

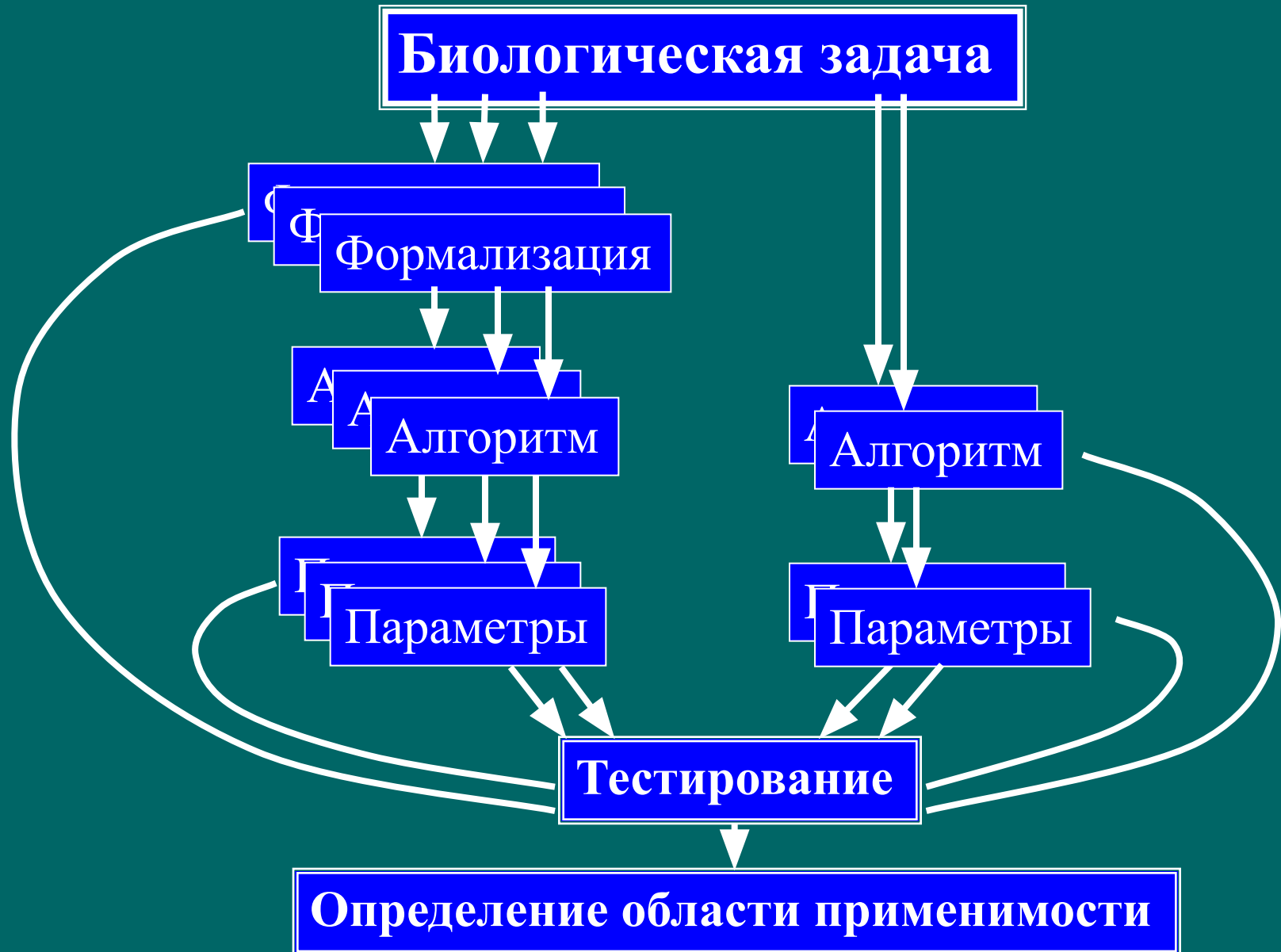
Алгоритмы биоинформатики

ФББ

2004 г., осенний семестр, 3-й курс.

Миронов Андрей Александрович

Информатика и Биоинформатика



Пример: сравнение последовательностей

- Тестирование: алгоритм должен распознавать последовательности, для которых известно, что они биологически (структурно и/или функционально) **сходны**

Сравнение последовательностей

- Формализация1: глобальное выравнивание
- Алгоритм1: Граф выравнивания, динамическое программирование
- Алгоритм1а: Граф выравнивания, динамическое программирование, линейная память
- Параметры: Матрица сходства, штраф за делецию

Сравнение последовательностей

- Формализация2: локальное выравнивание
- Алгоритм2: Граф локального выравнивания, динамическое программирование
- Параметры: Матрица сходства, штраф за делецию

Сравнение последовательностей

- Формализация3: локальное выравнивание с аффинными штрафами
- Алгоритм3: Расширенный граф локального выравнивания, динамическое программирование
- Параметры: Матрица сходства, штраф за открытие делеции, штраф за расширение делеции

Сравнение последовательностей

- Алгоритм4: FASTA. формальная задача плохо определена
- Параметры: Размер якоря, матрица сходства, штраф за делецию

Сравнение последовательностей

- Алгоритм5: BLAST. формальная задача плохо определена
- Параметры: Размер якоря, матрица сходства, штраф за делецию

Выравнивания

Редакционное расстояние

- Элементарное преобразование последовательности: замена буквы или удаление буквы или вставка буквы.
- Редакционное расстояние: минимальное количество элементарных преобразований, переводящих одну последовательность в другую.
- Формализация задачи сравнения последовательностей: найти редакционное расстояние и набор преобразований, его реализующий

Сколько существует выравниваний?

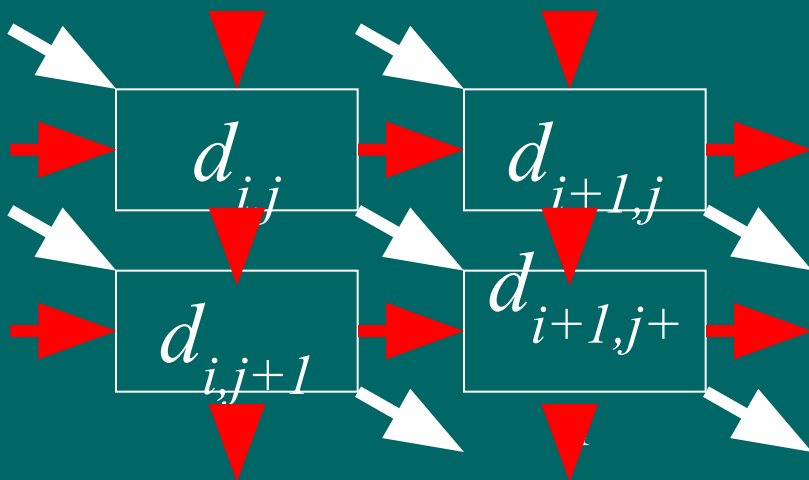
- Дано: две последовательности S_1 и S_2 длиной m и n . Сколько есть способов написать одну последовательность под другой (со вставками)?
- Построим выборочную последовательность S длиной $m+n$ следующим образом: возьмем несколько символов из последовательности S_1 , потом несколько символов из последовательности S_2 потом опять несколько символов из S_1 , потом опять несколько из S_2 .
 - Каждой выборочной последовательности S соответствует выравнивание и по каждому выравниванию можно построить выборочную последовательность. (Доказать!)
 - Количество выборочных последовательностей равно $N_{sel} = C_{n+m}^m = (m+n)! / (m! * n!)$ (Доказать!)

$$N_{align} = \frac{(2n)!}{(n!)^2} = \frac{2^{2n}}{\sqrt{\pi n}} \approx \frac{2^{2n}}{\sqrt{\pi n}}$$

Динамическое программирование для редакционного расстояния

- Граф редакционного расстояния для последовательностей S^1, S^2 : вершина $v_{i,j}$ соответствует префиксам последовательностей $\{S^1_{1..i}\}, \{S^2_{1..j}\}$. На вершине записано редакционное расстояние между префиксами.

(красные стрелки соответствуют вставкам и удалениям)



$$d_{i+1,j+1} = \min \{ d_{i+1,j} + 1, \\ d_{i,j+1} + 1, \\ d_{i,j} + e_{i+1,j+1} \}$$

$$e_{i,j} = \begin{cases} 0, & S^1_i = S^2_j; \\ 1, & S^1_i \neq S^2_j \end{cases}$$

Подмена задачи и обобщение

- Заменим расстояния $d_{i,j}$ на $-d_{i,j}$. Тогда операцию **min** надо заменить на **max**.
- Прибавим к $-d_{i,j}$ $1/2$ ($w_{i,j} = 1/2 - d_{i,j}$), тогда получим функцию сходства: совпадение = $1/2$, замена = $-1/2$, делеция = -1 .
- Функцию сходства W легко обобщить, варьируя штрафы за замену и делеции.
- Новая задача: написать одну последовательность под другой так, чтобы максимизировать сходство

Граничные условия

Нача
по

$w_{1,1}$

$w_{1,2}$

$w_{2,1}$

$w_{i,j}$

$w_{i+1,j}$

$w_{i,j+1}$

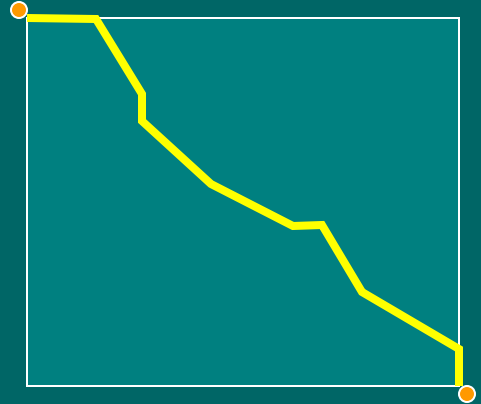
$w_{i+1,j+1}$

$w_{n,m-1}$

$w_{n-1,m}$

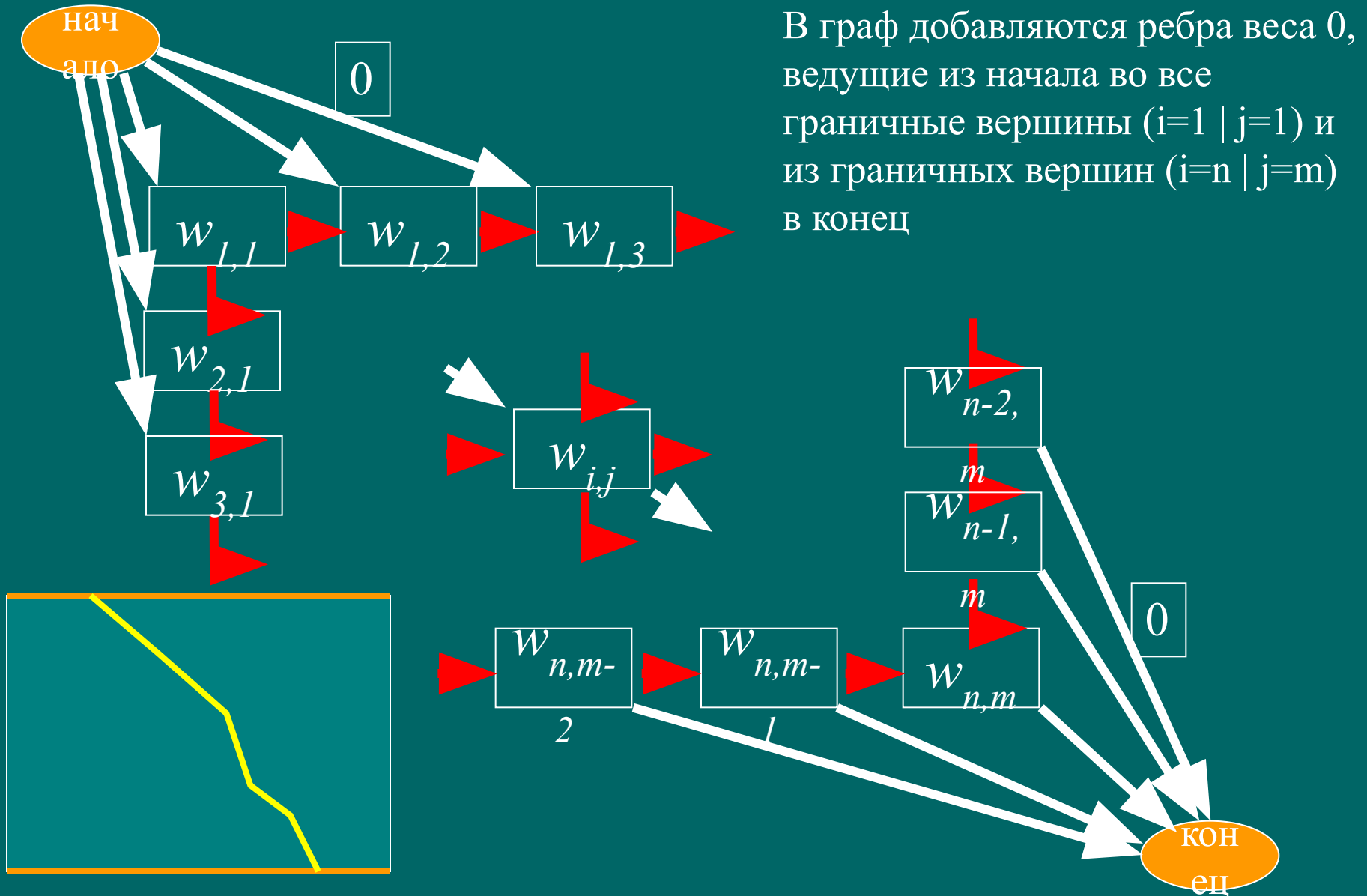
$w_{n,m}$

КОНЕ
ц



При таких граничных условиях начальные и конечные делеции штрафуются

Как не штрафовать за концевые делеции



Оценка времени работы и необходимой памяти

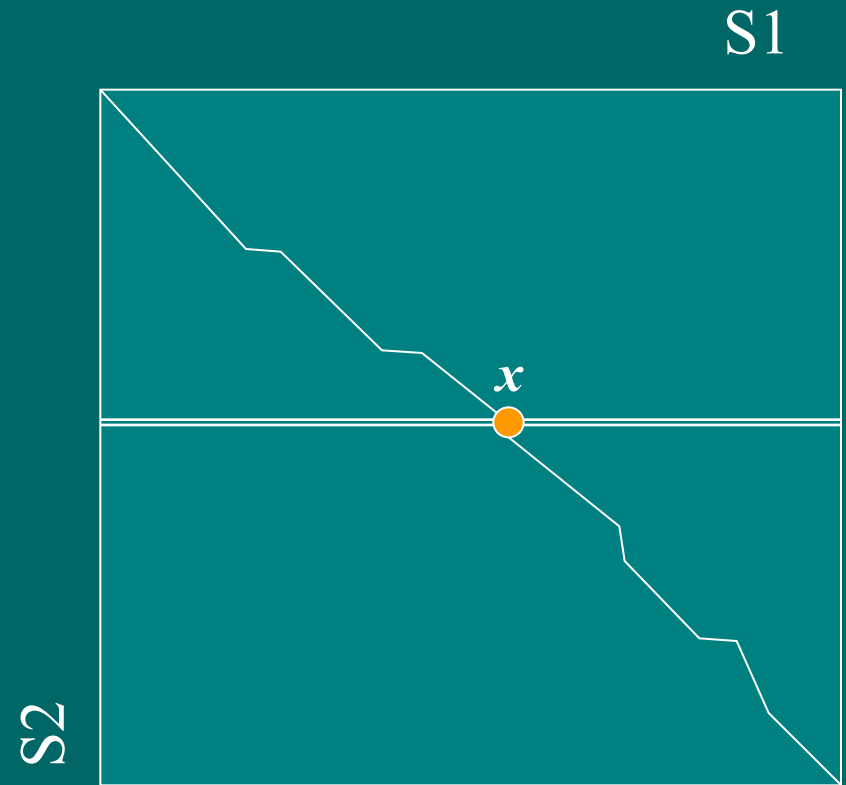
- Алгоритм посматривает все вершины графа
- В каждой вершине делается 3 сравнения
- Количество необходимых операций (время работы алгоритма): $T=O(n*m)$. Говорят, что алгоритм выравнивания квадратичен по времени работы.
- Для запоминания весов и восстановления оптимального выравнивания надо в каждой вершине запомнить ее вес и направление перехода. Таким образом, алгоритм квадратичен по памяти.

Где можно сэкономить?

- Во-первых не обязательно запоминать веса во всех вершинах. При просмотре матрицы выравнивания (графа выравнивания) можно идти по строкам. При этом нам необходима только предыдущая строка.

