Конспект лекций по биоалгоритмам

Лектор: Миронов Андрей Александрович
Вёрстка: Курочкин Илья, Мараховская Александра, Нагаев Борис

2011

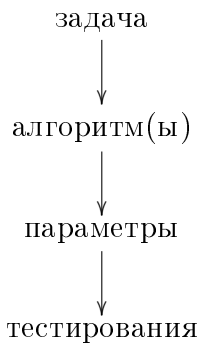
Содержание

1 Введение. Структура решения биологической задачи

1. биологическая задача
   (a) Есть последовательность — найти гены
   (b) Есть последовательности: они гомологичны или нет
2. Формализация задачи
   (a) Формализация задача должна быть математической.
   (b) Формализаций может быть много.
   (c) Пример: вводят редакционное расстояние.
3. Для каждой формализации разные алгоритмы.
   (a) подбор параметров
4. тестирование
   (a) область применимости

   Если тестирование не получается, меняют параметры или формализацию. Про алгоритм считают, что он всё же решает задачу, поэтому при провале тестирования алгоритм обычно не меняют.

1.1 Другой подход: кулинарный подход



То есть алгоритм решает ровно ту задачу, которая в нем написана.
Пример:

1. сравнения последовательностей — математические задачи
2. глобальное выравнивание NW — поиск оптимального пути в графе (формализация)
3. SW локальное выравнивание -тоже путь в графе
4. FASTA — сравнивает последовательности — пример «кулинарного алгоритма»

5. BLAST — формальной оптимизации нет — пример «кулинарного алгоритма»
6. CLUSTUL — множественное выравнивание. тоже решает ровно то, что в нем написано

Итак, мы видим два пути: с постановкой формальной задачи и без неё.

1.2 Формализация задачи сравнения последовательностей

Есть строки

$$S_1 = s_1^1 ... s_n^1$$
$$S_2 = s_1^2 ... s_n^2$$

Возможны следующие операции:

1. замена
2. вставка
3. удаление

Если есть набор операций прямого преобразования (S_1 в S_2), то набор операций для обратного преобразования получается однозначно. В тех случаях, где мы в прямом преобразовании делали замены, делаем обратные замены. Вместо делеций делаем соответствующие вставки, а вместо вставок — делеции. Поэтому расстояние от S_1 до S_2 равно расстоянию от S_2 до S_1: $D(S_1, S_2) = D(S_2, S_1)$ Что же такое расстояние в данном случае? Существует некое пространство M, для всех точек определена функция расстояния или метрика, которая задается следующим образом:

1. D: $M \bigotimes M \mapsto R$

Свойства расстояния:

1. $D \geq 0$
2. $D(a, a) = 0$
3. симметричность: $D(a, b) = D(b, a)$
4. неравенство треугольника: $D(a, b) \leq D(a, c) + D(c, b)$

Редакционное расстояние – минимальное количество элементарных преобразований, позволяющих получить из S_1 S_2. Пространство же состоит из всевозможных строчек различных длин, состоящих из букв определенного алфавита (в нашем случае либо ДНКовый, либо белковый алфавит). Редакционное расстояние обладает всеми свойствами функции расстояния.

Запомним пару заклинаний для белковых последовательностей:

1. Если редакционное расстояние маленькое, последовательности гомологичны.
2. Если что-то консервативно, находится под давлением отбора, значит, функционально важно.

Итак, мы преобразуем строку S_1 в S_2, при этом мы можем построить граф преобразований, где ребрами будут конкретные операции, вес которых определен ниже, а вершинами – начальные части выравнивания, где последние сопоставленные символы образуют координаты вершины:

```
||   ||s11||...||s1i||...||
||s21||   ||   ||   ||   ||
||...||   ||   ||   ||   ||
||s2j||   || ->|| x ||   ||
||...||   ||   ||   ||   ||
```

Вес каждой вершины расчитывается следующим образом:

1. если в нее попали из ячейки сверху или слева, что означает делецию в одной из последовательностей, то к весу ячейки из которой пришли добавляем 1;

2. если пришли по диагонали, то это означает либо сопостановление совпадающих символов, либо нет, что описывается дельта функцией.

$$\delta = \begin{cases} 0 & \text{если буквы равны} \\ 1 & \text{если буквы неравны} \end{cases}$$

Редакционное расстояние — вес минимального пути на этом графе.

1. Формализация задачи: минимизация количества операций.

2. Решение задачи: динамическое программирование на графах

3. Параметров нет.

