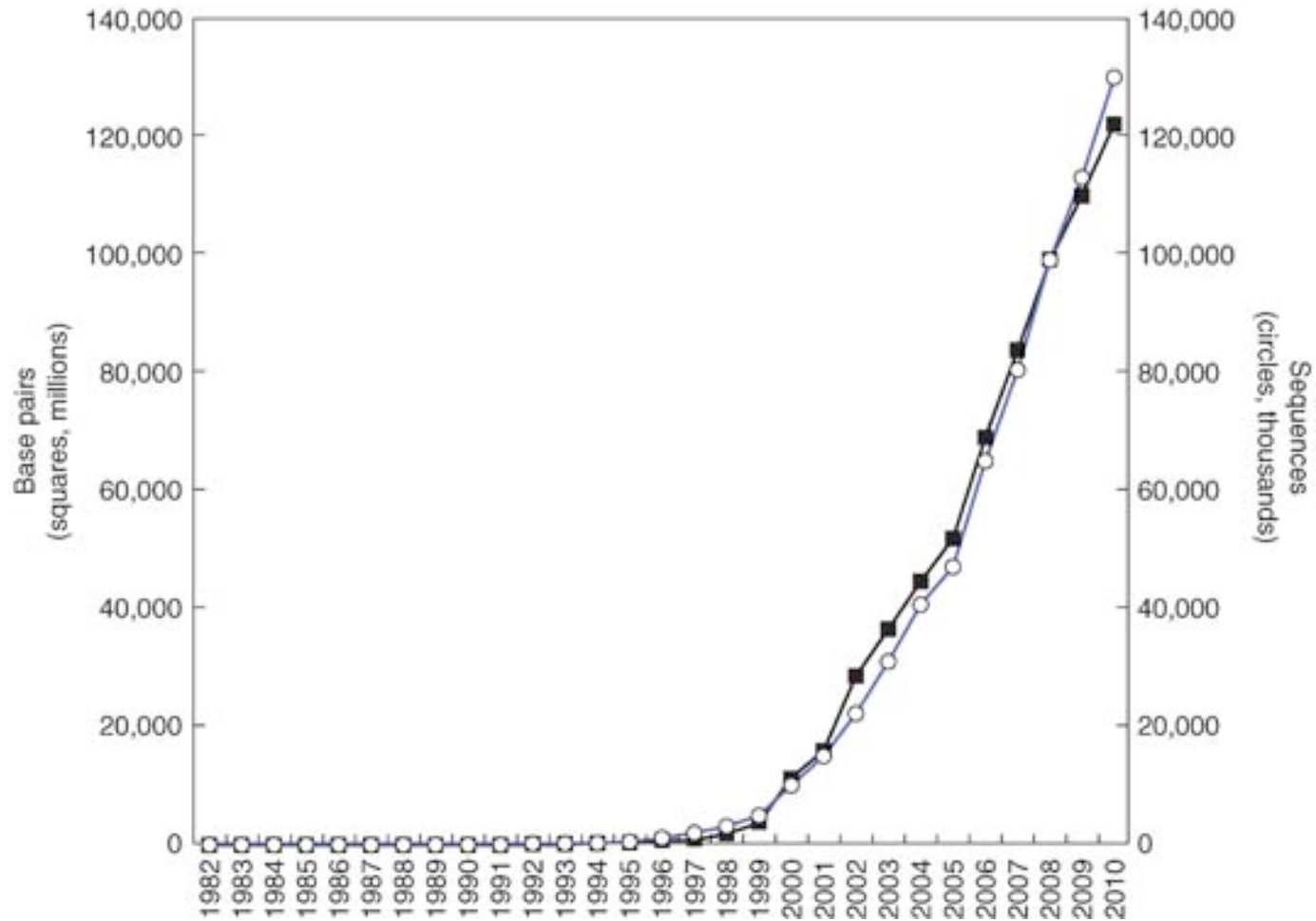


Алгоритм BLAST

Размеры геномов

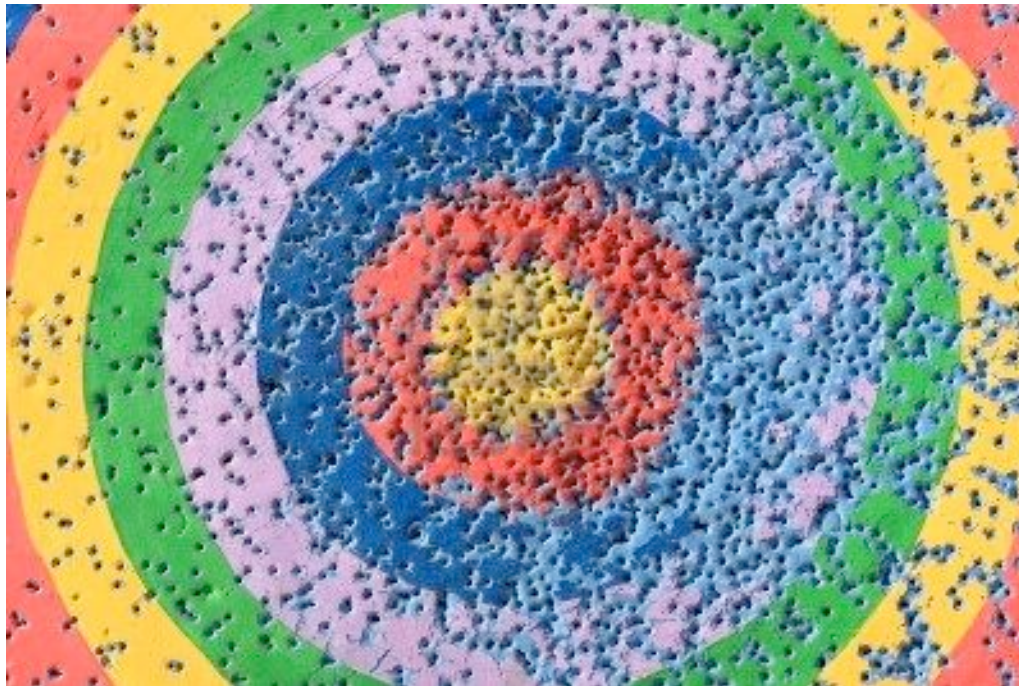


GenBank: экспоненциальный рост объема данных



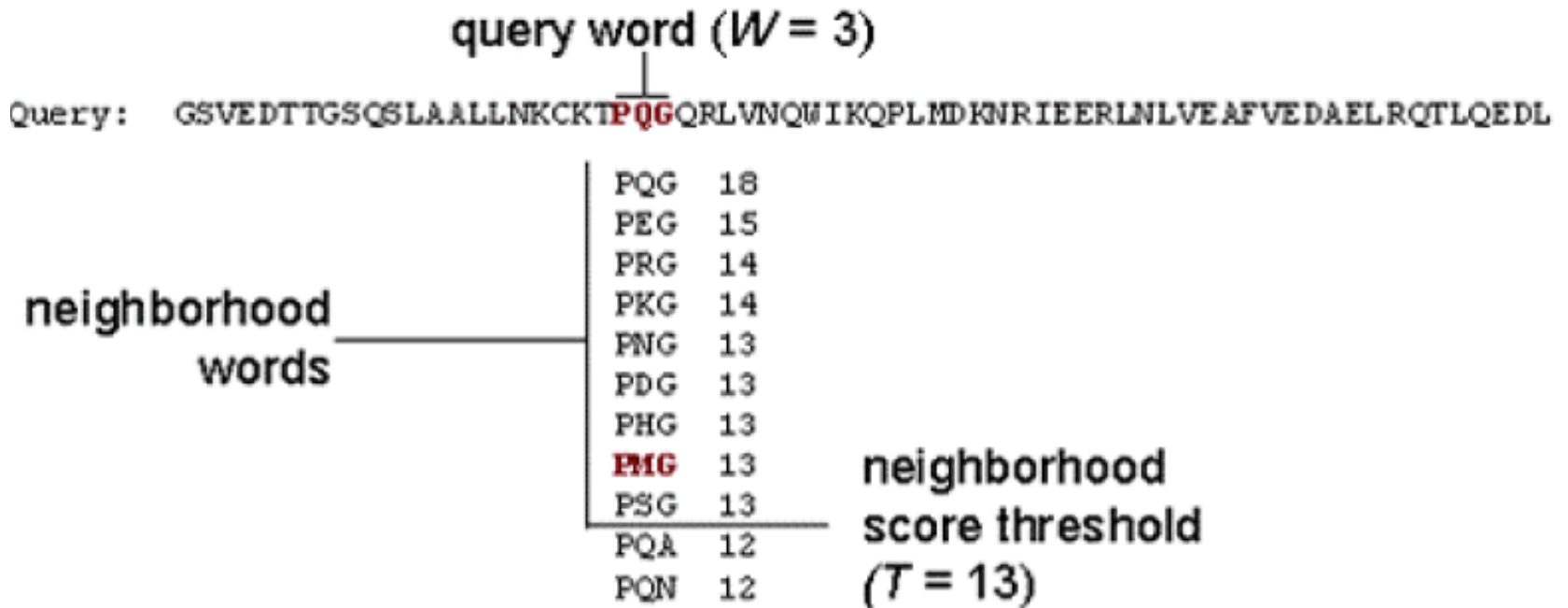
Алгоритм BLAST: идея

Несмотря на снижение сходства родственных последовательностей при их расхождении с течением времени, мы можем рассчитать и обнаружить короткие участки высокого сходства, не затронутые мутациями.