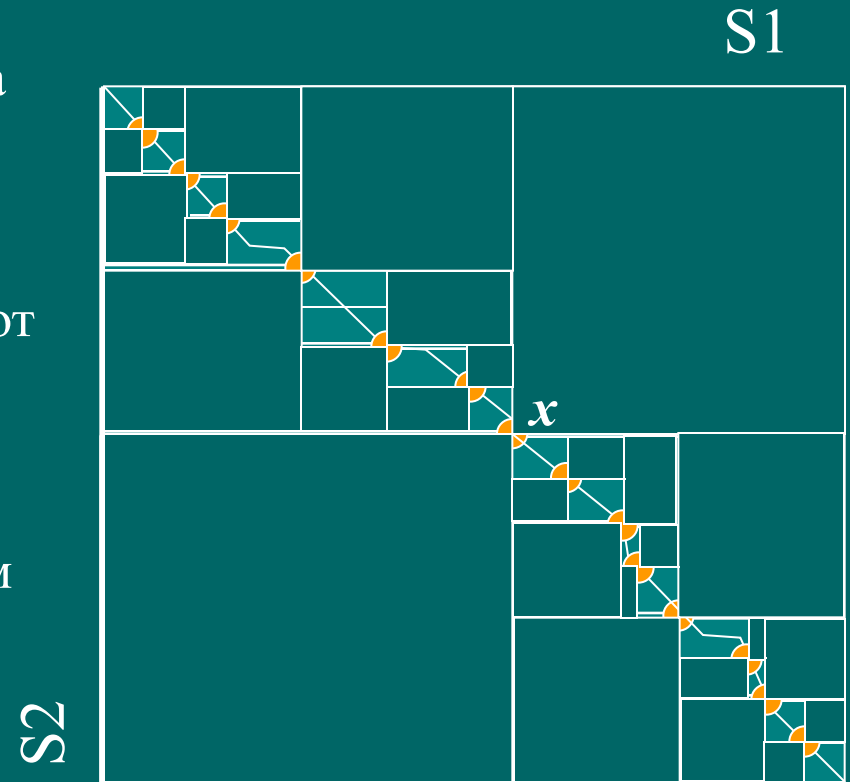
Линейный по памяти алгоритм Миллера-Маерса

- Разбиваем одну из последовательностей на две равные части
- Для каждой точки x линии раздела находим веса оптимальных выравниваний из начала в x и из конца в x :
 $W^+(x)$, $W^-(x)$.
- Вес оптимального выравнивания, проходящего через точку x равен
 $W(x) = W^+(x) + W^-(x)$.
- Вес оптимального выравнивания равен
 $W = \max_x (W(x))$
- Таким образом, найдена одна точка, чрез которую проходит оптимальное выравнивание за время $T = C * n^2$.



Алгоритм Миллера-Маерса

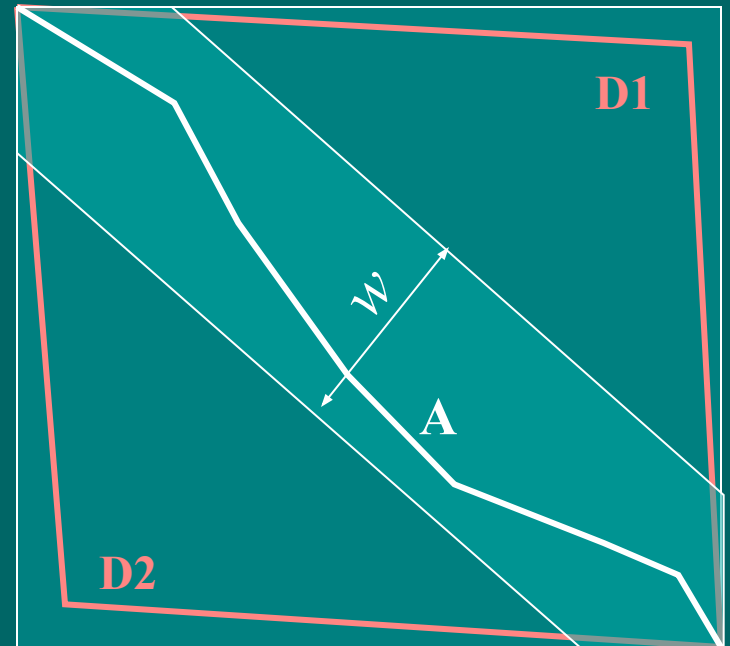
- Найденная точка x разбивает матрицу выравнивания на четыре квадранта, два из которых заведомо не содержат оптимального выравнивания
- Для двух квадрантов, содержащих оптимальный путь можно применить тот же прием, и запомнить точки x' и x'' .
- Просмотр оставшихся квадрантов требует времени $T=C*n^2/2$ (почему?)
- Продолжая процедуру деления пополам найдем все точки, через которые проходит оптимальный путь.
- Время работы алгоритма
$$T=C*n^2+C*n^2/2+C*n^2/4+...=$$
$$C*n^2(1+1/2+1/4+1/8+...);$$
$$T=2C*n^2;$$



Важно, что при просмотре мы не запоминали обратных переходов!

Еще один способ сэкономить время и память

- Ясно, что выравнивания D1 и D2 не представляют интереса, поскольку содержат в основном делеции
- Разумные выравнивания (A) лежат в полосе
- Алгоритм: задаемся шириной полосы w и просматриваем только те вершины графа, что лежат в указанной полосе.

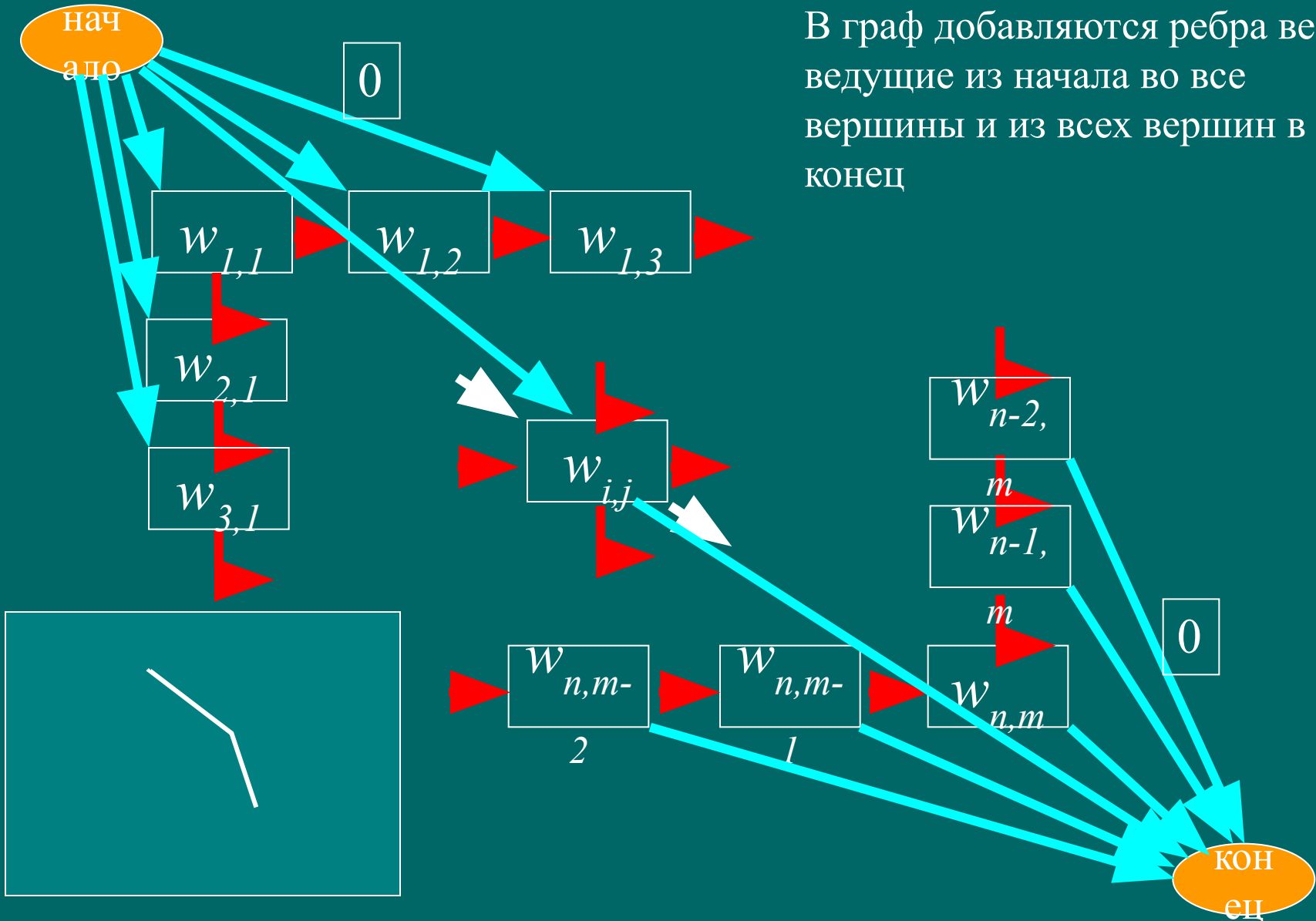


Локальное выравнивание

- Локальным оптимальным выравниванием называется такое оптимальное выравнивание фрагментов последовательностей, при котором любое удлинение или укорочение фрагментов приводит только к уменьшению веса.
- Локальному оптимальному выравниванию отвечает путь с наибольшим весом, независимо от того, где он начинается и где кончается.

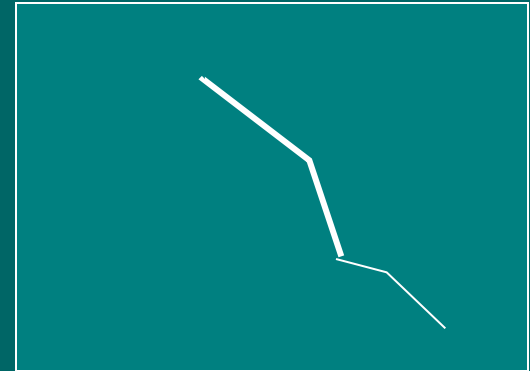
Алгоритм Смита-Ватермана

В граф добавляются ребра веса 0, ведущие из начала во все вершины и из всех вершин в конец



Алгоритм Смита-Ватермана

- Пусть есть какой-то путь с неотрицательными весами
- Построим график веса вдоль пути
- Абсолютный максимум на этом графике определит точку окончания пути



Алгоритм Смита-Ватермана

$$w_{i,j} = \max \left\{ \begin{array}{l} w_{i-i,j-1} + e_{i,j}, \quad i > 1, j > 1 \\ w_{i-1,j} - d, \quad i > 1 \\ w_{i,j-1} - d, \quad j > 1 \\ \mathbf{0} \end{array} \right\}$$

- Точка конца пути (от нее начинаем обратный просмотр и восстановление пути) определяется так:

$$(i_{max}, j_{max}) = \mathbf{argmax} (w_{i,j})$$

Пусть (при одинаковых параметрах) мы получили вес глобального выравнивания w_{glob} и вес локального выравнивания w_{loc} . Какая величина больше?

Более общая зависимость штрафа за делецию от величины делеции

- Простейшая модель делеции: элементарное событие – удаление одного символа. Протяженная делеция – несколько независимых событий удаления одного символа. Работает плохо.
- По-видимому более реалистичная модель делеция нескольких символов происходит за одно элементарное событие, а размер делеции является некоторой случайной величиной. Поэтому в качестве штрафа хорошо бы взять что-нибудь вроде $\Delta(l) = a \log(l + 1)$, где l – длина делеции
В любом случае функция $\Delta(l)$ должна быть выпуклой – должно выполняться неравенство треугольника:

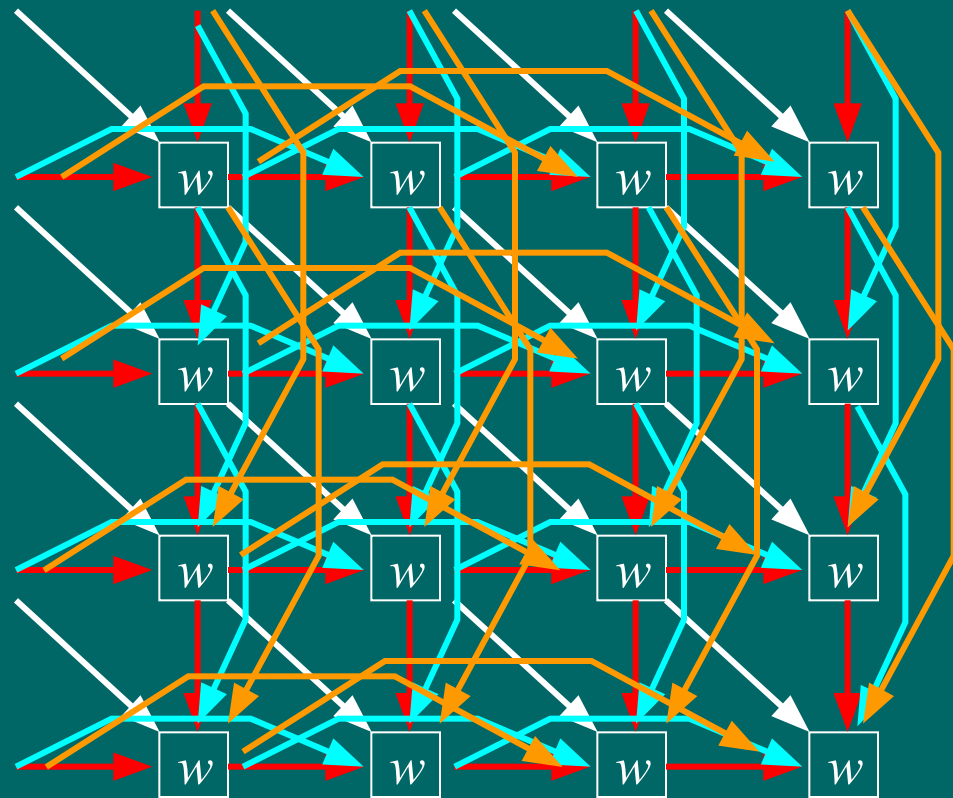
$$\Delta(l_1 + l_2) \leq \Delta(l_1) + \Delta(l_2)$$

Более общая зависимость штрафа за делецию от величины делеции. Алгоритм.

Теперь надо просматривать все возможные варианты делеций. Поэтому в каждую вершину входит не 3 ребра, а примерно $(n+m)/2$ ребер, где n, m – длины последовательностей

Поэтому время работы алгоритма становится кубичным:

$$T = O (nm (n+m));$$

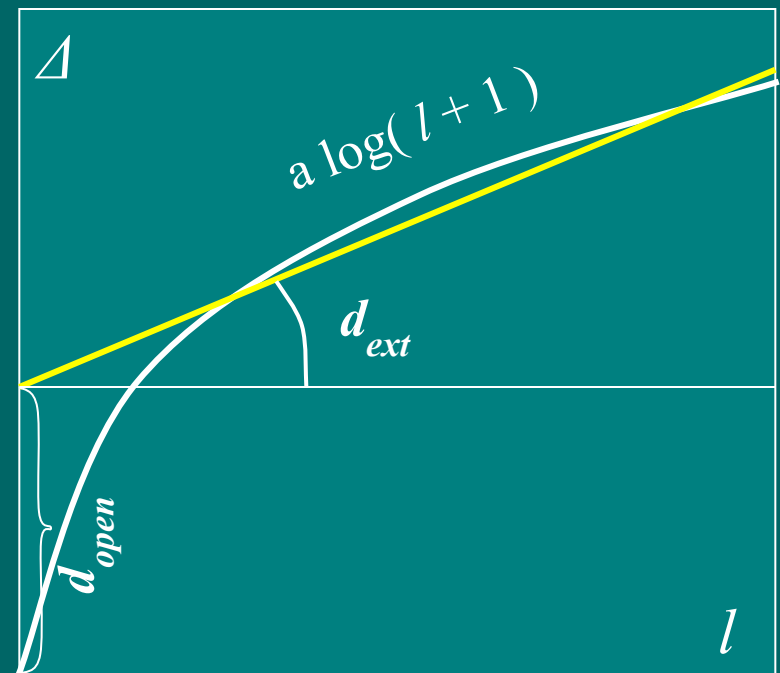


Аффинные штрафы за делецию

- Вместо логарифмической зависимости используют зависимость вида:

$$\Delta(l) = d_{open} + l d_{ext}$$

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



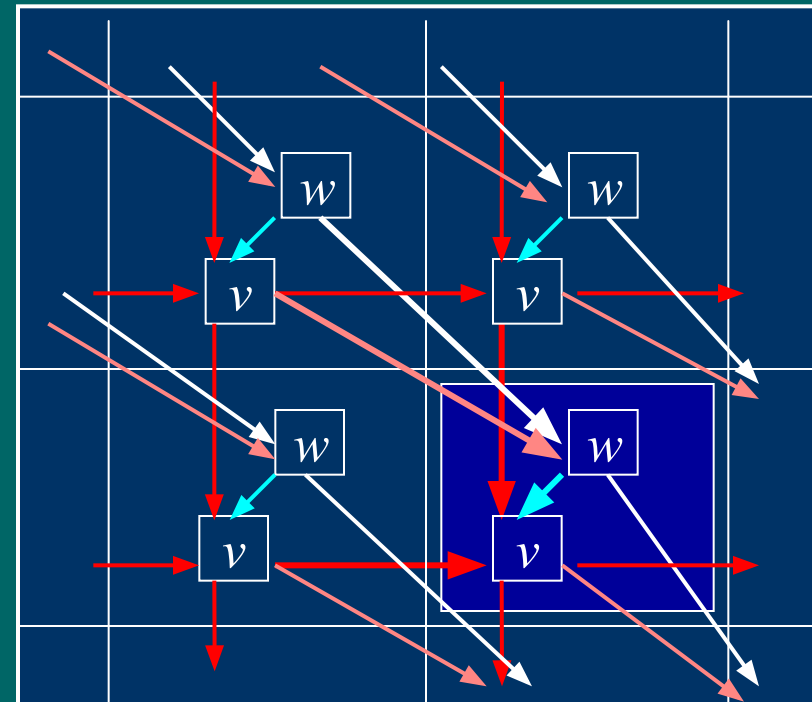
Алгоритм для аффинных штрафов

Модификация стандартного графа:





1. В каждой ячейке вводится дополнительная вершина (v), отвечающая делеционному пути
2. Вводятся делеционные ребра для открытия и закрытия делеции (из вершин типа w в вершины типа v и обратно)
3. Ребра, отвечающие продолжению делеции переносятся на новые вершины

Число вершин графа равно $2mn$
число ребер равно $5mn$

Трудоемкость алгоритма равна:
 $T = O(mn)$



Веса на ребрах

-  $e_{i,j}$ сопоставление
-  d_{open} открытие делеции
-  d_{ext} продолжение делеции
-  $e_{i,j}$ закрытие делеции

Рекурсия для аффинных штрафов

- $w_{i,j} = \max (w_{i-1,j-1} + e_{ij}, v_{i-1,j-1} + e_{ij}, 0);$

- $v_{i,j} = \max (w_{i,j} - d_{\text{open}}, v_{i-1,j} - d_{\text{ext}}, v_{i,j-1} - d_{\text{ext}});$

- $(i_{\text{max}}, j_{\text{max}}) = \operatorname{argmax} (w_{i,j})$

Матрицы замен

Откуда берутся параметры для выравнивания?