4. Тестирование показало, что работает не очень хорошо. Есть экспериментально обоснованная гомология. Если выравнивание совпадает с выравниванием пространственных структур, то хорошо. Выяснилось, что это решение хорошо работает, если $identity < 90\%$, блестяще, когда 100%

Задачи проверки последовательностей на гомологичность и выравнивание — разные задачи.

Сколько всего бывает выравниваний последовательностей длин m и n: $2\sqrt{\pi/n}2^{2n}$ или более точно $(2n)!/(n!)^2$

Как это посчитать? Выравнивание можно представить в виде выборочной последовательности:

$$s_1^1, s_2^1, s_1^2, s_2^2, s_1^2, s_1^2 ...$$

Выборочная последовательность $\Leftrightarrow$ выравнивание Выборочную последовательность получают, читая выравнивание слева направо, по столбикам. Однако следующие выравнивания дадут одинаковую выборочную последовательность:

```
-a-          -a--
-b-    и     --b-
```

То есть, выборочных последовательностей немного(?) меньше. Теперь, выборочной последовательности сопоставим последовательность из 0 и 1. Пишем 0, когда берем из S_1 и 1, когда берем из S_2. Число всевозможных последовательнльностей, где общая длина n+n число нулей n, а единиц n.

$$C_n^{2n} = \frac{(2n)!}{(n!)^2}$$

дальше по формуле Стирлинга. Итак, просматриваемое пространство очень большое $\sim 2^n$

1.3 Динамическое программирование для нахождения редакционного расстояния

Есть очередная вершина графа, ячейка таблицы. В ней есть число, хранящее редакционное расстояние.

1. $$d_{i,j} = \min \begin{cases} d_{i-1,j} + 1, \\ d_{i,j-1} + 1, \\ d_{i-1,j-1} + \delta(s_i^1, s_j^1) \end{cases}$$

2. $$\delta(a,b) = \begin{cases} 0 & \text{если буквы равны} \\ 1 & \text{если буквы неравны} \end{cases}$$

3. *indel* — вставка-или-делеция

Теперь обсудим дно рекурсии / базу индукции. Вводим краевые вершины слева от таблицы и сверху над таблицей, с которых всё начинается. Аналогично завершение осуществляется через краевые вершины справа и снизу. как определить d в краевой вершине завершения:

$$d_{m+1,i} = \min \begin{cases} d_{m,i}, \\ d_{m+1,i-1} + 1 \end{cases}$$

движение по краевым полосам соответствует инделям в начале или в конце.

Замена: $w_{i,j} = -d_{i,j}$. На ребрах будем ставить не 1 и $-\Delta$, а -1 и Δ. $w_{i,j}$ — max вес сходства.

Ещё изменения:

1. диагональное ребро: -q — вес на несовпадение, p — вес за совпадение

2. вертикальное и горизонтальное: $-\Delta$ — штраф за геп (не путать с δ)

Такое решение позволяет решать задачу уже при $identity = 80\%$

Если на горизонтальных и вертикальных ребрах в краевой зоне написать 0, то мы не штрафуем за индели в краевой зоне. Это выравнивание с висячими концами. Бывают разные варианты висячих концов. Когда мы натягиваем фрагмент на геном, мы не штрафуем за висячие концы генома, но штрафуем за висячие концы фрагмента.

Время работы алгоритма: $T = O(mn)$ Количество памяти: $M = O(mn)$ — раньше это было слишком много.

1.3.1 Алгоритм Мюллера-Майерса

Количество памяти можно сократить. Он не очень актуален при современной технике, но красив и поучителен. Гоним алгоритм в обратном направлении, пользуясь симметричностью замен, которые нужно произвести. Берем линию, рассекающую таблицу пополам: $n/2$ Для всех точек этой линии за один проход находим значения оптимальных путей от начала и от конца. Можем найти ту точку, через которую проходит оптимальное выравнивание. Знаем одну точку на оптимальном выравнивании и его вес.

```
|| ... || ... || ... ||
|| тут ||     ||     ||
||     ||  x  ||     ||
||     ||     || тут ||
|| ... || ... || ... ||
```

Потом запускаем ту же задачу для двух более маленьких частей. Нам не нужно помнить само динамическое программирование, только его результат. Расход памяти: $M = O(m + n)$ Из-за этого динамическое программирование для одной точки приходится проводить много раз. Пусть m = n

$$T \sim n^2 + \frac{n^2}{2} + \frac{n^2}{4} + ... = 2n^2$$

То есть, всего в два раза больше операций, а экономия памяти значительна.