BLAST

1. Для каждого слова длины W в искомой последовательности составляется список схожих слов, вес выравнивания которых выше определенного порога T .



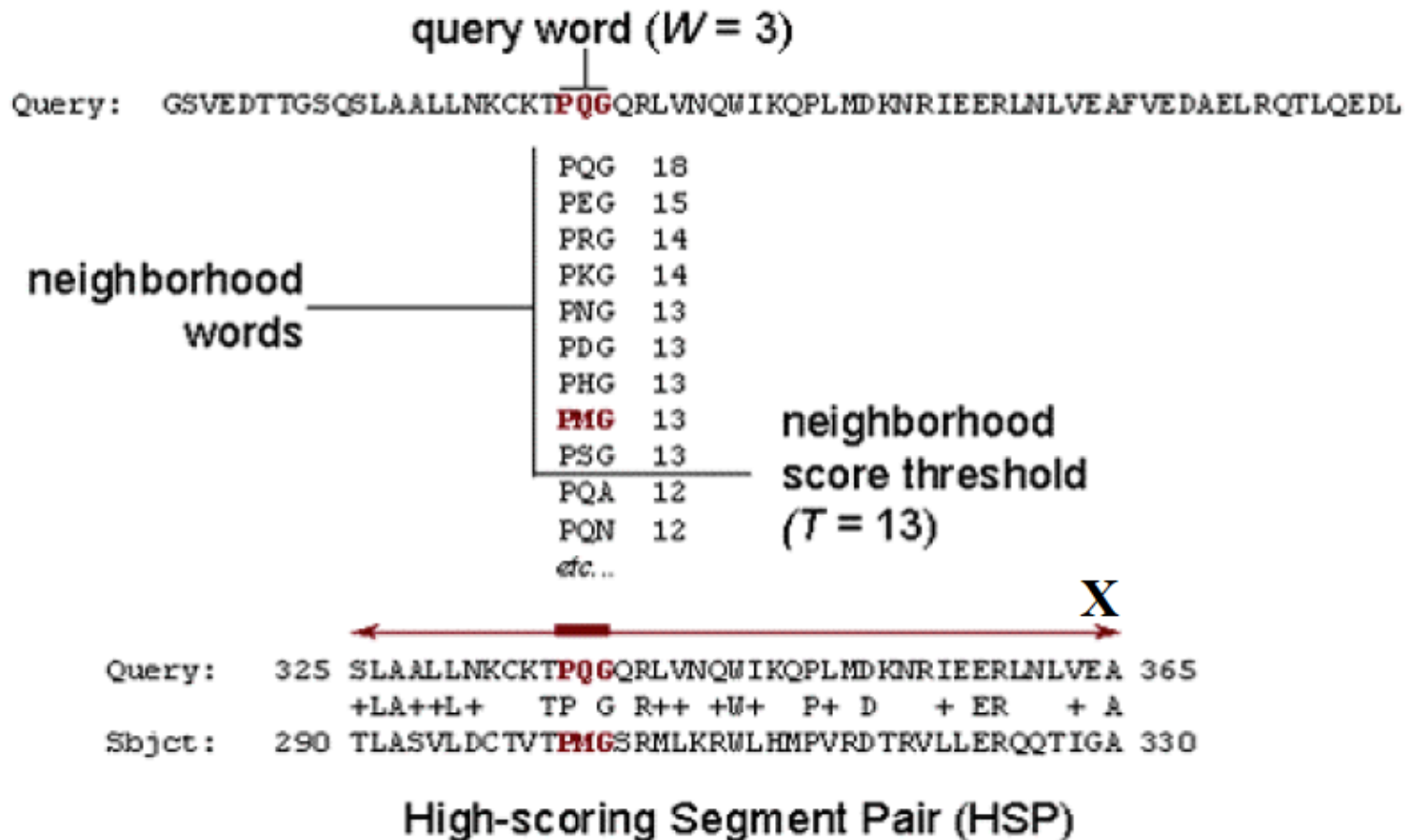
BLAST

2. Для каждого слова обрабатываем составленный для него список схожих слов - ищем, по заранее построенной хэш-таблице, последовательности в базе данных, имеющие точное вхождение данных слов.