- Пусть у нас есть выравнивание. Если последовательности случайные и независимые (модель R), то вероятность увидеть букву α против β

$$p(\alpha, \beta | R) = p(\alpha) p(\beta)$$

а вероятность выравнивания (x, y) будет равна

$$p(x, y | R) = \prod p(x_i) \prod p(y_i)$$

Если выравнивание не случайно (модель M), то

$$p(x, y | M) = \prod p(x_i, y_i)$$

Отношение правдоподобия:

$$\frac{p(x, y | \underline{M})}{p(x, y | R)} = \frac{\prod p(x_i, y_i)}{\prod p(x_i) \prod p(y_i)}$$

Логарифмируя, получаем

$$\log(p(x, y | M) / p(x, y | R)) = \sum s(x_i, y_i);$$

$$\underline{\text{Матрица замен:}} \quad s(\alpha, \beta) = \log(p_{\alpha\beta} / p_{\alpha} p_{\beta})$$

Серия матриц BLOSUM

- База данных BLOCKS (*Henikoff & Henikoff*) – безделеционные фрагменты множественных выравниваний (выравнивания получены *экспертом*).
- В каждом блоке отбираем подмножество последовательностей, имеющих процент идентичных аминокислот не больше заданного значения ID.
- В урезанном блоке в каждой колонке подсчитываем число пар аминокислот
$$n_{col}^{bl}(\alpha, \beta)$$
- Усредняем по всем колонкам и по всем блокам:
$$f(\alpha, \beta) = \sum n_{col}^{bl}(\alpha, \beta) / N_{col}$$
- Элемент матрицы $BLOSUM_{ID}$:

$$BLOSUM_{ID}(\alpha, \beta) = \log(f(\alpha, \beta) / f(\alpha) f(\beta))$$

Серия матриц РАМ

- Point Accepted Mutation – эволюционное расстояние, при котором произошла одна замена на 100 остатков.
- Эволюционный процесс можно представить как Марковский процесс. Если в начальный момент времени $t=0$ в некоторой позиции был остаток α , то через время Δt в этой позиции с некоторой вероятностью будет остаток β :

$$p(\beta | \alpha, \Delta t) = M_{\Delta t}(\beta, \alpha)$$

M_{Δ} – эволюционная матрица

Через время $2 \cdot \Delta t$

$$p(\beta | \alpha, 2 \cdot \Delta t) = \sum_{\gamma} M_{\Delta t}(\beta, \gamma) \cdot M_{\Delta t}(\gamma, \alpha) = M_{\Delta t}^2(\beta, \alpha)$$

Через время $N \cdot \Delta t$

$$p(\beta | \alpha, N \cdot \Delta t) = M_{\Delta t}^N(\beta, \alpha)$$

Серия матриц РАМ

- Находим выравнивания, отвечающие расстоянию РАМ1
- Находим частоты пар и вычисляем частоты пар:

$$p(\alpha\beta) = p(\alpha \rightarrow \beta) p(\alpha) + p(\beta \rightarrow \alpha) p(\beta)$$

полагая $p(\alpha \rightarrow \beta) = p(\beta \rightarrow \alpha)$ получаем

$$p(\alpha \rightarrow \beta) = p(\alpha\beta) / (p(\alpha) + p(\beta))$$

$$p(\alpha \rightarrow \alpha) = 1 - \sum_{\beta \neq \alpha} p(\alpha \rightarrow \beta)$$

$$\text{РАМ}_N(\alpha\beta) = \log \left(p_{(\alpha \rightarrow \beta)}^N / p_\alpha p_\beta \right)$$

Статистика выравниваний

Параметры выравнивания

- В простейшем случае есть три параметра:
 - премия за совпадение (*match*)
 - штраф за несовпадение (*mism*)
 - штраф за делецию (*indel*)
- Если все параметры умножить на одну и ту же положительную величину, то само оптимальное выравнивание не изменится, а вес выравнивания умножится на ту же величину
- Поэтому можно положить *match*=1.
- Если $mism > 2 * indel$, то выравнивание не будет иметь замен. (почему?)

Статистика выравниваний

- Допустим мы выровняли две последовательности длиной 100 и получили вес 20. Что это значит? Может быть при выравнивании двух случайных последовательностей будет тот же вес?
- А что такое случайные последовательности?

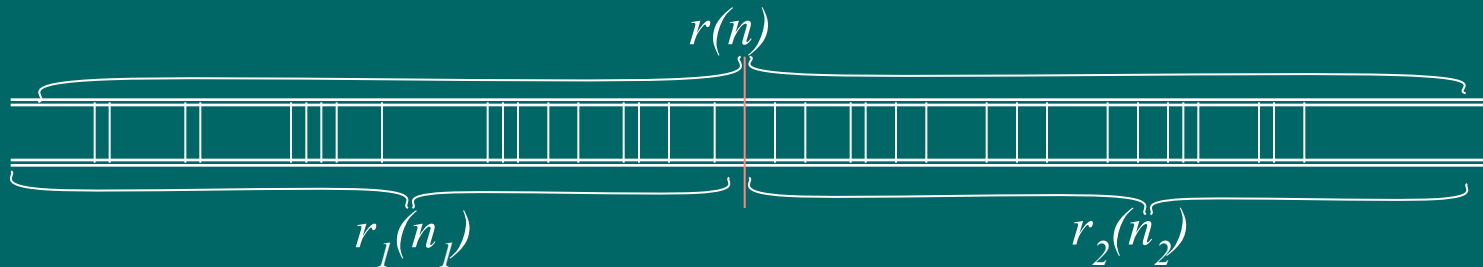
Статистика выравниваний

- Базовая (вообще говоря неправильная) модель — Бернуллиевские последовательности (символы генерируются независимо друг от друга с заданной вероятностью). Для этой модели математика проще и проще получить оценки
- Уточненная модель (лучше, но тоже неправильная) — Марковская цепь (вероятность появления следующего символа зависит от нескольких предыдущих символов). Математика значительно сложнее. Почти ничего не известно.

Частные случаи локального выравнивания

- $mism = 0, indel = 0$ – максимальная общая подпоследовательность
- $mism = \infty, indel = \infty$ – максимальное общее подслово

Наибольшая общая подпоследовательность



- Длина оптимальной подпоследовательности есть случайная величина $r(n)$, зависящая от длины последовательностей.
- Пусть две последовательности длиной n разбиты на два фрагмента длиной n_1 и n_2 ($n_1 + n_2 = n$)
- Ясно, что оптимальная подпоследовательность будет не хуже, чем объединение оптимальных подпоследовательностей для фрагментов:

$$r(n) \geq r_1(n_1) + r_2(n_2) \quad (\text{попробуйте понять смысл неравенства})$$

- Отсюда следует, что математическое ожидание $M(r(n)) \geq M(r(n_1)) + M(r(n_2))$, или $M(r(n)) \geq c \cdot n$
- Можно показать, что $M(r(n)) - M(r(n_1)) + M(r(n_2)) \rightarrow 0$

- Поэтому:

$$M(r(n)) \approx c \cdot n, \quad (n \rightarrow \infty)$$

Наибольшее общее слово

- Наложим одну последовательность на другую. Будем идти вдоль пары последовательностей и, если буквы совпали, то будем считать успехом, иначе – неудача. Имеем классическую схему испытаний Бернулли. Наибольшему общему слову при таких испытаниях будет соответствовать максимальная серия успехов. Известно, что средняя величина максимальной серии успехов равна:

$$M(l) = \log_{1/p}(n)$$

- Возможных наложений много (порядка длины последовательности). Максимальное общее слово есть максимум от максимальных серий успехов при всех возможных наложениях. Показано (*Waterman*), что:

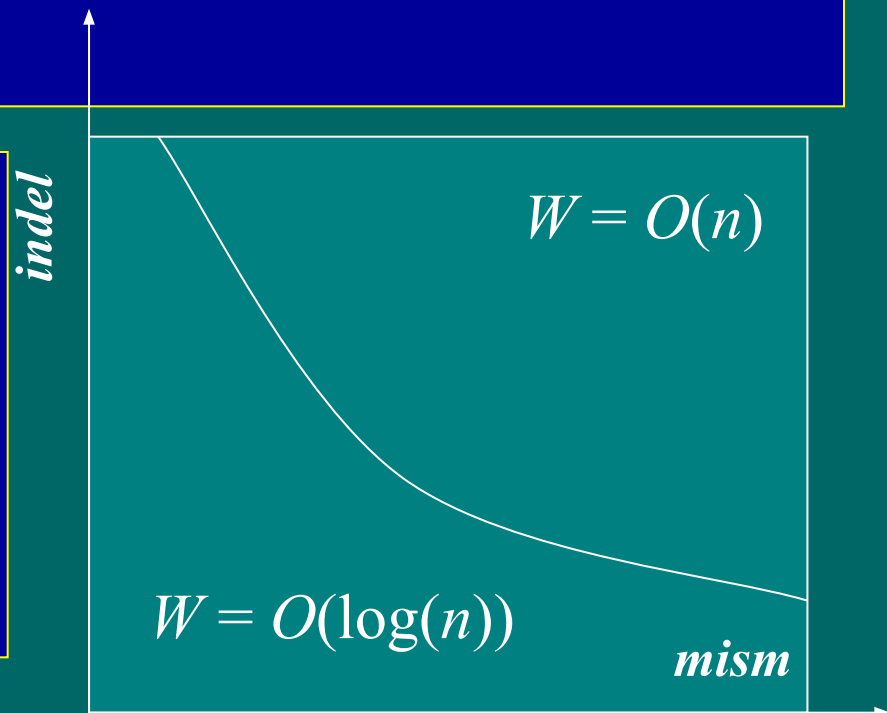
$$M(l) \approx \log_{1/p}(nm) + \log_{1/p}(1-p) + \gamma \cdot \log(e) - 1/2 = \log_{1/p}(Knm), \quad (m, n \rightarrow \infty, \gamma \approx 0.577)$$

$$\sigma(l) \approx [\pi \log_{1/p}(e)]^2 / 6 + 1/2, \quad (\text{не зависит от } l !)$$

Зависимость от параметров

- Показано, что зависимость *ожидаемого* веса выравнивания от длины последовательности может быть либо логарифмической либо линейной в зависимости от параметров. Все пространство параметров разбивается некой поверхностью на две области поведения.

При безделеционном выравнивании поведение логарифмическое, если мат. ожидание веса двух случайных сегментов отрицательно.



Распределение экстремальных значений

- Пусть вес выравнивания x (случайная величина) имеет распределение

$$G(S) = P(x < S)$$

- Тогда при N независимых испытаниях распределение максимального значения будет

$$G_N(x) = G^N(x);$$

- Можно показать, что для нормально распределенного $G(x)$

$$G_N(x) \approx \exp(-KN e^{\lambda(x-\mu)})$$

e-value & p-value

- Количество независимых локальных выравниваний с весом $>S$ описывается распределением Пуассона (*Karlin & Altschul*) :

$$E(S) = Kmn e^{-\lambda S}$$

где λ – положительный корень уравнения

$\sum p_\alpha p_\beta e^{\lambda s(\alpha\beta)} = 1$, $s(\alpha\beta)$ – матрица замен
K – константа, зависящая от p_α и $s(\alpha\beta)$.

- **e-value:** $E(S)$ – *ожидаемое количество выравниваний с заданным весом*
- **p-value:** $p(x>S) = 1 - e^{-E(S)}$ – *Вероятность встретить выравнивание с таким или большим весом*

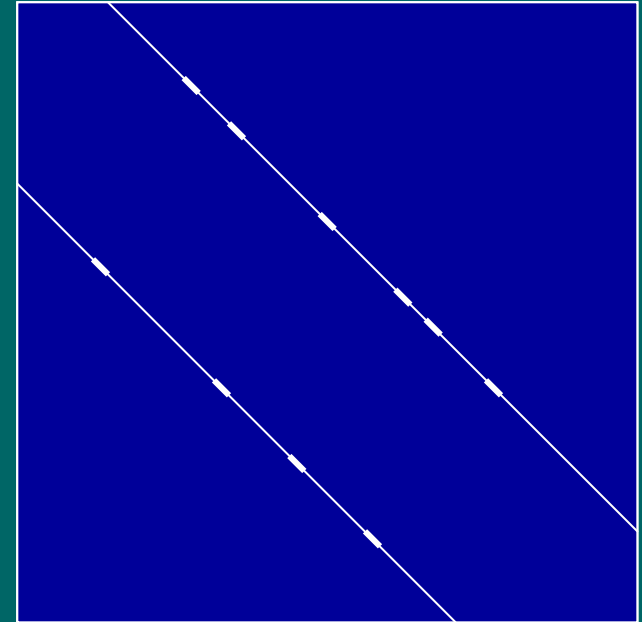
Поиск по банку

Поиск по банку. Хеширование.

- Подготовка банка – построение хэш-таблицы. Хэш-функция – номер слова заданного размера (1-tuple, *l-грамма*).
- В хэш-таблице хранятся списки ссылок на последовательности и на позиции в последовательностях, где встречается соответствующая *l-грамма*.
- При поиске запроса (query) в последовательности запроса последовательно находятся *l-граммы*, далее, по хэш-таблице для них находятся соответствующие документы и позиции.
- Пара совпадающих *l-грамм* в запросе и в банке называется *затравкой, якорем, seed*.

Поиск по банку. FASTA.

- Используется техника поиска якорей с помощью хэш-таблицы.
- Два якоря (i_1, j_1) , (i_2, j_2) принадлежат одной диагонали, если
$$i_1 - j_1 = i_2 - j_2$$
- Мощностью диагонали называется количество якорей, принадлежащих диагонали. Иногда в мощность диагонали включают мощности соседних диагоналей (чтобы учесть возможность делеций)
- Отбираем n^* ($n^*=10$) самых мощных диагоналей и для них пытаемся построить цепочки якорей, или строим S-W выравнивание в полосе (*Wilbur-Lipman-Pearson*)



Для оценки стат. значимости используют z-score

Поиск по банку. BLAST1.

- Ищем якоря с помощью хэш-таблицы
- Каждый якорь расширяем с тем, чтобы получить сегмент совпадения наибольшего веса (HSP – high scoring pair).
- Оцениваем его статистическую значимость, и, если она больше порога, то репортируем
- Для оценки значимости используется формула Альтшуля

(Altschul, Lipman, Pearson)

Поиск по банку. BLAST2.

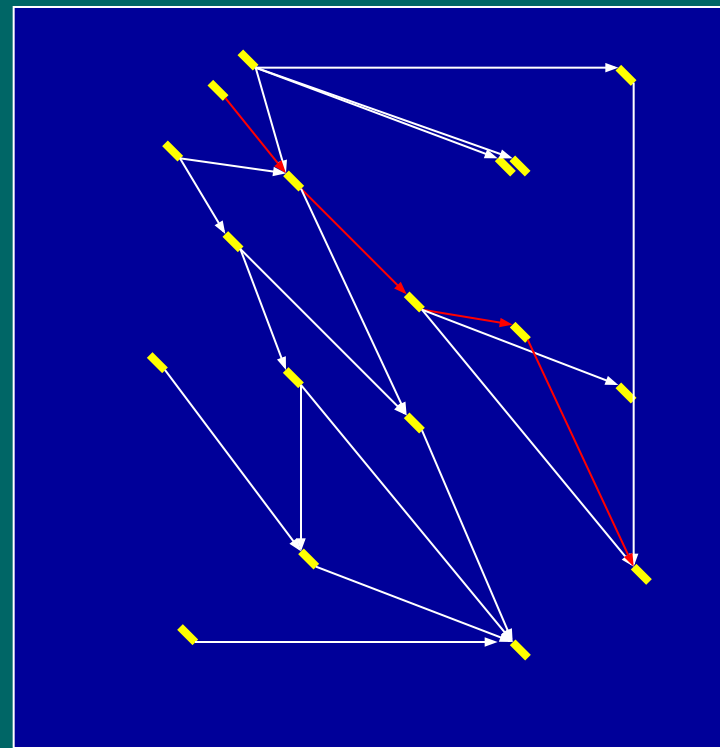
- T-соседней 1-граммой L^T для 1-граммы L называется такая 1-грамма, что вес ее сравнения с L не меньше заданного T :

$$\sum s(L_p, L_i^T) \geq T$$

- Для аминокислотных последовательностей при просмотре запроса формируем не только те 1-граммы, которые встретились в нем, но также все T-соседние 1-граммы. Характерное количество 1-грамм для белка длиной 300 остатков – 15000.
- Расширяются только те якоря, которые принадлежат мощной диагонали (как в FASTA), причем мощность диагонали должна быть $\geq 2T$
- При расширении диагонали допускается небольшое количество делеций

Быстрое выравнивание

- Ищем якоря с помощью хэш-таблицы
- Якорь (i_1, j_1) предшествует якорю (i_2, j_2) , если
$$i_1 < i_2 \ \& \ j_1 < j_2$$
$$\& \ i_2 - i_1 < d \ \& \ j_2 - j_1 < d$$
- Получаем ориентированный граф с небольшим количеством вершин и ребер
- Можно найти оптимальную цепочку якорей методом динамического программирования



Введение в Байесову статистику

Введение в Байесову статистику

- Задача. Мы 3 раза бросили монету и 3 раза выпал орел. Какова вероятность выпадения орла у этой монеты?
 - Если мы уверены, что монета не кривая, то $p = 1/2$
 - Допустим, что мы взяли монету из мешка, а в мешке монеты разной кривизны. Но при этом мы знаем как распределена кривизна монет $P_a(p)$ (априорное распределение).
 - Мы хотим на основе наблюдения Z_0 и априорного распределения *распределений вероятностей* оценить вероятность выпадения орла у *данной* монеты.