Ещё одна оптимизация: вряд ли путь будет проходить близко к границам таблицы, скорее пойдет в основном посередине. Проводим полоску по диагонали и рассматриваем только лежащие в ней точки. Мы заставляем выравнивание лежать в этой полосе. Это экономит и время, и память.

При должном упорстве все описанные алгоритмы хорошо работают при $identity > 70\%$ Вводим локально-оптимальное выравнивание — такое выравнивание фрагментов последовательностей, что любое удлиннение или укорочение его приводит к уменшению его веса.

1.4 SW

Локальное выравнивание — выравнивание с максимальным весом, его нельзя улучшить. За всё, что происходит за границами выравнивания, даем цену 0. Из стартовой вершины проводим ребра нулевого веса в каждую из точек. Из каждой точки также ребра нулевого веса в конечную точку.

Примечание: ввели новое понятие S — матрица замен. Используется вместо δ.

$$w_{i,j} = \max \begin{cases} w_{i-1,j} - d, \\ w_{i,j-1} - d, \\ S(s_i, s_j), \\ 0 \end{cases}$$

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

Вычисляем вес от вхождения до конца, а потом находим максимум — там и делаем конец выравнивания.

В большей степени зависит от параметров, чем глобальное выравнивание. Хорошо работает при $identity > 60\%$

Хорошее выравнивание часто разбивается на блоки:

```
WWW---WWWW---
```

Плохое выравнивание:

```
W-W-WW--WW-W-W
```

Хочется, чтобы выравнивание получалось блочным.

Переформализуем задачу. Вводим штраф за делецию, нелинейно зависящий от её длины. Чем больше делеция, тем меньше должен уменьшаться штраф за увеличение её на одну букву. То есть, график зависимости штрафа за делецию от длины делеции должен быть выпуклым (как логарифм или корень). Чем больше одиночных делеций, тем меньше вклад каждой из них. Если есть длинная делеция, алгоритм не должен пытаться разбить её на несколько.

Реализация: вводим в нашем графе вертикальные и горизонтальные переходы на несколько ячеек. Такой переход будет соответствовать делеции блока.

$$w_{i,j} = \max \begin{cases} \max\limits_k w_{i-k,j} - delta_{del}(i-k), \\ \max\limits_k w_{i,j-k} - delta_{del}(j-k), \\ S(s_i, s_j), \\ 0 \end{cases}$$

1. $T = O(n^3)$
2. $delta_{del}$ — афинный штраф за делецию:

$$delta_{del}(l) = d_0 + d_{ext} l$$

Можно от O(n^3) перейти к O(n^2): Вводим два пути: v-путь — путь делеций, w-путь — путь замен, а также ребра «рождения» и закрытия делеций. На ребрах v-пути стоит d_{ext}. Всего будет в каждую вершину входить ограниченное число ребер (5). закрытие делеции (переход $v \to w$) — бесплатно рекурсия:

$$v_{i,j} = \max \begin{cases} w_{i-1,j-1} - d_0, \\ v_{i-1,j} - d_{ext}, \\ v_{i,j-1} - d_{ext}, \end{cases}$$

$$w_{i,j} = \max \begin{cases} v_{i,j} + S(s_i, s_j), \\ S(s_i, s_j), \\ 0 \end{cases}$$

Часто выравнивание блочное, но в нем есть плохие участки. Хочется выделять невыравниваемые участки. Чтобы это реализовать, к предыдущей схеме добавляем переход $v \to v$ с ценой d_{un}(невыравниваниваемое). Афинные штрафы идут ещё и за эти переходы. В каждую ячейку 6 переходов

$$v_{i,j} = \max \begin{cases} w_{i-1,j-1} - d_0, \\ v_{i-1,j} - d_{ext}, \\ v_{i,j-1} - d_{ext}, \\ v_{i-1,j-1} - d_{un} \end{cases}$$

Параметры: match, mism, del. Пусть match = 1 (так как конкретный размер нам неважен) mism > 2 del (иначе любое несовпадение можно будет обойти двумя делециями).

Случайная аминокислотная последовательность. При оценке E-value в BLAST используется бернулиевская модель случайной последовательности, а белки в реальности подчиняются не этой модели.

Считаем, что есть две последовательности бернулиевские, причем все буквы равновероятны. Получили вес выравнивания 18. Надо оценить вероятность получить на случайных последовательностях вес 18 или больше.

Можно всю теорию перенести на марковскую модель.

1.5 Наибольшая общая подпоследовательность

Нет штрафов, только награды за совпадения. В данном случае локальное выравнивание не отличается от глобального. Находим под-последовательность (набор позиций), в которой обе последовательности совпадают.