Слово	Индексы записей в БД
AAA	1,7,457,2957,...
AAC	34,756,2345,71928,...
AAD	3,75,827,1876,...
AAE	7,15,234,987,...
AAF	71,743,18762,...
AAG	55,221,347,876,...
...	...

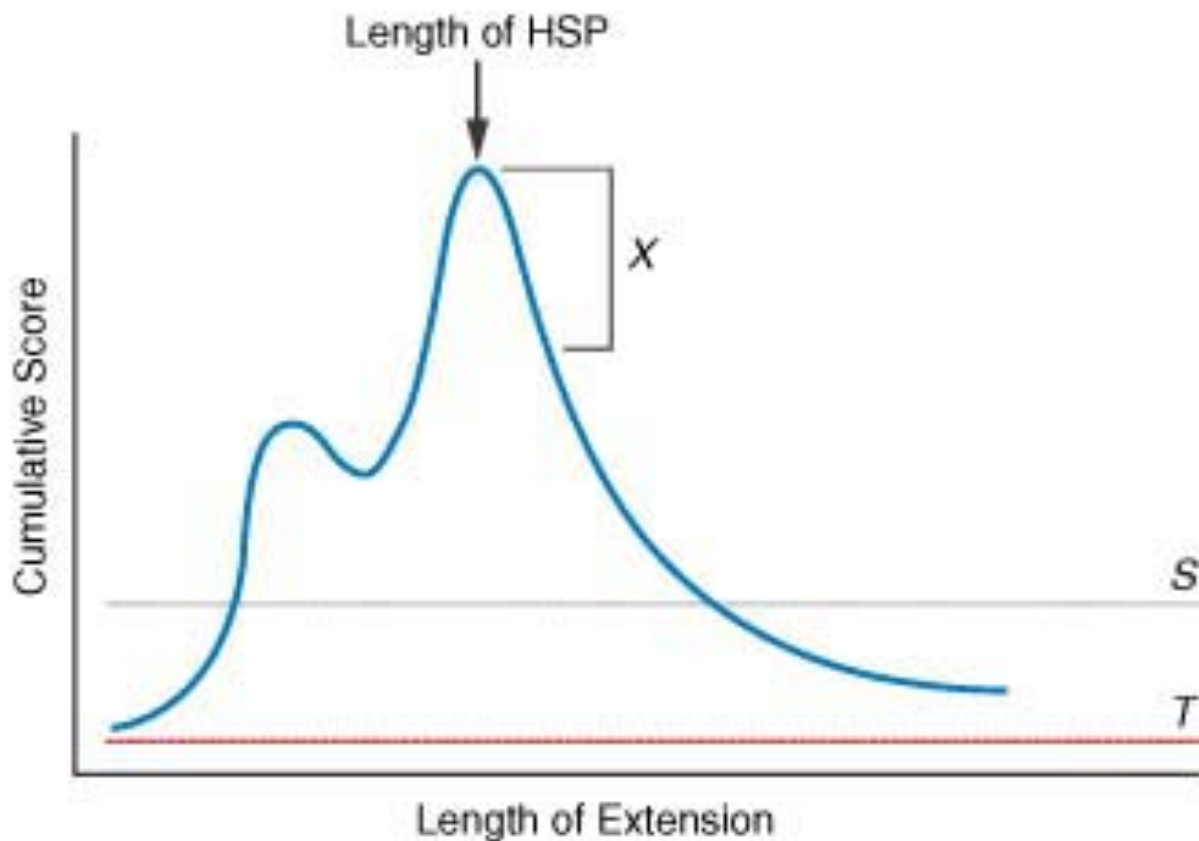
BLAST

3. Расширяем выравнивание вправо и влево от найденных “затравок” используя алгоритм динамического программирования.



BLAST

4. Прекращаем расширение выравнивание если падение суммарного веса выравнивания от точки последнего максимума достигнет заранее установленного порога X . Устанавливаем длину выравнивания в позиции последнего максимума.



Благодарности

- При подготовке слайдов использовались материалы лекций:
 - Михаила Гельфанда (ИППИ)
 - Андрея Миронова (МГУ)
 - Serafim Batzoglou (Stanford)
 - Manolis Kellis (MIT)
 - Pavel Pevzner (UCSD)