Введение в Байесову статистику

- $P(Zo | p) = p^3$;
- $P(Zo, p) = P(Zo | p) P_a(p) = P(p | Zo) P(Zo)$;
- $P(p | Zo) = \{P(Zo | p) P_a(p)\} / P(Zo)$;
- Загадочный объект $P(Zo)$ – безусловная вероятность трех орлов. Определяется из условия нормировки: $\int P(p | Zo) = 1$;
- Окончательно, распределение вероятностей вероятности орла будет:
- $P(p | Zo) = p^3 P_a(p) / \int p^3 P_a(p)$;

Введение в Байесову статистику

- $P(p | Z_0) = p^3 P_a(p) / \int p^3 P_a(p) dp$;
- В качестве оценки для искомой вероятности удобно иметь число, а не распределение:
 - Максимальное значение
 $p^{ML} = \operatorname{argmax}_p (P(Z_0 | p))$ – максимальное правдоподобие (max likelihood, ML)
 - Среднее значение
 $p^E = E(P(p | Z_0)) = \int p P(p | Z_0) dp$;

Введение в Байесову статистику

- ML Оценка:

$$p^{ML} = \operatorname{argmax} (p^3) = 1;$$

(не зависит от распределения P_a)

- E оценка (матожидание апостериорной вероятности)

$$p^E = \int p^4 P_a(p) dp / \int p^3 P_a(p) dp;$$

– Если мы уверены, что монета правильная, то $P_a(p) = \delta(p - 1/2)$; $p^E = 1/2$;

– Если мы ничего не знаем о распределении $P_a(p)$, то положим $P_a(p) = \text{const}$. Тогда

$$p^E = \int p^4 P_a(p) dp / \int p^3 P_a(p) dp = 1/4 / 1/3 = 3/4;$$

В более общем случае

$$p^E(n0) = (n+1)/(n+2);$$

- MAP оценка (максимум апостериорной вероятности)

$$p^{MAP} = \operatorname{argmax} \{ P(p | 30) \};$$

Определения

- Пусть у нас есть несколько источников Y событий X (например, несколько монет). Тогда :

$P(X | Y)$ – *условная вероятность*

$P(X, Y) = P(X | Y) P(Y)$ – *совместная вероятность*

$P(X) = \sum_Y P(X, Y) = \sum_Y P(X | Y) P(Y)$ – *полная вероятность*

$P(Y | X)$ – *апостериорная вероятность выбора источника (правдоподобие гипотезы)*

$P(Y)$ – *априорная вероятность выбора источника*

- **Теорема Байеса:**

$$P(X | Y) = P(Y | X) P(X) / P(Y)$$

Пример

- Пусть есть две кости – правильная и кривая (с вероятностью выпадать 6 – 0.5). И пусть нам подсовывают кривую кость с вероятностью 1%. Мы бросили кость 3 раза и 3 раза получили 6. Какова вероятность того, что нам дали кривую кость?

- $$P(\text{кривая кость} \mid 3 \text{ шестерки}) = \frac{P(3 \text{ шестерки} \mid \text{кривая кость}) \cdot P(\text{кривая кость})}{P(3 \text{ шестерки})}$$

$$P(3 \text{ шестерки}) = P(3 \text{ шестерки} \mid \text{кривая кость}) \cdot P(\text{кривая кость}) + P(3 \text{ шестерки} \mid \text{правильная кость}) \cdot P(\text{правильная кость}) = 0.5^3 \cdot 0.01 + (1/6)^3 \cdot 0.99 = 0.00125 + 0.0046 = 0.00585$$

$$P(\text{кривая кость} \mid 3 \text{ шестерки}) = 0.00125 / 0.00585 = 0.21$$

- **Вывод – кость скорее правильная!**

Оценка параметров по результатам

- Пусть у нас есть наблюдение D и некоторый набор параметров распределения θ , которые мы хотим оценить (см. пример про 3 орла). Кроме того, у нас есть представление о том, как эти параметры распределены (*prior*)
- Апостериорное распределение вероятностей параметров получаем из теоремы Байеса:

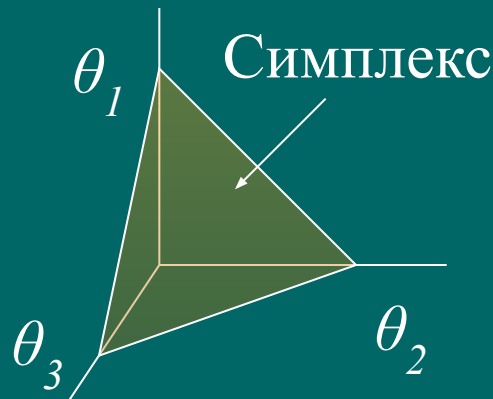
$$P(\theta | D) = \frac{P(\theta) P(D | \theta)}{\int_{\theta} P(\theta') P(D | \theta')}$$

Распределение Дирихле

- Определение:

$$D(\theta|\alpha) = Z^{-1} \prod \theta_i^{\alpha_i} \delta(\sum \theta_i - 1);$$

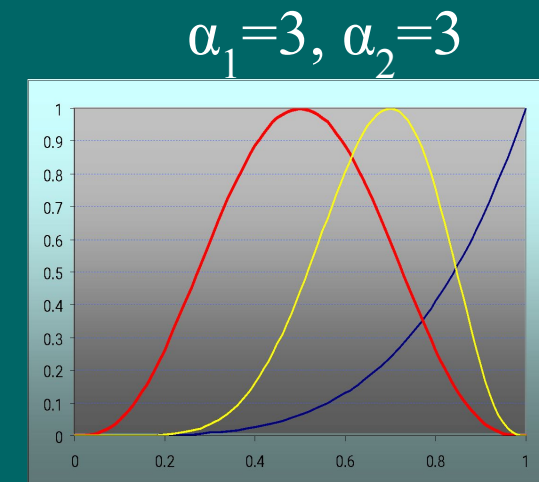
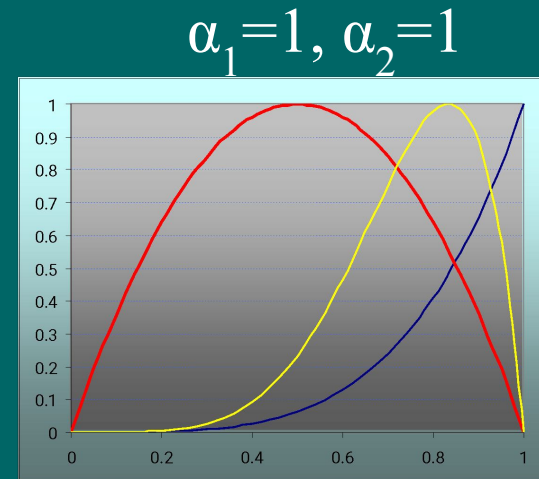
- Z – нормировочный множитель
- α_i – параметры распределения
- $\theta_i \geq 0$ – область определения распределения
- δ – дельта-функция ($\delta(x)=0, x \neq 0; \int \delta(x)dx=1;$)



Задача: найти объем симплекса в n -мерном пространстве

prior = распределение Дирихле

- Часто в качестве prior используют распределение Дирихле. Параметры этого распределения α_i называют *псевдо-отсчетами (pseudo counts)*. Они определяют степень нашего доверия к результатам
- На графиках показаны распределения для случая 4-х орлов при 4-х бросаниях монеты. θ – вероятность орла
 - Синяя линия – $P(D | \theta)$
 - Красная линия – распределение Дирихле $P(\theta)$
 - Желтая линия – апостериорная вероятность выпадения орла $P(\theta | D)$



Скрытые Марковские модели (HMM)

Пример

- Пусть некто имеет две монеты – правильную и кривую. Он бросает монету и сообщает нам серию результатов. С некоторой вероятностью он может подменить монету. Моменты подмены монеты нам неизвестны, но известно:
 - результаты бросков
 - вероятность с которой он заменяет монету
 - степень кривизны каждой монеты
- Задача: определить моменты смены монеты

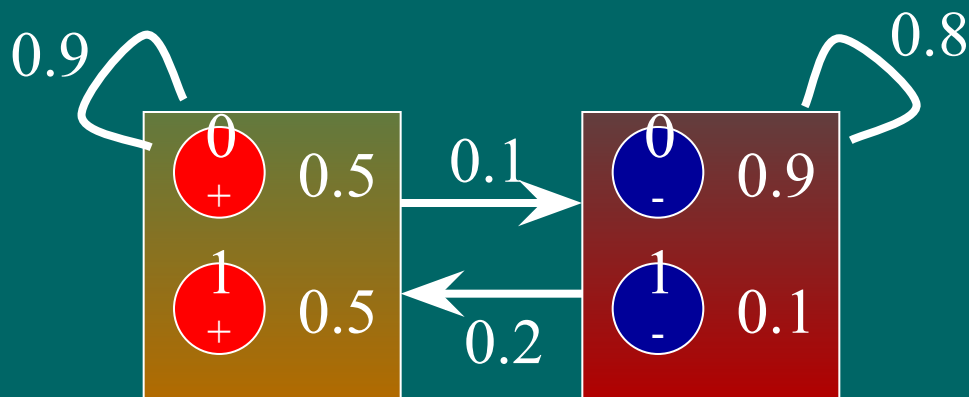
Биологические примеры

- Дана аминокислотная последовательность трансмембранного белка. Известно, что частоты встречаемости аминокислот в трансмембранных и в растворимых частях белка различаются (аналог разных монет). Определить по последовательности где находятся трансмембранные участки.
- Дана геномная последовательность. Статистические свойства кодирующих областей отличаются от свойств некодирующих областей. Найти кодирующие области.
- • • •
- • • •
- • • •

Описание НММ

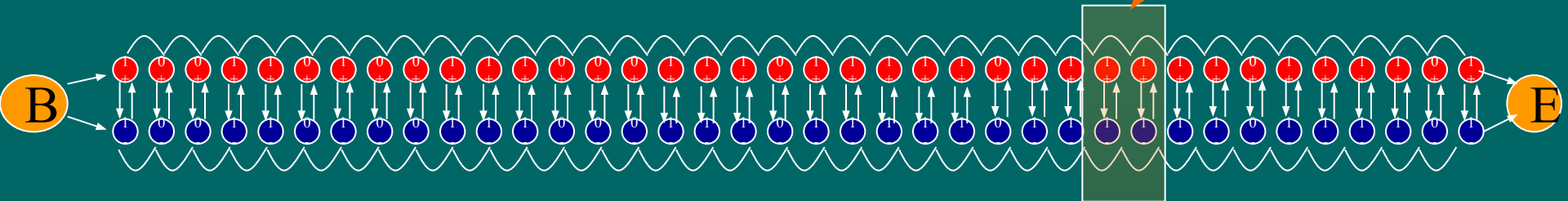
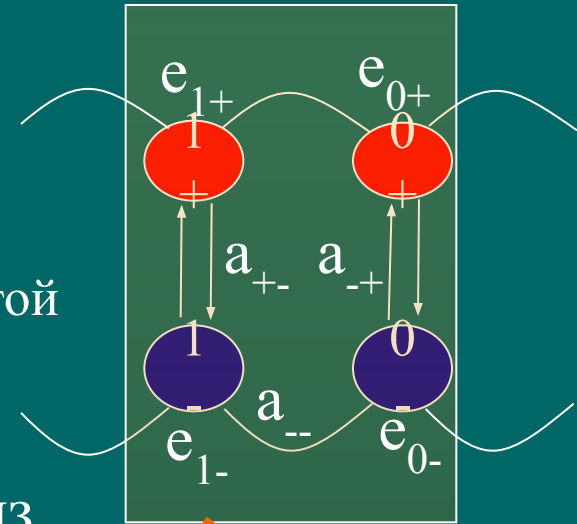
- Пример с монетой можно представить в виде схемы конечного автомата:
 - Прямоугольники означают состояния
 - Кружки означают результат бросания (эмиссии)
 - Стрелки – возможные переходы между состояниями
 - Числа около кружков – вероятности эмиссии e_i
 - числа около стрелок – вероятности переходов между состояниями a_{ik}

- Сумма весов исходящих стрелок равна 1
- Сумма весов эмиссии в каждом состоянии равна 1



Решение задачи о монете

- Пусть нам известна серия бросков:
100110100111000111011111011111011110111101
- Этой серии можно поставить в соответствие граф переходов:
 - Красные вершины соответствуют эмиссии соответствующих значений правильной монетой
 - Синие вершины – эмиссия значений кривой монетой
 - на ребрах – вероятности переходов
 - на вершинах – вероятности эмиссии
- Каждому пути по графу соответствует одна из гипотез о порядке смены монеты



Решение задачи о монете

- Для любого пути можно подсчитать вероятность того, что наблюдаемая серия соответствует этому пути (порядку смены монет)

$$P = a_{0,1} \cdot \prod a_{i,i+1} \cdot e_{i+1}$$

- Найдем путь, отвечающий максимуму P . \log является монотонной функцией, поэтому можно прологарифмировать формулу для вероятности.

$$\pi^* = \operatorname{argmin} \left\{ -\log a_{0,1} - \sum_{\pi} (\log(a_{i,i+1}) + \log(e_{i+1})) \right\}$$

- Это задача поиска оптимального пути на графе. Решается динамическим программированием
- Алгоритм динамического программирования для поиска наиболее вероятного пути называется Viterbi

Viterbi рекурсия

- Обозначения

- $v_k(i)$ – наилучшая вероятность пути, проходящего через позицию i в состоянии k .
- $\pi_k(i)$ – наилучший переход из позиции i в состоянии k в предыдущую позицию (предыдущее состояние)
- $\pi^*(i)$ – наилучшее состояние в позиции i

- Инициация

$$v_k(0) = \delta(0, k); \quad k - \text{номер состояния}$$

- Рекурсия

$$\begin{aligned} v_k(i) &= e_k(x_i) \max_m (v_m(i-1) a_{mk}); \\ \pi(i, k) &= \operatorname{argmax}_m (v_m(i-1) a_{mk}); \end{aligned} \quad \text{обратный переход}$$

- Завершение

$$\begin{aligned} P(x, \pi^*) &= \max_m (v_m(L) a_{m0}); \\ \pi^*(L) &= \operatorname{argmax}_m (v_m(L) a_{m0}); \end{aligned}$$

- Оптимальный путь

$$\pi^*(i-1) = \pi(i, \pi^*(i));$$

Другая постановка задачи

- Для каждого наблюдаемого значения определить вероятность того, что в этот момент монета была правильной.
- Для этого надо просуммировать по всем путям, проходящим через точку i_+ вероятности этих путей. Для решения этой задачи достаточно вспомнить динамическое программирование над полукольцом с использованием операции сложения и умножения.

- Оцениваем значение

$$P(x, \pi_i=k) = P(x_1 \dots x_i, \pi_i=k) \cdot P(x_{i+1} \dots x_L | \pi_i=k) / P(x);$$

- Первый сомножитель $f_k(i) = P(x_1 \dots x_i, \pi_i=k)$ определяем просмотром вперед
- Второй сомножитель $b_k(i+1) = P(x_{i+1} \dots x_L | \pi_i=k)$ определяем просмотром назад

Оценка параметров НММ

- Есть две постановки задачи.
 - Есть множество наблюдений с указанием, где происходит смена моделей (*обучающая выборка, training set*)
 - Есть множество наблюдений, но смена моделей нам не дана
- В обоих случаях предполагается известными сами модели, т.е. конечные автоматы описаны, но неизвестны числа на стрелках и вероятности эмиссии.

Оценка параметров НММ при наличии обучающей выборки

- Здесь используется техника оценки параметров методом наибольшего правдоподобия.
- Пусть
 - x^n – набор независимых наблюдений
 - θ – набор параметров, которые надо оценить
- Тогда надо максимизировать
$$\theta^* = \operatorname{argmax}_{\theta} l(x^1 \dots x^n | \theta) = \operatorname{argmax}_{\theta} \left\{ \sum_j \log P(x^j | \theta) \right\}$$

Оценка параметров НММ при наличии обучающей выборки

- Можно показать, что при большом количестве наблюдений справедливы оценки

$$a_{kl} = A_{kl} / \sum_{l'} A_{kl'} ; \quad e_k(\mathbf{b}) = E_k(\mathbf{b}) / \sum_{\mathbf{b}'} E_k(\mathbf{b});$$

– A_{kl} – наблюдаемое количество переходов между моделями

– $E_k(\mathbf{b})$ – количество порожденных символов в соответствующих моделях

- При малых размерах выборки используют технику псевдоотсчетов, добавляя к наблюдаемым значениям некоторое количество шума.