Теорема. Мат.ожидание длины наибольшей общей подпоследовательности пропорционально длине последовательности. Доказательство. Обе последовательности длины n. Возьмем половину первой последовательности, половину второй последовательности, построим выравнивание половин. Получили вес выравнивания правой части и левой части. Вес общего выравнивания больше или равен суммы весов выравниваний половин.

$$r(n) \geq r1 + r2$$

1. $E(r(n)) \geq E(r1) + E(r2)$, так как E — аддитивная функция
2. $E(r) \geq const * n$

Но $E(r) \leq 2n$, так как длина общей подпоследовательности не может быть больше сумме длин исходных последовательностей. Значит, E(r) $\sim$ n

1.6 Наибольшее общее слово

1. $match = 1$
2. $mism = +inf$
3. $del = +inf$

Если наложить последовательности без сдвигов, то длина наибольшего общего слова будет равна числу последовательных успехов в серии испытаний Бернулли. При одном наложении

$$E(r) = log_{1/p} mn + const$$

const — известная константа, p — вероятность совпадения. Таким образом, $E(r) \sim log(n)$

Утв. На плоскости с осями mism, del ассимптотика E в некоторой области около 0 будет линейной, а во внешной области — логарифмическая. Параметры (матрица BLOSUM) подбираются так, чтобы получить логарифмическую ассимптотику, чтобы не было больших висячих хвостов

1.7 Замечания

1. Последовательнbость caagacatac — случайная последовательность? Нет, случайное число/последовательность/событие не произошло, данная последовательность — реализация случайной последовательности. Как только случайное число было получено, его вероятность стала 1. Вероятность выпавшего орла равна 1, так как он уже выпал.

2. Мы писали:

   $$... = w(i-1, j-1) + S(s(i), s(j))$$

   а до этого использовали $\delta(s(i), s(j))$. S — матрица сопоставления букв.

   Мы получили:

   (a) свободу сопоставления букв

   (b) возможность решать задачу локального выравнивания (так она решается проще)

2 Составление матрицы сопоставления

Общее соображение по составлению таких матриц: Транзиция (пурин на пурин или пиримидин на пиримидин) более вероятна, чем трансверсия.

Рассмотрим парное выравнивание без делеций (вернее, их мы сейчас не учитываем). Считаем позиции независимыми (Бернулиевская модель) x и y — последовательности. p(x и y — выравнивание) $= \prod p(x(i), y(i)$ | выравнивание) по всем i p(x,y | нет выравнивания) $= \prod p(x(i))p(y(i))$ — вероятность, что совпали случайно

Введем обозначения:

1. i, j, k — позиции

2. α и β — буквы, типы аминокислот

$$L = \log \frac{p(\alpha\beta|M)}{p(\alpha\beta|R)} = \log \frac{p(\alpha\beta|M)}{p(\alpha)p(\beta)} = S(\alpha\beta)$$

L — правдоподобие

Это и становится значением матрицы сопоставления (например, BLOSUM)

Используем программу локального выравнивания, использующую простую функцию δ Строим стопки множественных выравниваний, построенных из локальных выравниваний. Такие локальные выравнивания ложатся в БД BLOCKS. Длина одного такого блока небольшая (например, 6 аминокислот).

$f(\alpha\beta) = \frac{n(\alpha\beta}{N}$ | все белки, все колонки $\approx p(\alpha\beta|M)$

$$S_{bl}(\alpha\beta) = \log_2 \frac{p(\alpha\beta|M)}{p(\alpha)p(\beta)}$$

Пример:

```
A
G
G
S
```

значения n:

1. пара AG — 2 раза
2. пара AS — 1 раз
3. пара GG — 1 раз
4. пара GS — 2 раза

Из-за большого количества близких последовательностей возникает перегруженность базы. Выкинем последовательности, слишком похожие на кого-то ещё в базе (когда уровень сходства выше определенного порога). Для разных значений порого получаем разные блоки. Если задать порого 50 percent (последовательности довольно далекие). Мы получим матрицу BLOSUM50. Это число определяет уровень фильтрации последовательностей.

Самой правильной матрицей определили BLOSUM62. Использовались структурные выравнивания. BLOSUM62 дало наилучшее соответствие структурным выравниваниям.

1. Нехорошо делать порог слишком низким (25, очень далекие последовательности, мало колонок), статистика будет слишком малой. Чтобы увидеть все 210 независимых параметров, нужно много испытаний.

2. если белки близкие, надо использовать высокие BLOSUM (80-90). В далеких белках далекие замены (триптофан на глицин) более вероятны. Можно сначала строить выравнивание при помощи BLOSUM62, а потом использовать другую BLOSUM, в зависимости от identity

примерные значения identity:

1. от 30 percent — если есть структуры
2. от 50 percent — для установления ортологичности
3. от 70 percent — для функциональной идентичности

Процесс эволюции: $\alpha \to \beta$. Близкие последовательности: повторных событий мало, то есть вторичных замен не произошло. $p(\alpha\beta) = p(\Delta t)(\alpha|\beta)p(\beta)$ — по прошествии времени Δt Это марковское событие

$p(\alpha\beta) = p(2\Delta t)(\alpha|\beta)p(\beta)$ — по прошествии времени $2\Delta t$

$p(2\Delta t) = p(\Delta t)^2$(возвели матрицу условных вероятностей в квадрат) $p(N\Delta t) = p(\Delta t)^N$ (возвели матрицу условных вероятностей в степень N)

2.1 Матрица PAM

PAM1 — 1 одна замена на 100 аминокислот. PAM2 — PAM1^2. И так далее.

PAM250 — на 100 аминокислотах произошло 250 замен.

$$p(N\Delta t)(\alpha\beta|M) = p(N\Delta t)(\alpha|\beta)p(\beta)$$

$$PAM(N) = \log \frac{p(N\Delta t)(\alpha|\beta)}{P(\alpha)}$$

Получается серия матриц PAM.

выбор матрицы сказывается на качестве выравнивания при identity около 40 percent

2.2 Распределение экстремальных значений

ξ — случайная величина (СВ) $G(x) = p(\xi < x)$ — функция распределения (ФР) случайной величины ξ. Бросили кубик несколько раз. Посмотрели максимальное значение — это новая СВ. N случайных испытаний. Хотим найти $G(max(\xi_1, \xi_2, ..., \xi_n))$

$$\zeta = max(\xi_1, \xi_2, ..., \xi_n)$$

$$p(x \le \zeta) = 1 - p(x > \zeta) = 1 - p(x > \xi_1) \times ... \times p(x > \xi_n)$$

то есть x больше каждого из $\xi_1\xi_2...\xi_n$

$$\text{ФР максимума} = 1 - (1 - G(x))^N$$

$$\text{ФР минимума} = G(x)^N$$

Для нормального распределения: $G_{max}(exp(Nk))$ FIXME

2.3 Безделеционное локальное выравнивание

Теорема. Если есть две последовательности длин m и n, w — порог на вес локального безделеционного выравнивания. ξ — число ЛБВ с весом $\geq w$.

Последовательности считаем бернулиевскими. Найдем математическое ожидание. $E(\xi) = kmne^{-\lambda w}$ — это у нас e-value константы k и λ зависят от матрицы замен. Одна из последовательностей — весь банк. E-value зависит от размера банка линейно. P-value — вероятность увидеть событие такое или лучше при заданном количестве испытаний. $p_{\text{value}} = 1 - e^{-E(\xi)}$.

2.4 Как делать поиск по банку

Быстрый поиск по банку осуществляется с помощью хеширования. Подготовка банка:

1. строим индексную таблицу: AAA встретилось там-то и там-то
2. эти AAA — L-граммы — все комбинации букв заданной длины
3. имея такую индексную таблицу, можем понять, кто на кого в принципе может быть похож.
4. получаем набор якорей (точно совпадающие L-граммы)
5. ...

2.5 Способы обработки индексной таблицы

2.5.1 FASTA

FAST A. входной формат: фаста-формат

Строим матрицу сидов. Сид имеет две координаты. Наносим сиды на таблицу. Ищем диагонали, на которых много сидов. Сиды:

$$S(i_1, j_1)S(i_2, j_2)$$

Сиды лежат на (одной) диагонали:

$$d = i_1 - j_1 = i_2 - j_2$$

отбираем несколько (параметр программы) диагоналей, на которых лежат сиды (не менее 2). В выделенной полосе вокруг диагонали запускаем SW около найденных сидов — это эффективно. Как только score SW доходит до 0, на этом месте останавливаемся.