Если нет обучающей выборки

- Итеративный алгоритм Баума-Велча.
 1. Выберем некоторые наборы параметров НММ (обычно они генерируются случайно).
 2. Найдем для них оптимальные пути во всех представленных примерах
 3. По найденным оптимальным путям определим новые параметры
 4. Перейдем к шагу 2.
- Показано, что алгоритм сходится (отношение правдоподобия растет на каждой итерации)
- Есть опасность нахождения локального, а не глобального экстремума.

Оценки параметров по Бауму-Велчу

- Имея заданные параметры модели можно определить вероятность перехода между состояниями:

$$P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = f_k(i) a_{kl} e_i(x_{i+1}) b_l(i+1) / P(x),$$

где $f_k(i) = P(x_1 \dots x_i, \pi_i = k)$, $b_l(i+1) \cdot P(x_{i+1} \dots x_L \mid \pi_{i+1} = l)$ – значения, полученные при прямом и обратном проходе. Тогда для переходных и эмиссионных вероятностей получим оценки для количества переходов и порожденных символов:

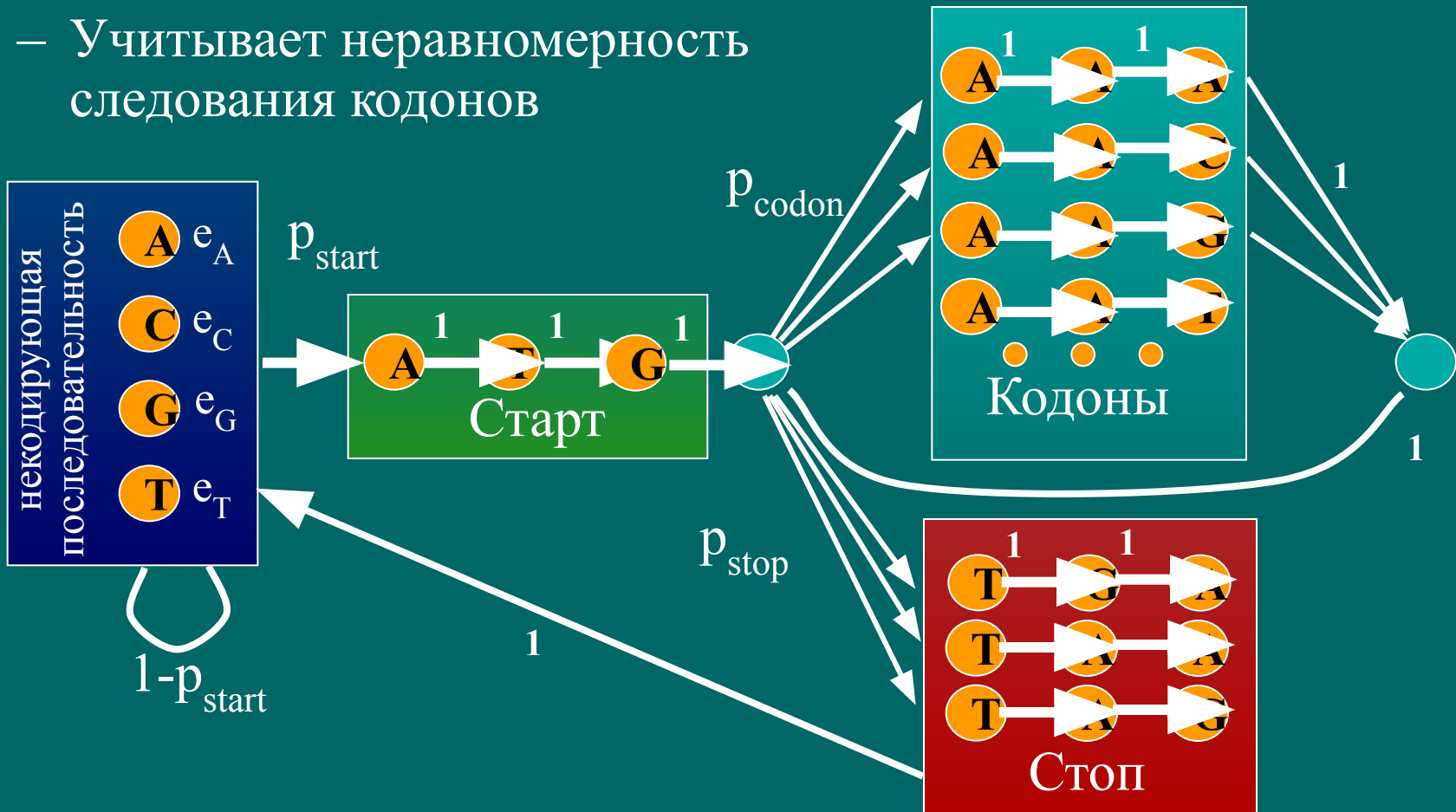
$$A_{kl} = \sum_j 1/P(x^j) \sum_i f_k^j(i) a_{kl} e_i(x_{i+1}) b_l^j(i+1);$$

$$E_k(b) = \sum_j 1/P(x^j) \sum_i f_k^j(i) b_k^j(i) \delta(x_i^j, b);$$

где x^j – j -последовательность в выборке,
 f_k^j , b_l^j – результаты прямого и обратного прохода по последовательности x^j

Предсказание кодирующих областей в прокариотах

- Реальная схема HMM для поиска кодирующих областей сложнее:
 - Включает в себя SD сайт
 - Учитывает неравномерность следования кодонов



Оценка качества обучения

- Выборку разбивают на два подмножества – обучающую и тестирующую
- На первой выборке подбирают параметры
- На второй – тестируют и определяют качество обучения:
 - TP – количество правильно определенных позитивных позиций (например, кодирующих)
 - TN – количество правильно определенных негативных позиций (например, не кодирующих)
 - FP – количество неправильно определенных позитивных позиций (не кодирующих, предсказанных как кодирующие)
 - FN – количество неправильно определенных негативных позиций (кодирующих не кодирующих, предсказанных как не кодирующие)

- Специфичность:

$$Sp = TP / (TP + FP)$$

- Чувствительность:

$$Sen =$$

- Качество

$$QQ =$$

- Коэффициент корреляции

$$CC =$$

Профили

Энтропия колонки

- Пусть колонка содержит n_α букв типа α . Тогда вероятность появления такой колонки при случайных независимых последовательностях будет определяться мультиномиальным распределением:

$$P_{\text{column}} = \frac{N!}{\prod_{\alpha} n_{\alpha}!} \prod_{\alpha} p_{\alpha}^{n_{\alpha}}; p_{\alpha} - \text{вероятность появления } \alpha$$

- Логарифм этой величины равен:

$$\log (P_{\text{column}}) = \log N! + \sum_{\alpha} (n_{\alpha} \log p_{\alpha} - \log n_{\alpha}!)$$

Заменим n на $N f_{\alpha}$ (f_{α} – частота) и применим оценку для факториала $n! \approx (n/e)^n$. Получим полную энтропию колонки

$$H_{\text{column}} = \log(P_{\text{column}}) = N \sum_{\alpha} f_{\alpha} (\log p_{\alpha} - \log f_{\alpha}); \text{ доказать!}$$

Величина

$$I = - \sum_{\alpha} f_{\alpha} (\log p_{\alpha} - \log f_{\alpha})$$

называется информационным содержанием колонки

НММ профиль

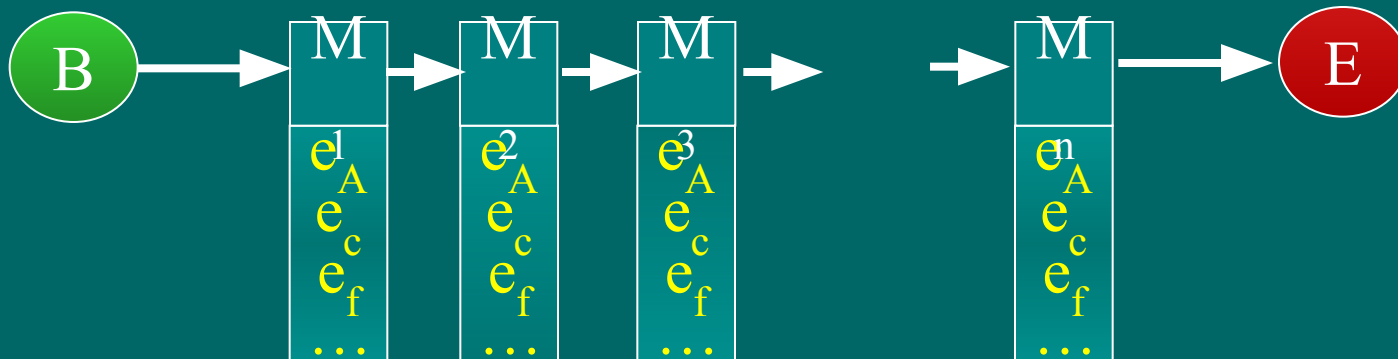
- Модель: каждая последовательность множественного выравнивания является серией скрытой Марковской модели.
- Профиль – описание Марковской модели. Каждой позиции соответствует свое состояние. Вероятности переходов между соседними состояниями равны 1.
- Вероятность того, что некоторая последовательность x соответствует профилю M :

$$P(x | M) = \prod e_i(x_i);$$

- Значимость определяется отношением правдоподобия: сравнением с $P(x | R)$ – вероятностью, что последовательность сгенерирована случайной моделью R :

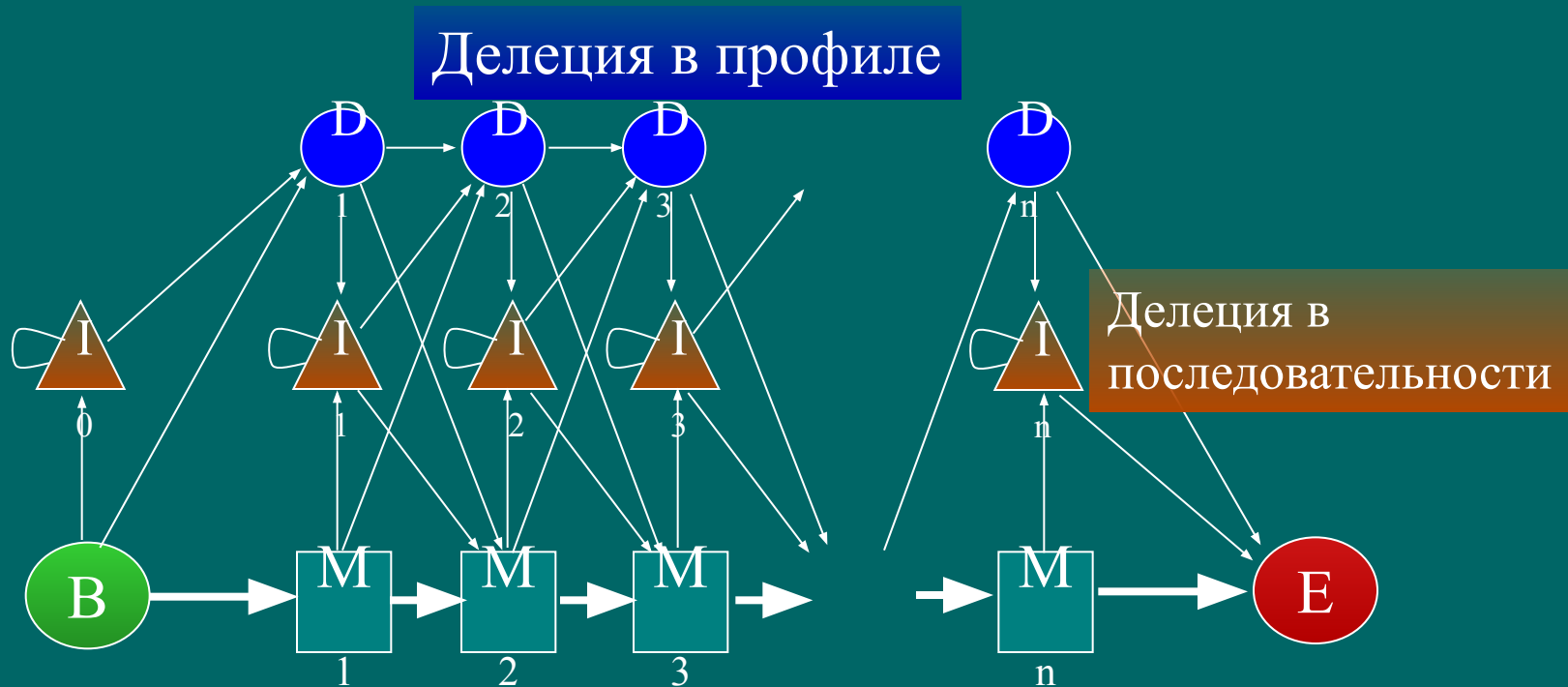
$$S = \log (P(x | M) / P(x | R)) = \sum \log \{e_i(x_i) / q(x_i)\};$$

- Величины $w_i(\alpha) = \log \{e_i(\alpha) / q(\alpha)\}$ называют позиционной весовой матрицей (PSSM)



НММ с учетом возможности вставок

- Делеция в профиле и в последовательности могут идти подряд (в отличие от парного выравнивания)
- Делеционные состояния – молчание (не имеют эмиссии)
- Вероятность перехода в делеционное состояние зависит от позиции



Определение параметров модели

- Для начала надо определиться с длиной модели. В случае, если обучающее множественное выравнивание не имеет вставок/делеций это тривиально. Наличие же вставок/делеций требует различать вставки и делеции. Простейшее правило если колонка содержит больше половины вставок, то она не включается в модель, а события вставок трактуются как вставки в последовательность.
- Если выравнивание толстое, то для параметров можно использовать обычные оценки:

$$a_{kl} = A_{kl} / \sum_{l'} A_{kl'} ; e_k(a) = E_k / \sum_{a'} E_k(a');$$

Для тонких выравниваний

- Простейшие варианты псевдоотсчетов:

- Правило Лапласа: к каждому счетчику прибавить 1:

$$e_k(a) = (E_k(a) + 1) / (\sum_{a'} E_k(a') + N_a);$$

где N_a – размер алфавита (20)

- Добавлять псевдоотсчеты, пропорционально фоновым частотам:

$$e_k(a) = (E_k(a) + Aq_a) / (\sum_{a'} E_k(a') + A); A \approx N_a;$$

Такие псевдоотсчеты соответствуют Байесовой оценке

$$P(\theta | D) = P(D | \theta) P(\theta) / P(D);$$

при априорном распределении $P(\theta)$ – распределение Дирихле с параметром $\alpha_a = Aq_a$.

Смеси Дирихле

- Представим себе, что на распределение вероятностей влияют несколько источников — частота встречаемости символа в белках вообще, частота встречаемости символа в петлях, частота встречаемости символа в трансмембранных сегментах и т.п. Каждое такое распределение дает свои псевдоотсчеты α^k . Тогда для вероятности эмиссии можно написать:

$$e_k(a) = \sum_d P(d | E_k) (E_k(a) + \alpha_a^d) / (\sum_{a'} E_k(a') + \sum_a \alpha_a^d);$$

где $P(d | E_k)$ — вероятность выбора распределения d при условии наблюдаемых частот:

$$P(d | E_k) = P(E_k | d) P(d) / \sum_{d'} P(E_k | d') P(d');$$

- Для оценки $P(E_k | d)$ используют простую формулу:

$$P(E_k | d) = \frac{(\sum_a E_k(a))! \Gamma(\sum_a (E_k(a) + \alpha_a^d)) \Gamma(\sum_a \alpha_a^d)}{\prod_a E_k(a)! \prod_a \Gamma(E_k(a) + \alpha_a^d) \prod_a \Gamma(\alpha_a^d)}$$

Использование матрицы замен

- Еще один способ введения псевдоотсчетов. У нас есть матрица замен аминокислотных остатков (например, РАМ120). Матрица замен может трактоваться как то, что каждая аминокислота является немножко другой аминокислотой. Поэтому в качестве псевдоотсчетов используют величину

$$\alpha_{ja} = A \sum_b f_{ib} P(a | b),$$

где f_{ib} – частота встречаемости в колонке буквы b , $P(a | b)$ – вероятности замены буквы b на a

Использование предка

- Все последовательности x^k в выравнивании произошли от общего предка y .

$$P(y_j=a \mid \text{alignment}) = q_a \prod_k P(x_j^k \mid a) / \sum_{a'} q_{a'} \prod_k P(x_j^k \mid a')$$

- Тогда для оценки эмиссионной вероятности

$$e_j(a) = \sum_{a'} P_j(a \mid a') P(y_j=a' \mid \text{alignment})$$

где $P_j(a \mid a')$ – матрица замен. Матрица замен зависит от скорости эволюции соответствующей колонки.

Для выбора матрицы можно использовать принцип максимального правдоподобия:

$$P(x_j^1, x_j^2, \dots, x_j^N) = \sum_{a'} q_{a'} \prod_k P(x_j^k \mid a, t) \rightarrow \max ;$$

- Для матрицы замен можно использовать выражение:

$$P(a \mid b, t) = \exp(t P(a \mid b, 1))$$

А чему же равно A ?