Программа FASTA работает медленнее BLAST, но аккуратнее

Как оценить статистически:

```
a1 w1
a2 w2
...
ak wk
```

$z - score = \frac{w-E}{\sigma}$ считают, что z-score распределен нормально (не всегда верно) исходя из этого рассчитывают p-value

2.5.2 BLAST

```
ITKAFLQV
ISRAFLEV
   сид
```

Начинает с сидо. Идем от сида вправо, накапливая веса, какие получатся, без вставок и делеций. Аналогично идем налево. Получаем участок высокого сходства. Выбираем максимум слева и справа, рассматриваем весь фрагмент между ними. Считает E-value для этого фрагмента. Отбирает по лучшим e-value.

Так работал BLAST1. Плохо ловил неблизкие последовательности.

```
BANK:  AFL
QUERY: GFI
```

2.5.3 BLAST2

Поэтому, чтобы такие сиды поднимались, берут не только саму L-грамму, но и множество всех L-грамм, которые отличаются L-граммы из QUERY не более, чем на определенный порог. Однако из-за этого количество сидов снова стало большим, что плохо. Поэтому используется подход FASTA: рассматривают только те диагонали, на которых есть много затравок. Делаем небольшого SW на несколько букв в сторону.

BLAST идеологически ищет короткие выравнивания, FASTA — длинные.

2.5.4 Ещё один подход

Наносим сиды на карту. Соединяем недалекие сиды ребрами, если они совместимы. После этого можно запускать динамическое программирование. Аккуратнее, чем все остальные, но помедленнее.

2.6 Немного математики

1. δ-функция — это не функция
   (a) $\delta(x) = 0 \forall x \neq 0$
   (b) $\int\limits_{-\infty}^{+\infty} \delta(x)dx = 1$
   (c) $\int\limits_{-\infty}^{+\infty} \delta(x-y)f(x)dx = f(y)$ (это её свойство)
2. $\delta(i,j) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$ — символ Кронекера, родственная к δ-функции
3. $\Gamma(x) = \int\limits_{0}^{+\infty} e^{-t}t^{x-1}dt$
   (a) $\Gamma(0) = \Gamma(1) = 0$
   (b) $\Gamma(x+1) = x\Gamma(x)$ — сравни с $(n+1)! = (n+1)n!$
   (c) $\Gamma(n+1) = n!$ — Γ является обобщением факториала на нецелые числа

2.6.1 Вероятностные модели последовательностей

Для многих задач важно уметь сказать для последовательности вероятность того, что она появилась.

Генерация последовательностей:

1. когда выборы букв для позиций не зависят друг от друга. $p = (\Pi p(ai))p(n), p(n)$ — вероятность длины. Минимальная длина белка, синтезированного рибосомой — 7 (сигнальный белок Ecoli). Средняя длина белка: 300 для бактерий, 600 для человека. Максимум: 4000. Распределение длин белков описывается с помощью p(n)
$$\text{число параметров} = \text{размер алфавита} - 1 = |\Sigma| - 1$$
2. в геноме слова CG сильно недопредставлены (вероятность меньше бернулиевской). Есть избегания некоторых комбинаций. Поэтому для описания последовательностей часто используют «марковскую модель первого порядка». Есть матрица переходных вероятностей. Вероятность буквы зависит от предыдущей буквы. Видится одна предыдущая буква.
   (a) $P(a_1)$
   (b) $P(a_2) = P(a_2|a_1)P(a_1), P(.|.)$ — матрица
   (c) $P(a_3) = P(a_3|a_2)P(a_2) = P(a_3|a_2)P(a_2|a_1)P(a_1)$
   (d) число параметров $= |\Sigma|(|\Sigma| - 1$
3. марковская модель более высокого порядка. Например, 3
   (a) $P(a_1)$
   (b) $P(a_2|a_1)$
   (c) $P(a_3|a_1, a_2)$
   (d) число параметров $|\Sigma|^3 - |\Sigma|^2$

Бернулиевская модель является частным случаем марковской первого порядка. Марковская первого порядка является частным случаем марковской второго порядка и т.д.

В марковской модели слишком большого порядка может теряться влияние далеких букв на выбор.

Порядок марковской модели ограничивается размером обучающей выборки.

$Lk =$сумма по всем словам в выборке $n(a_1, ..., a_{i+k}) \log P(...)$

$BIC = -L_k + A^k(A-1)\log N, A = |\Sigma|$

как только BIC достигает максимума, это значит, что что оптимальный порядок достигнут.

2.6.2 Статистика

Тестировали лекарство. В таблице показано, сколько людей из общего количества выздровели.