- Для компенсации малости выборок используют псевдоотсчеты.
- Разные подходы дают разные распределения псевдоотчетов α_i , но не определяют величину коэффициента A при α_i .
- Часто предполагают, что псевдоотсчеты должны быть сопоставимыми с точностью определения частот Δ , которая пропорциональна $\Delta \approx \sqrt{N}$, где N – количество испытаний (толщина выравнивания) поэтому полагают:

$$A = \kappa \sqrt{N}, \kappa \approx 1 (0.5 \dots 1);$$

Это еще не все ...

- При вычислении эмиссионных вероятностей используется предположение о независимости испытаний. Однако, в выравнивании часто встречаются близкие последовательности, и это предположение неверно. Например, если мы в выравнивание добавим много копий одной из последовательностей, то эмиссионные вероятности будут в основном отражать свойства именно этой последовательности.
- Пример: выравнивание содержит последовательности белка из человека, шимпанзе, гиббона, орангутанга, мыши, рыбы, мухи, комара, червяка. Очевидно, что последовательности приматов перепредставлены. Кроме того, последовательности двукрылых также перепредставлены.
- Поэтому при подсчете вероятностей необходимо каждую последовательность учитывать с весом, отражающим ее уникальность в данной выборке.

Взвешивание последовательностей

- Способ учета неравномерной представленности последовательностей в выборке называется взвешиванием последовательностей.
- Каждой последовательности в выравнивании присваивается свой вес β_k . Тогда частота каждого символа a в колонке k подсчитывается по формуле:

$$E_k^a = \sum_i \beta_i \delta(S_k^i, a) / \sum \beta_i$$

где S_k^i – буква в последовательности i в колонке k , β_i – вес последовательности i .

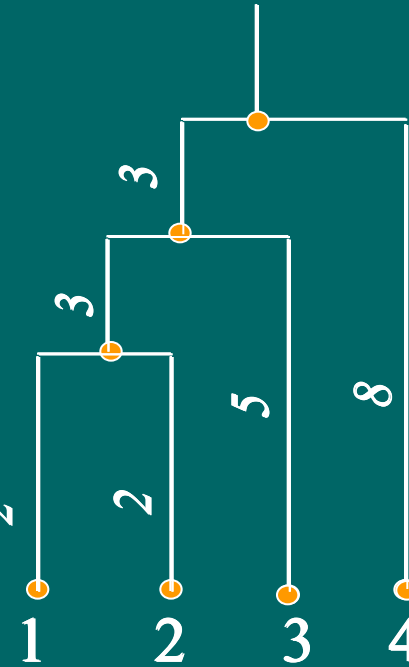
Взвешивание последовательностей

Метод Герштейна-Сонхаммера-Чотьи

- Пусть нам известно филогенетическое дерево с расстояниями на ветвях. На листьях – последовательности.
- В начале все веса последовательностей приравниваются длинам веток
- Далее веса определяем итеративно, внося поправки в веса по ходу движения вверх по дереву:

$$\Delta w_i = t_n w_i / \sum_{k\text{-листья ниже узла } n} w_k$$

- Смысл заключается в том, что длина ветки распределяется по дочерним узлам



$$w_1 = 2 + 3 + 2 = 3.5 / 12 = 4.4$$

$$w_2 = 2 + 3 + 2 = 3.5 / 12 = 4.4$$

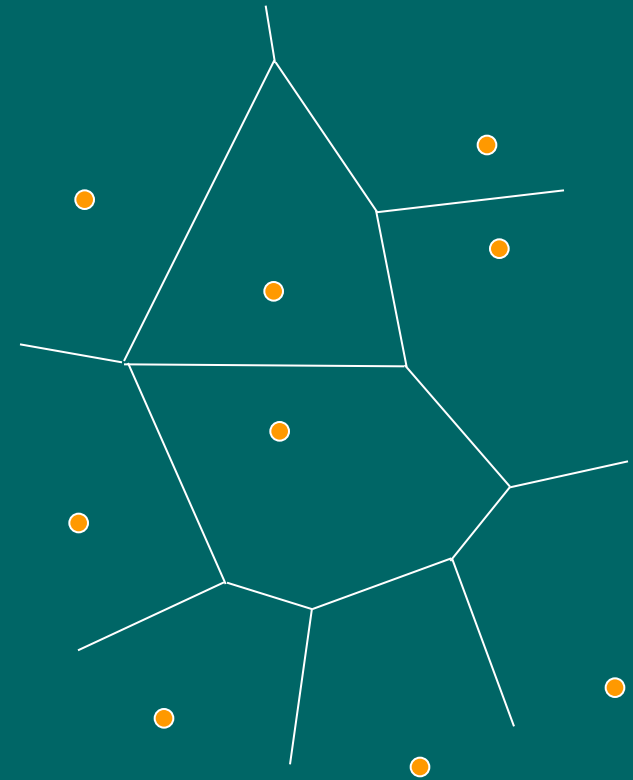
$$w_3 = 5 + 2.5 = 5 / 12 = 6.25$$

$$w_4 = 8$$

Взвешивание последовательностей

Многогранники Воронова

- Поместим объекты в некоторое метрическое пространство. Каждый объект хочет иметь "поместье" – некоторую область пространства. Проведем границы между поместьями посередине между объектами. В результате все "поместья" будут иметь форму многогранника. Эта конструкция называется многогранниками Воронова.
- Можно определить вес последовательности как объем поместья. Вопрос только в том как и в какое метрическое пространство помещать последовательности.



Взвешивание последовательностей

Максимально дискриминирующие веса

- Вероятность модели при условии, что последовательность x принадлежит модели M :

$$P(M | x) = \frac{P(x | M) P(M)}{P(x | M) P(M) + P(x | R)(1-P(M))}$$

- Вероятность модели при условии нескольких последовательностей (дискриминатор):

$$D = \prod_k P(M | x^k)$$

- Веса последовательностей подбираем так, чтобы максимизировать D .
- Но: чтобы вычислить D нам необходимы параметры модели, которые зависят от того, как мы взвешивали последовательности. Применяют итеративную процедуру.

Взвешивание последовательностей

Максимизация энтропии

- Пусть $k(i, a)$ – количество остатков типа a в колонке i , m_i – количество типов остатков в колонке i . Выберем вес для последовательности k равным

$$w_k(i) = 1 / (m_i k(i, a)).$$

- Такой вес обеспечивает наиболее равномерное распределение частот остатков в колонке. Чтобы задать вес для последовательности в целом, просуммируем соответствующие веса:

$$w_k = \sum_i w_k(i) = \sum_i 1 / (m_i k(i, a)).$$

Взвешивание последовательностей

Максимизация энтропии

- Обобщенный подход:

$$\sum_i H_i(w) \rightarrow \max, \sum_k w_k = 1;$$

где $H_i(w) = \sum_a p_{ia} \log p_{ia}$;

p_{ia} – вероятности встречаемости аминокислоты a в колонке i , подсчитанные с учетом весов последовательностей:

$$p_{ia} = \sum_k w_k \delta(x_i^k, a);$$

- Задача максимизации приводит к системе уравнений:

$$\sum_k w_k = 1;$$

$$\sum_i \partial H_i(w) / \partial w_k - \lambda = 0;$$

- Здесь неизвестные w_k и неопределенный множитель Лагранжа λ

Множественное выравнивание

Множественное выравнивание

- Способ написать несколько последовательностей друг под другом (может быть с пропусками) так, чтобы в одной колонке стояли гомологичные позиции.
- "Золотой стандарт" – совмещенные пространственные структуры гомологичных белков. Соответствующие позиции в разных последовательностях отвечают гомологичным позициям
- Задача. Найти способ (алгоритм и параметры), выравнивающий последовательности "золотого стандарта" правильно. Есть надежда, что в случаях, когда пространственные структуры неизвестны, этот алгоритм правильно выровняет последовательности.

Оценка качества множественного выравнивания

Энтропийная оценка

- Обычно считают, что колонки в выравнивании независимы. Поэтому качество выравнивания можно оценить как сумму качеств колонок:

$$S = G + \sum_{\text{columns}} S(m_k)$$

G – веса делеций, $S(m_k)$ – вес колонки

- Пусть c_{ia} – количество появлений аминокислоты a в колонке i . Вероятность колонки можно описать как
- $P(m_i) = \prod_a p_{ia}^{c_{ia}}$
- Вероятность выравнивания = $\prod_i P(m_i)$; В качестве веса можно использовать логарифм вероятности:

$$S = \sum_{\text{columns}} S(m_k);$$
$$S(m_k) = - \sum_a c_{ia} \log p_{ia} = H(m_i)$$

$H(m_i)$ – энтропия колонки; для вероятностей остатков принимают:

$$p_{ia} = \tilde{c}_{ia} / \sum_{a'} \tilde{c}_{ia'}$$

где \tilde{c}_{ia} – количество остатков в колонке с поправкой на псевдоотсчеты

Оценка качества множественного выравнивания

Сумма пар

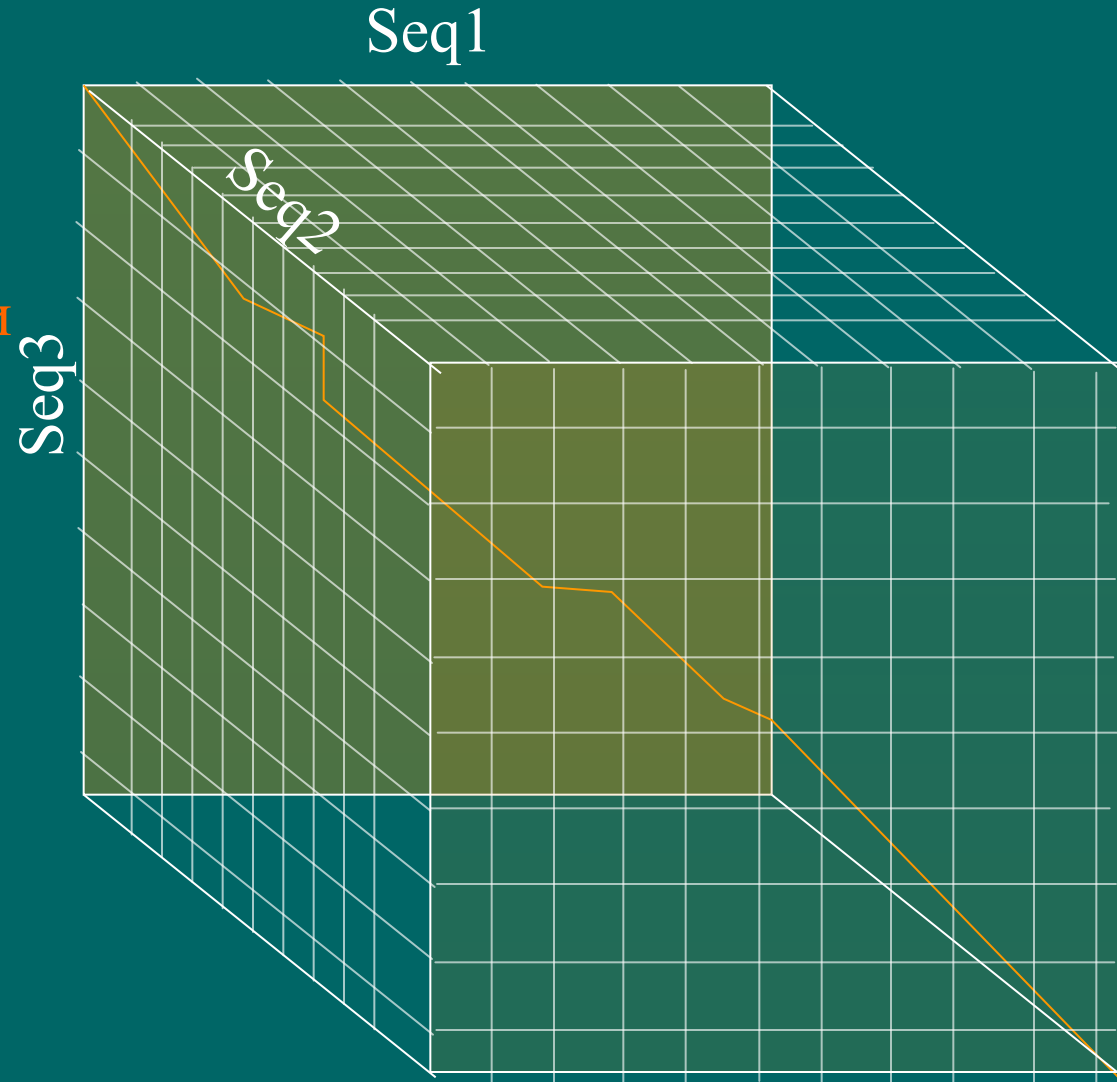
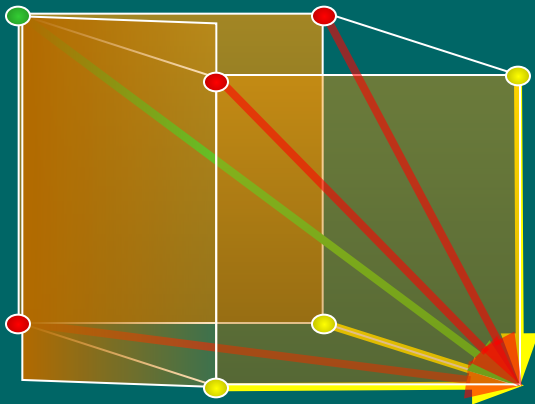
- Другой традиционный способ оценки – сумма весов матрицы соответствия аминокислотных остатков SP:

$$S(m_i) = \sum_{k < l} s(x_i^k, x_i^l);$$

- Способ не совсем правильный. Более правильная оценка для трех последовательностей $S(m_i) = \log(p_{abc} / q_a q_b q_c)$, а не $\log(p_{ab} / q_a q_b) + \log(p_{bc} / q_b q_c) + \log(p_{ac} / q_a q_c)$; (вспомним определение матрицы замен)

Если есть функционал, то его надо оптимизировать

- Элементарные переходы:
 - Сопоставление трех
 - Сопоставление двух и одна делеция
 - Делеция в двух последовательностях

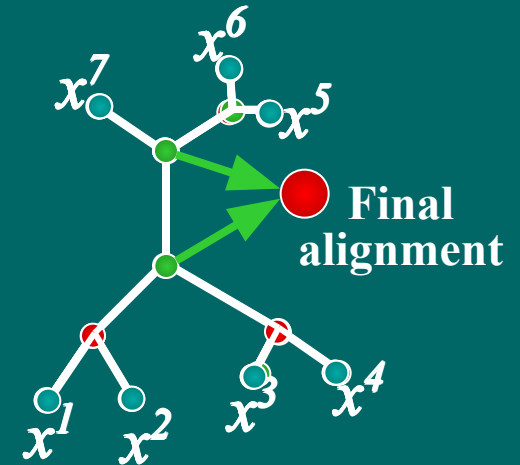


Динамическое программирование для множественного выравнивания

- Количество вершин равно $\prod_{\text{послед.}} L_i = O(L^N)$
- Количество ребер из каждой вершины = $2^N - 1$
(почему?)
- Количество операций равно
$$T = O(L^N)$$
- Надо запоминать обратные переходы в L^N вершинах.
- Если количество последовательностей > 4 , то задача практически не разрешима.

Прогрессивное выравнивание

- Строится бинарное дерево (guide tree, путеводное дерево) – листья = последовательности
- Дерево обходится начиная с листьев. При объединении двух узлов строится парное выравнивание супер-последовательностей (профилей) и получается новая суперпоследовательность



Путеводное дерево строится приближенно – главное быстро. Обычно это кластерное дерево

Выравнивание профилей

- Выравнивание одной стопки последовательности относительно другой – обычное динамическое программирование.
- Оптимизируется сумма парных весов:

$$\sum_i S(m_i) \rightarrow \max$$
$$S(m_i) = \sum_{k < l \leq N} s(x_i^k, x_i^l)$$

- Если мы выравниваем две стопки – $0 < i \leq n$ и $n < i \leq N$, то сумму разбиваем на три части:

$$S(m_i) = \sum_{k < l \leq n} s(x_i^k, x_i^l) + \sum_{\substack{n < k < l \leq N \\ k < l \leq N}} s(x_i^k, x_i^l) + \sum_{k \leq n, n < l \leq N} s(x_i^k, x_i^l)$$

- Две первые суммы являются внутренним делом стопок, последняя сумма отвечает за сравнение стопок (профилей)
- При сравнении используем расширенную матрицу сходства, добавив в нее сравнение делеционного символа '-':

$$s(-, -) = 0, s(a, -) = -d;$$

- При множественном выравнивании обычно используют линейные штрафы за делеции

ClustalW

1. Строится матрица расстояний с использованием попарных выравниваний
2. Строится NJ дерево (метод ближайшего соседа)
3. Строится прогрессивное выравнивание
 - Используются дополнительные эвристики:
 - Взвешивание последовательностей (с учетом только топологии дерева)
 - На разных уровнях дерева используются разные матрицы сходства
 - Используется контекстно-зависимые штрафы за открытие делеции
 - Если при построении выравнивания появляются очень низкие веса, то дерево корректируется

Улучшение выравнивания

- Недостаток прогрессивных методов: если для некоторой группы последовательностей выравнивание построено, то оно уже не перестраивается.
- Алгоритм итеративного улучшения
 1. Вынимаем из выравнивания одну последовательность
 2. По оставшимся последовательностям строим профиль
 3. Выравниваем вынутую последовательность с профилем
 4. Переходим к этапу 1.