	лечили	контроль
Мужчины	50/90	4/12
Женщины	10/12	80/120

Вывод: мужчин лечит ($5/9 > 1/3$), женщин лечит ($5/6 > 2/3$), а если в сумме людей калечит ($20/34 < 21/33$)

2.6.3 Теория вероятности

Сделали N испытаний, n успехов. Вероятность успеха: n/N

Однако: при малом числе испытаний получается такое: три раза выпал орел. Мы всегда делаем свою априорную оценку вероятности. Вероятность выигрыша оценивается априорно. После первого забега возникает поправка — апостериорная вероятность. Но при этом вероятностный характер (из чего исходили в самом начале) сохраняется.

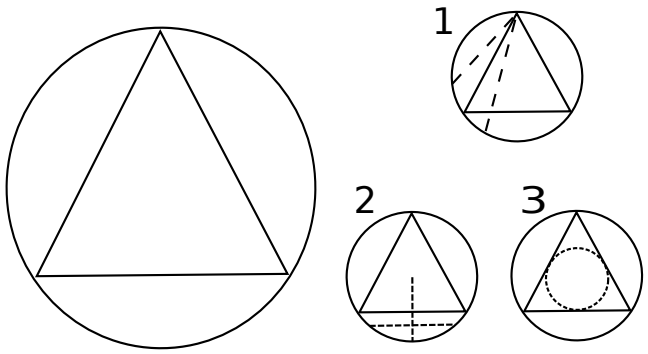
3 К слову о теории вероятности

Какая вероятность того, что длина случайной хорды больше стороны вписанного в окружность равностороннего треугольника?

1. Кладем один конец хорды в вершину треугольника. Вероятность того, что случайная хорда длиннее, — 1/3

2. Будем проводить хорду параллельно одной из сторон круга в той же половине круга, где лежит эта сторона. Если хорда попадет снаружи от этой стороны, она будет короче, иначе длиннее. Сторона круга рассекает перпендикулярный ей радиус пополам. Вероятность того, что случайная хорда длиннее, — 1/2

3. Впишем круг в этот равносторонний треугольник. Если хорда пересечет внутренний круг, она длиннее стороны треугольника. Отношение площадей внешнего и внутреннего круга: 4 Вероятность — 1/4



Ответ зависит от модели случайной хорды

3.1 Байесова статистика

Есть монета, про которую неивестно, правильная она или нет. На ней три раза выпал Орел. С какой вероятностью Орел выпадет в следующий раз?

Если мы заранее знаем, что $p = 1/2$, то можно было и не бросать.

Имеется мешок монет разной кривизны. $f = P(p)$ — распределение (плотность) вероятностей для вероятности выпадения Орла. Три раза Орел.

Формула Байеса:

$$P(x, y) = P(y|x)P(x)$$

$P(x, y)$ — совместная вероятность x и y, $P(x, y) = P(x|y)P(y)$, $P(x|y) = \frac{P(y|x)P(x)}{P(y)}$, $P(y)$ — полная вероятность.

Вероятность того, что снова выпадет Орел:

$$p(p|3 \text{ орла}) = \frac{P(p, 3 \text{ орла})}{P(3 \text{ орла})} = \frac{P(3 \text{ орла}|p)P(p)}{P(3 \text{ орла})} =$$

$$= \frac{p^3 P(p)}{P(3 \text{ орла})} = (*) = \frac{p^3 P(p)}{\int_0^1 p^3 P(p) dp}$$

Пояснение: $P(3) = z$ — вероятность трех орлов вообще,

$$f_{posterior}(p) = P(p|3 \text{ орла})$$

$$\int_0^1 f_{post}(p) dp = 1$$

$$f_{post}(p) = 1/z$$

$$1 = \frac{1}{z \int_0^1 p^3 P(p) dp}$$

$$z = P(3 \text{ орла}) = \int_0^1 p^3 P(p) dp$$

В некоторых случаях надо брать не ингеграл, а сумму

Получили апостериорную распределение вероятностей свойств данной монеты

$p = 1/2$ — априорное распределение вероятностей

Будем считать, что априорное распределение вероятностей равномерное (какая вероятность встретить динозавра за углом — $1/2$) В таком случае $P(p) = 1$

Для монеты: $P(p, 3 \text{ орла}) = \frac{p^3}{0.25} = 4p^3$ — плотность вероятности. Это плотность вероятности того, что вероятность орла в следующий раз будет p. К примеру, подставим $1/2$ в эту формулу — получим плотность вероятности.

1. MAP — оценка максимальной апостериорной вероятности В данном случае равна 1, это точка максимума плотности вероятности $4p^3$

2. ML — оценка максимального правдоподобия — max P(исхода|p) это оценка априорной вероятности. В данном случае тож равна 1. Если бы априорное распределение не было равномерным, эта величина осталась бы равной 1, а MAP бы изменилось.