Множественное выравнивание с помощью НММ

- Каждому множественное выравнивание соответствует скрытая Марковская модель.
- Можно применить алгоритм максимизации ожидания Баума-Велча:
 - Порождаем случайные параметры НММ.
 - Выравниваем все последовательности с этой моделью
 - Переоцениваем параметры.
- Проблема: легко попасть в локальный максимум
- Обход проблемы: время от времени параметры НММ возмущаются.
- Другой вариант – использование искусственного отжига.
- Достоинство подхода: одновременно анализируются все последовательности. Нет проблемы необратимости, характерной для прогрессивного выравнивания.

Блочное выравнивание

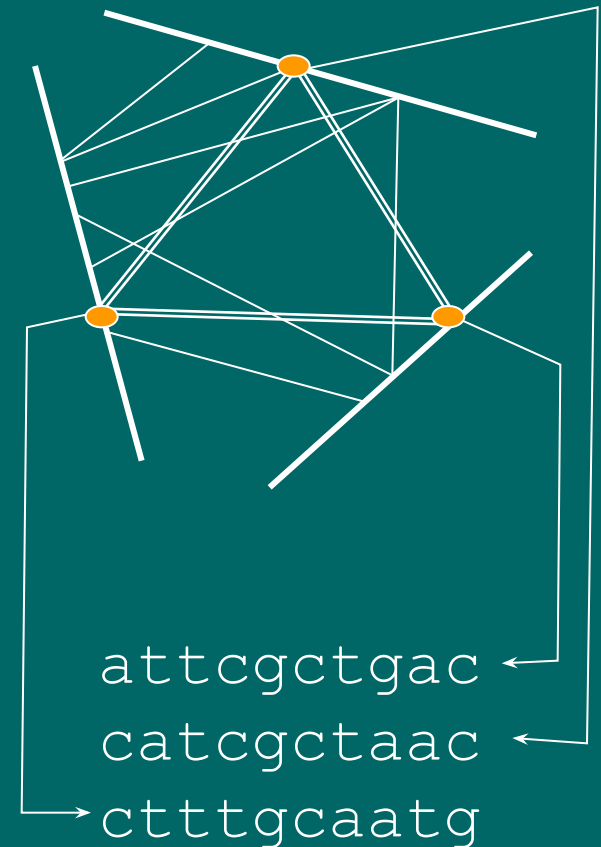
Поиск СИГНАЛОВ

Постановка задачи

- Дано несколько (например, 20) последовательностей. Длина каждой последовательности - 200
- В каждой последовательности найти короткий (длиной 20) фрагмент (сайт), такой, что все сайты между собой похожи.
- Например, даны регуляторные области совместно регулируемых генов. Найти сайты связывания белков-регуляторов.

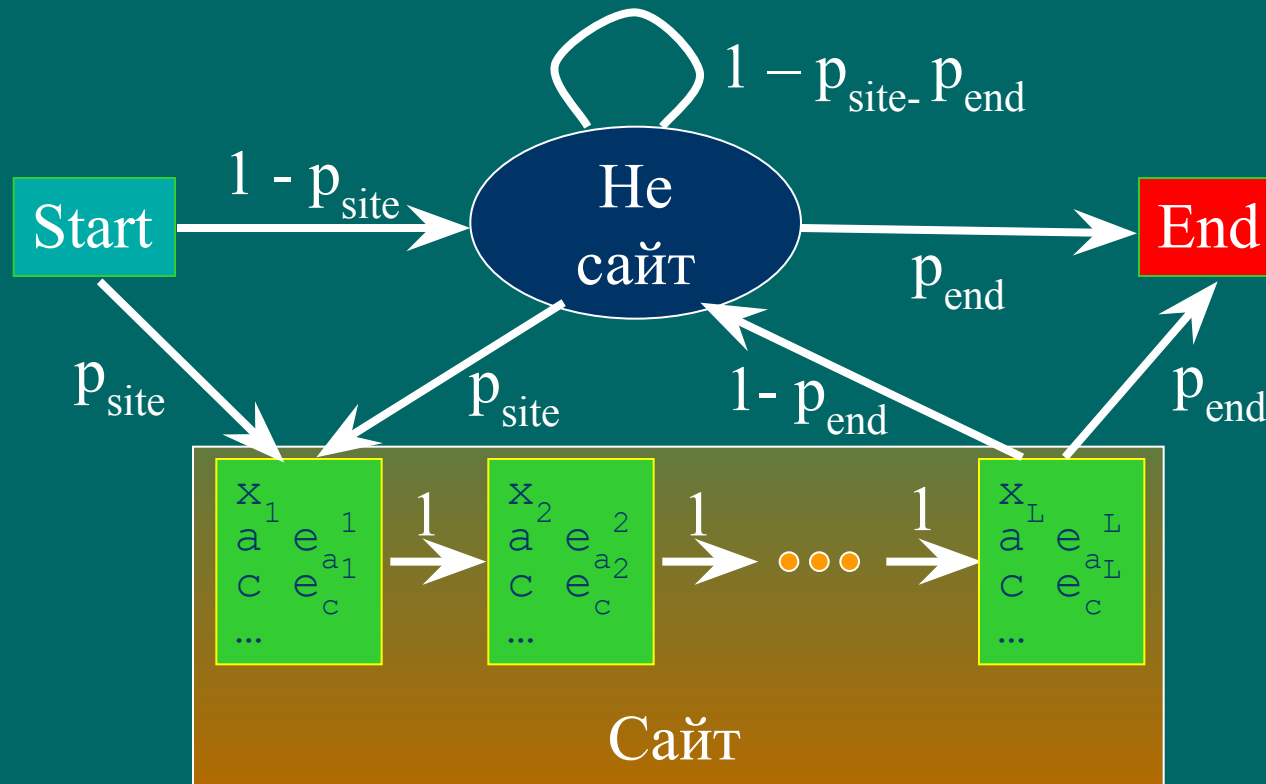
Графвая постановка задачи.

- Дан многодольный граф:
 - Каждой доле соответствует последовательность
 - Вершины – сайты
 - Ребра проводятся между всеми сайтами, или если эти сайты между собой похожи.
- На каждой клике графа определено число. Например, информационное содержание безделеционного множественного выравнивания сайтов
- Найти клику наибольшего веса



НММ-постановка задачи

- Найти НММ, описывающую наилучший сайт.
- Для описания сайта используют следующую модель:



Алгоритм максимизации ожидания

- Допустим нам приблизительно известна структура сайта.
- Применяем алгоритм Баума-Велча.
- Получаем структуру сайта.
- Алгоритм MEME:
 - В качестве исходной модели выбираем модель, индуцированную первым словом в первой последовательности (с учетом псевдоотсетов).
 - Находим НММ
 - Берем в качестве исходной следующее слово из первой последовательности.
 - Так перебираем *все* слова во *всех* последовательностях
 - Отбираем наилучшие НММ

Гиббс сэмплер

- **Задача:** найти набор позиций сайтов в последовательностях
- **Инициация:** В качестве решения выбираем произвольный набор позиций.
- **Итерации:**
 - Удаляем из выборки одну последовательность.
 - По позициям, определенным для остальных последовательностей строим профиль (HMM).
 - Для каждой позиции в удаленной последовательности рассчитываем вероятность того, что сайт находится там.
 - Разыгрываем позицию сайта в удаленной последовательности в соответствии с рассчитанными вероятностями.
 - Повторяем процедуру много раз для всех последовательностей

Вероятности для Гиббс сэмплера

- Вероятности для Гиббс сэмплера

Комбинаторные методы

RNA

Вторичная структура РНК

- Вторичной структурой называется совокупность спаренных оснований
- Биологическая роль вторичной структуры:
 - Структурная РНК –
 - рибосомная,
 - тРНК
 - Регуляция –
 - Рибопереключатели
 - аттенуация
 - микроРНК
 - Рибозимы
 - Стабильность РНК

Элементы вторичной структуры



Способы представления вторичных структур

Топологическая
схема



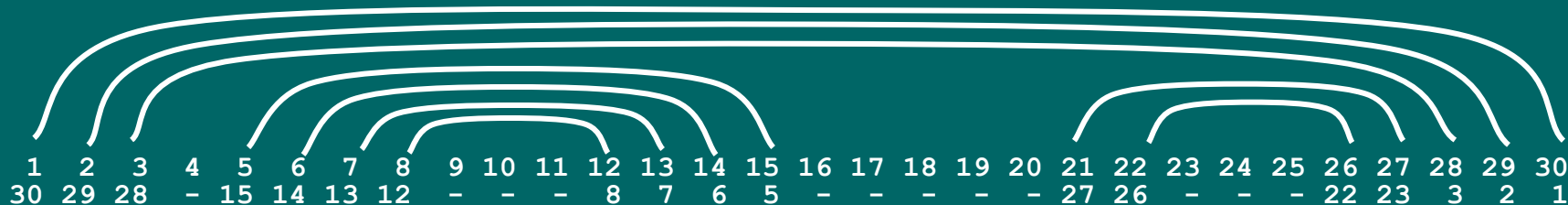
Круговая диаграмма



Список
спиралей

	from ₁	to ₁	from ₂	to ₂
A	1	3	28	30
B	5	8	12	15
C	21	22	26	27

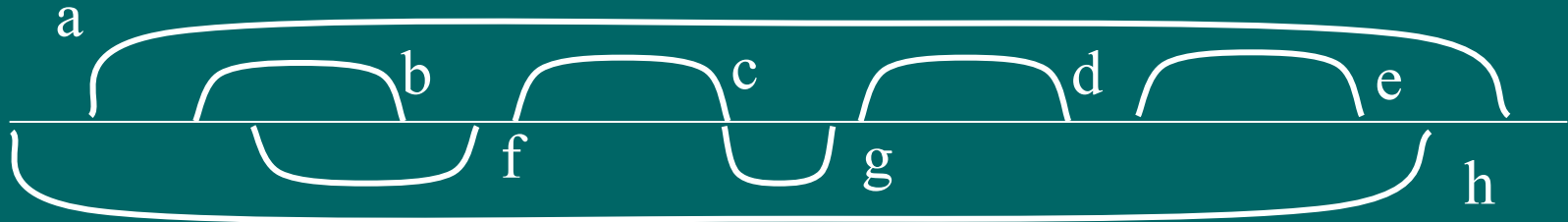
Массив спаренных оснований



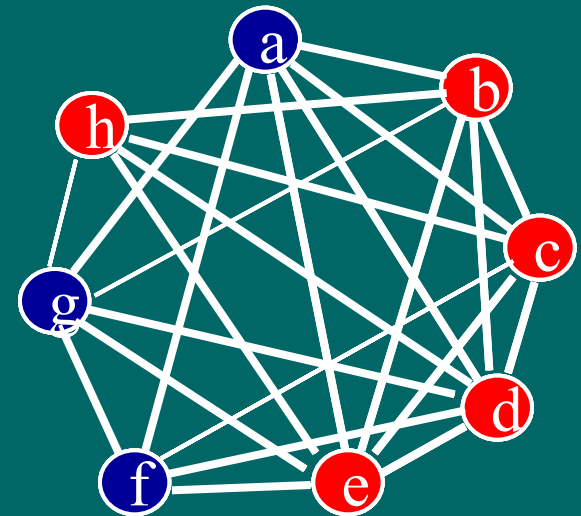
Задача

- Дана последовательность.
- Найти правильную вторичную структуру.
- Золотой стандарт: tРНК, рРНК.
- Количество возможных вторичных структур очень велико.
- Дополнительные ограничения:
 - Нет псевдоузлов. (На самом деле они очень редки и энергетически невыгодны)
- Количество возможных структур все равно очень велико
- Надо найти *оптимальную* структуру. А что оптимизировать? Как оптимизировать?

Комбинаторный подход



- Построим граф:
 - вершины – потенциальные нуклеотидные пары (или потенциальные спирали)
 - Ребро проводится, если пары совместимы (не образуют псевдоузлов и не имеют общих оснований)
- Допустимая вторичная структура – клика в ЭТОМ графе



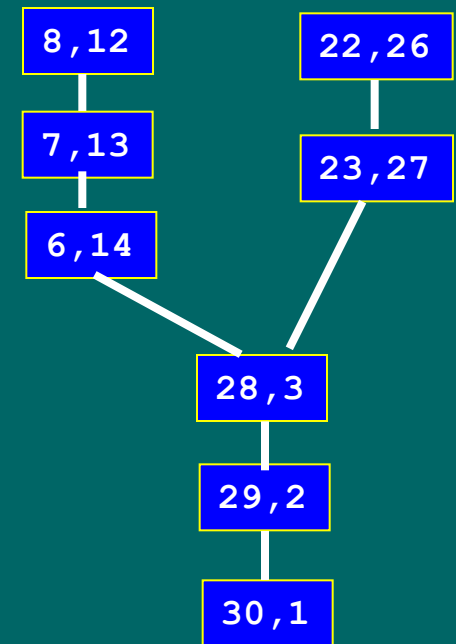
Структуры без псевдоузлов



- Структура без псевдоузлов = правильное скобочное выражение
- Может быть представлено в виде дерева
- Оценка количества возможных структур:

$$T(L) \approx 1.8^L$$

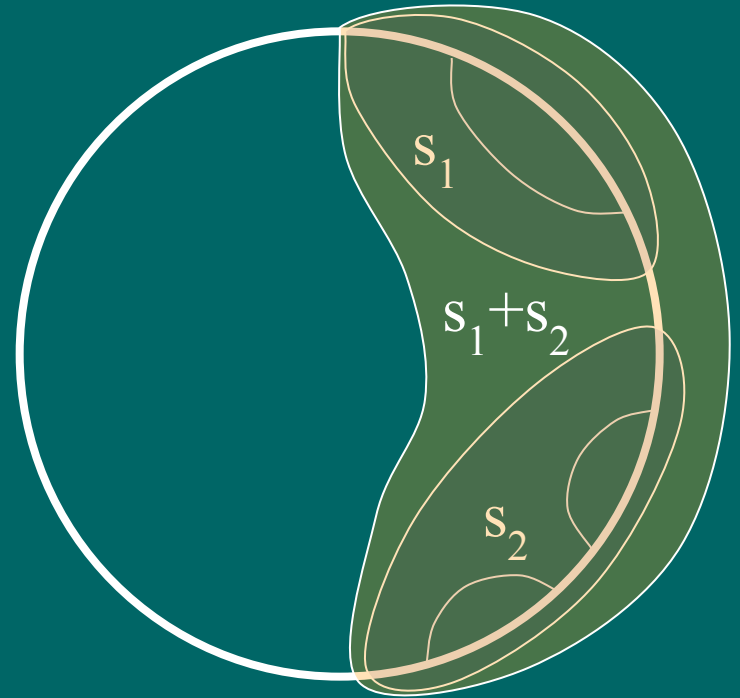
(очень много)



Оптимизация количества спаренных оснований

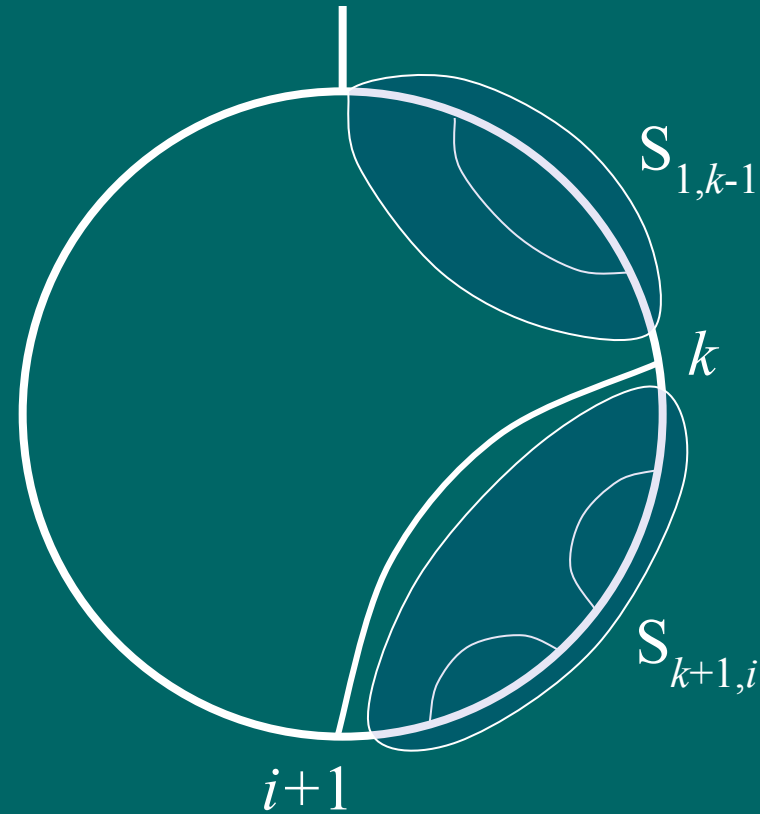
- Обозначим $|s|$ - мощность структуры (количество спаренных оснований)
- Пусть s_1 и s_2 две непересекающиеся структуры (структуры без общих оснований)
- Тогда

$$|s_1 + s_2| = |s_1| + |s_2|$$



Оптимизация количества спаренных оснований

- Пусть нам известны оптимальные структуры S_{rt} для всех фрагментов $i \leq r \leq t \leq j$
- Тогда можно найти оптимальную структуру для сегмента $[i, j+1]$
- Для этого нам надо понять, спаривать ли основание $j+1$, и, если спаривать, то с кем



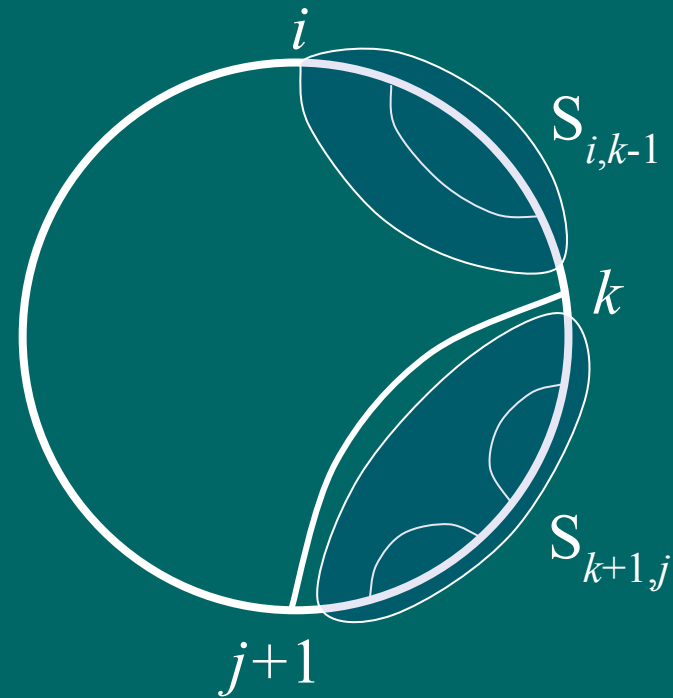
Динамическое программирование для количества спаренных оснований (Нуссинофф)

- Количество спаренных оснований в оптимальной структуре $S_{i,j+1}^*$ определяется как максимум:

$$S_{i,j+1}^* = \max \left\{ \begin{array}{l} S_{i,j}^* \text{; (нет спаривания)} \\ \max_k (S_{i,k-1}^* + S_{k,j}^*) + 1; \\ \text{(} k \text{ спаривается с } j+1) \end{array} \right\};$$

Время работы алгоритма:

$$T \approx O(L^3)$$



Динамическое программирование для количества спаренных оснований

- При поиске оптимального количества спаренных оснований заполняется треугольная матрица весов $S_{i,j}$, $i < j$.
- Обозначим π_{ij} – номер основания, с которым надо спарить основание j при анализе сегмента $[i, j]$, или 0, если не надо спаривать. При оптимизации запоминаем треугольную матрицу спаривания (аналог матрицы обратных переходов)

Восстановление структуры по матрице спаривания

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	-																
2		-															
3			-														
4				-													
5					-												
6						-											
7					5		-										
8				4				-									
9						6			-								
10										-							
11											-						
12				-	10	10	10	10	10	10		-					
13				9	9	9	9	9	9				-				
14														11	11	11	
15																-	
16	2	2															-
17																	-

Восстановление структуры по матрице спаривания

```
SearchStruct (int i, int j)
{
    int i0=i, j0=j;
    do{
        if(i >= j) return;
        if( $\pi_{ij}$  == 0) j--;
        if( $\pi_{ij}$  != i) i++;
        if( $\pi_{ij}$  == i)
        {
            StorePair(i, j);
            SearchStruct (i0, i-1);
            SearchStruct (i+1, j0);
            return;
        }
    }while (true)
}
```

Энергия вторичной структуры

- Энергия спиралей
- Энергия петель (энтропия)

Энергия спирали рассчитывается как сумма энергий стэкингов

	AU	CG			
AU	-2	-3.2			
CG	-3.2	-4.8			
GC	-3.7	-4.5			

A – U
C – G
A – U
G – C
C – G

$$\Delta G = -3.2 -3.2 -3.7 -4.5$$
$$= - 14.6$$

Энергия петель

- Энергия свободной цепи

$$\Delta G = V + 3/2 kT \ln L$$

- Для шпилек при $L=3..5$ кроме энтропии есть некоторое напряжение структуры.
- Для внутренних петель и для мультипетель L – суммарная длина петель + количество ветвей.
- Параметр V зависит от типа петли
- Для выпячивания сохраняется стэкинг.
- Обычно используют не формулу, а таблицы.

Минимизация энергии

Обычное динамическое программирование не проходит – нет аддитивности.

- Определения

- нуклеотид h называется доступным для пары $i \cdot j$, если **НЕ** существует спаривания $k \cdot l$, такого, что

$$i < k < h < l < j$$

- Множество доступных нуклеотидов для пары $i \cdot j$ называется петлей L_{ij} , а пара $i \cdot j$ называется закрывающей парой. Частный случай петли – стэкинг.

- Энергия структуры рассчитывается как сумма энергий петель (в том числе и стекингов):

$$\Delta G = \sum e(L_{ij})$$

Алгоритм Зукера

- Введем две переменные:
 - $W(i,j)$ – минимальная энергия для структуры на фрагменте последовательности $[i, j]$;
 - $V(i,j)$ – минимальная энергия для структуры на фрагменте последовательности $[i, j]$ при условии, что i и j спарены;

- Рекурсия:

$$V(i,j) = \min_{i \leq i_1 < j_1 < i_2 < j_2 < \dots < i_k < j_k \leq j} \sum_l^k V(i_l, j_l)$$

$$W(i,j) = \min \{ \quad W(i+1,j), \quad i \text{ не спарено}$$

$$W(i,j-1), \quad j \text{ не спарено}$$

$$V(i,j), \quad i \text{ и } j \text{ спарены}$$

$$\min_{i < k < j} (W(i,k) + W(k+1,j)) \}$$

i и j спарены с кем-то.

Алгоритм Зукера

- Рекурсия для W требует времени

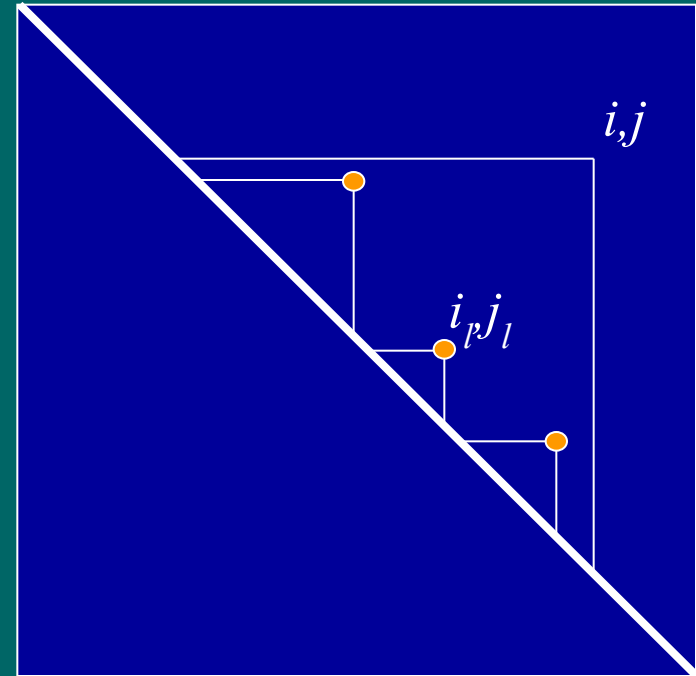
$$T \approx O(L^3)$$

- Рекурсия для V требует гораздо большего времени

$$T \approx O(2^L)$$

- Причина – мультипетли. Можно:

- Ограничить размер или индекс мультипетель
- Применить упрощенную формулу для их энергии
- Просматривать мультипетли только если $i+1, j-1$ не спарены.
- Применить приближенную эвристику



Проблемы минимизации энергии

1. Только около 80% тРНК сворачиваются в правильную структуру
2. Энергетические параметры определены не очень точно. Более того, в клетке бывают разные условия, и, соответственно, реализуются разные параметры.
3. Находится единственная структура с минимальной энергией, в то время как обычно существует несколько структур с энергией, близкой к оптимальной.

Решение проблем

- Искать субоптимальные структуры
- Искать эволюционно консервативные структуры.
 - структуры тРНК и рРНК определены именно так

Поиск субоптимальных структур и структурных элементов

- Статистическая сумма

$$Z = \sum \exp(-\Delta G_i / kT)$$

- Если мы просуммируем по всем структурам, содержащим данную пару, то мы можем оценить ее значимость (чем Z больше, тем более значимым является спаривание)
- Для подсчета Z можно использовать тот же алгоритм динамического программирования, заменив \min на суммирование, а сложение на умножение.
- Больцмановская вероятность того, что нуклеотиды i, j спарены равна:

$$P(i, j) = \exp(-\Delta G_{ij} / kT) / Z_{ij};$$

- Разыграваем пары оснований в соответствии с этой вероятностью и восстанавливаем соответствующие субоптимальные вторичные структуры.

**Консенсусные
вторичные
структуры РНК**

Основные задачи

- Построение консенсуса

- Дано: набор последовательностей для которых известно, что они имеют общую вторичную структуру (например, тРНК или регуляторный элемент)
- Описать общую структуру

- Поиск консенсуса

- Дано: описание консенсуса.
- Найти в данной последовательности (например, в геноме) все случаи встречи консенсуса

Метод ковариаций

- Пусть дано множественное выравнивание последовательностей
- Взаимная информация двух колонок:

$$I(A,B) = \sum_{\alpha\beta} f_{AB}(\alpha\beta) \log_2 \{f_{AB}(\alpha\beta) / (f_A(\alpha) f_B(\beta))\}$$

$f_{AB}(\alpha\beta)$ – частоты одновременной встречи буквы α в колонке A и буквы β в колонке B .

$f_A(\alpha)$ – частота встречаемости буквы α в колонке A .

$f_B(\beta)$ – частота встречаемости буквы β в колонке B .

- Пары колонок с высоким значением взаимной информации с большой степенью вероятности образуют комплементарную пару (если высоки совместные частоты для пар букв AT, CG)
- Для восстановления вторичной структуры можно использовать алгоритм Нуссинофф, приписывая в качестве весов пар значение взаимной информации.

Грамматика

- Определения
 - Терминальным символом называется символ, который может получаться в строке (обозначается малыми буквами)
 - Нетерминальный символ – символ для обозначения промежуточной подстроки
 - Грамматика – набор правил генерации слов

- Пример:

$$W_2 \rightarrow aW_1, W_1 \rightarrow bW_2, W_1 \rightarrow \varepsilon;$$

Порождает слова вида "abababab"

Стохастические контекстно-свободные грамматики

- Контекстно-свободные грамматики имеют правила вида

$$W \rightarrow \beta$$

β – терминальные и/или нетерминальные *исключая* нулевую строку

- Правила преобразования могут быть снабжены вероятностями
- Обобщает скрытые Марковские модели. Позволяет описывать вторичные структуры РНК.
- Пример. Грамматика

$$S \rightarrow aW_1t \mid cW_1g \mid gW_1c \mid tW_1a;$$

$$W_1 \rightarrow aW_2t \mid cW_2g \mid gW_2c \mid tW_2a;$$

$$W_2 \rightarrow aW_3t \mid cW_3g \mid gW_3c \mid tW_3a;$$

$$W_3 \rightarrow gaaa \mid gcaa$$

Порождает шпильки с длиной спирали 3 и с последовательностью в петле $gaaa$ или $gcaa$

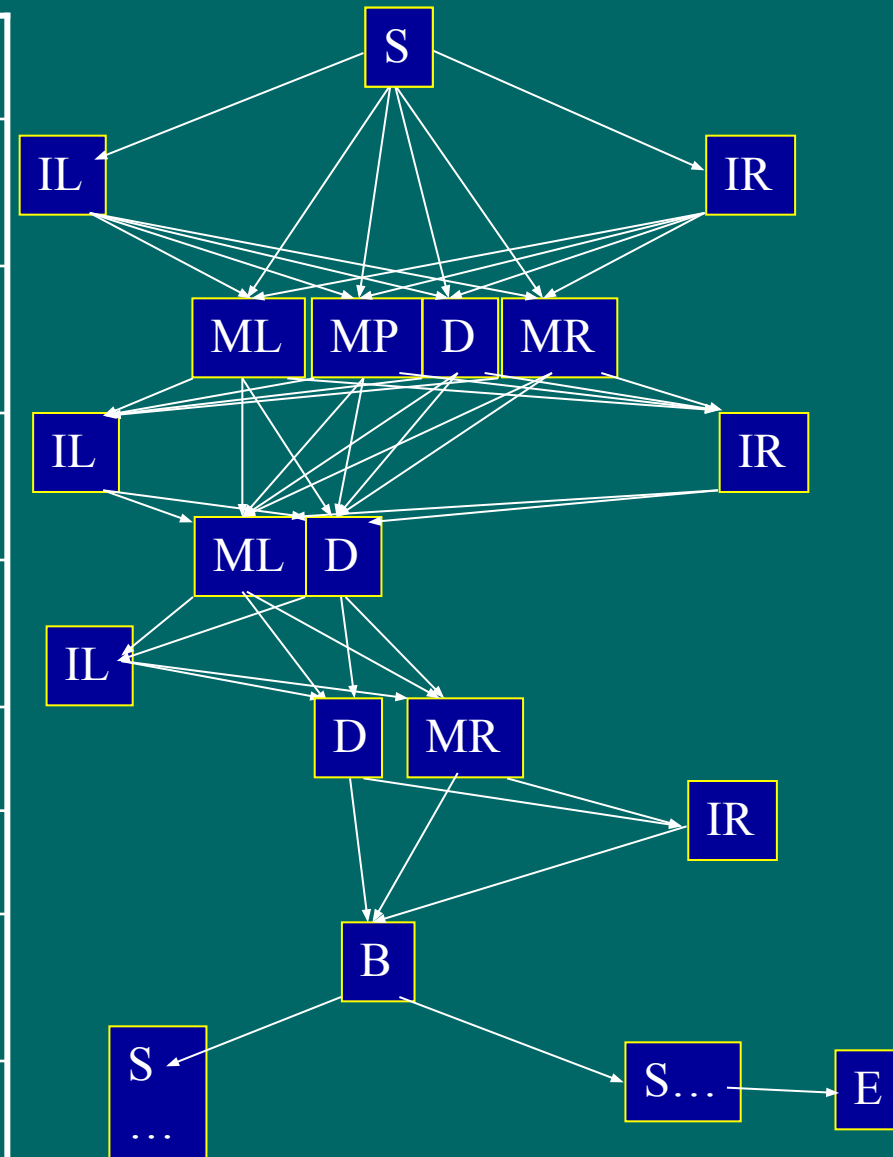
Задача выравнивания СКСТ с последовательностью

- СКСТ для описания вторичной структуры. Есть шесть типов преобразований:

Правило	Что делает	Эмиссия
$P \rightarrow aWb$	Порождение взаимно-комплементарной пары в спирали	16 вероятностей
$L \rightarrow aW$	Порождение символа слева от "центра"	4 вероятности
$R \rightarrow Wa$	Порождение символа справа от "центра"	4 вероятности
$B \rightarrow SS$	Бифуркация	1
$S \rightarrow W$	Старт спирали	1
$E \rightarrow \varepsilon$	Конец	1

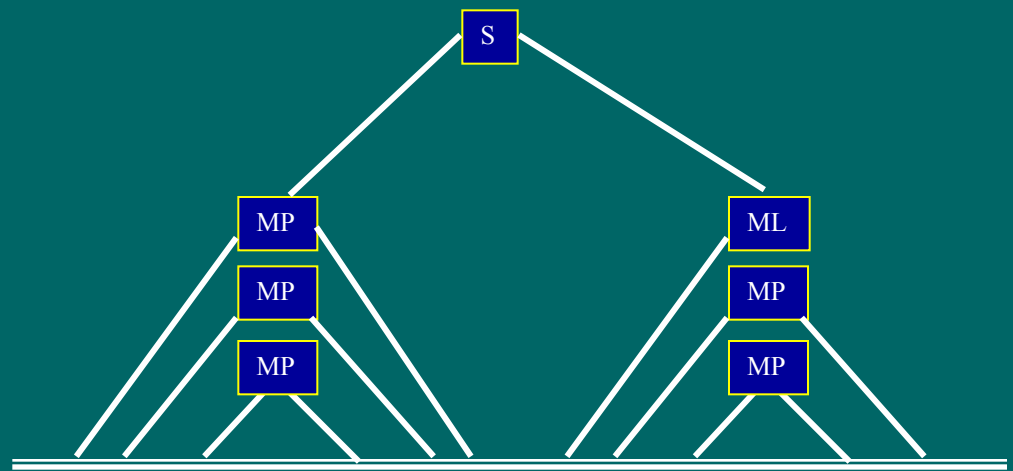
Общая модель для выравнивания вторичной структуры с последовательностью

S	Начало спирали
IL	Вставка символа в левое плечо спирали
IR	Вставка символа в правое плечо спирали
ML	Совпадение символа с символом в левом плече (делеция в правом)
MR	Совпадение символа с символом в правом плече (делеция в левом)
MP	Совпадение пары
D	Делеция пары
B	Бифуркация (порождение двух дочерних спиралей)
E	конец



Общая идея алгоритма разбора последовательности

- Заполняется трехмерная матрица $A(i,j,v)$: i,j – рассмотренный сегмент, v – тип вершины (MP ...)
- Просмотр начинается "изнутри" с коротких фрагментов. для каждого сегмента вероятность того, что этот сегмент может быть порожден соответствующей грамматикой. (Вариант динамического программирования)
- Затем просмотром "внутрь" находится способ вложения последовательности в грамматику



Поиск генов