3. E-оценка Оценим мат.ожидание.

$$E(p) = \int_0^1 p \times f(p) dp = \int p \times 4p^3 dp = 4/5$$

Если бы орел выпал не 3 раза, а n, получилось бы $(n + 1)/(n + 2) = \frac{n*}{N*}$

3.1.1 Задача 1

Есть шулер. У него есть прямая кость и кривая кость

Прямая кость: $p(6) = 1/6$

Кривая кость: $p(6) = 1/2$

Вероятность того, что шулер использует кривую кость: 0.01

С ним сыграли. 3 раза выпала 6. Какова вероятность того, что он взял кривую кость?

$$P(\text{кривая}|3\text{ раза:}6) = \frac{P(\text{кривая}, 3\text{ раза:}6)}{P(3\text{ раза:}6)} = 0.21$$

$$P(\text{кривая}, 3\text{ раза:}6) = P(3\text{ раза:}6|\text{кривая}) \times P(\text{кривая})$$

$$P(3\text{ раза:}6) = P(6|\text{прямая})^3 \times P(\text{прямая}) + P(6|\text{кривая})^3 \times P(\text{кривая}) = 0.06$$

Итак, $P(\text{кривая}|3\text{ раза:}6) = 0.99 \times 0.005 + 0.01$

Сколько раз он должен выбросить 6, чтобы вероятность того, что он жульничал, превысила 50 percent ?

3.1.2 Задача 2

Есть редкая болезнь, встречается 1 на 1 000 000 Есть тест. Если болен, то безошибочно (100 percent) определяет, что он болен. Если здоров, то ошибается (ошибочно считая его больным) с вероятностью 10^{-4} Есть ли смысл проходить такой тест?

$$P(\text{здоров}|_{\text{тест}}+) = \frac{P(+|\text{здоров}) \times P(\text{здоров})}{P(+|\text{здоров}) \times P(\text{здоров}) + P(+|_{\text{болен}}) \times P(_{\text{болен}})} \approx \frac{10^{-4}}{(10^{-4} + 10^{-6})} = 0.99$$

То есть, если тест оказался положительный, с 99 percent вероятностью человек здоров. А если тест оказался отрицательным, то и подавно толку от теста нет.

3.1.3 Задача 3

Есть два генома, довольно похожих. известен percent identity (обозначим id) обнаружили шпильку в первом геноме. На том же месте во втором геноме обнаружили шпильку. Буквы другие, но всё же шпилька (комплементарность).

Какова вероятность обнаружить шпильку в втором геноме при наличии шпильки в первом геноме? P(шпилька 2 | шпилька 1)

Обозначения:

```
=== a1 ==== a2 ==== геном 1

=== b1 ==== b2 ==== геном 2
```

$$P(\beta_1 \diamond \beta_2|\alpha_1 \diamond \alpha_2) = p(\beta = at|\alpha_1 \diamond \alpha_2) + ...(4\text{ штуки})$$

$$p(\beta = at|\alpha_1 \diamond \alpha_2) = ... = \frac{\text{id}^2 + 3(\frac{1-\text{id}}{3})^2}{16} = \frac{\text{id}^2 + \frac{1-\text{id}^2}{3}}{16}$$

(в этой формуле ошибка)

Исправим ошибку:

$$p(\beta = at|\alpha_1 \diamond \alpha_2) = \frac{p(\alpha_1 \diamond \alpha_2|\beta = at)P(\beta = at)}{p(\alpha_1 \diamond \alpha_2)}$$

$$p(\alpha_1 \diamond \alpha_2) = 0.25$$

$$p(\alpha_1 \diamond \alpha_2|\beta = at)P(\beta = at) =$$

$$= \frac{p(\alpha = at|\beta = at)p(\beta = at) + p(\alpha = gc|\beta = at)p(\beta = at) + ...(\text{еще 2 такие штуки})}{0.25} =$$

$$= \frac{\mathrm{id}^2 + \frac{(1-\mathrm{id})^2}{3}}{4}$$

$$P(\beta_1 \diamond \beta_2|\alpha_1 \diamond \alpha_2) = \mathrm{id}^2 + \frac{(1-\mathrm{id})^2}{3}$$

Проверка:

id	$P(\beta_1\beta_2\|a1 \diamond a2)$
0	1/3
1/4	1/4
1	1

Забавно, что при $id = 0$ вероятность найти шпильку больше, чем для $id = 1/4$

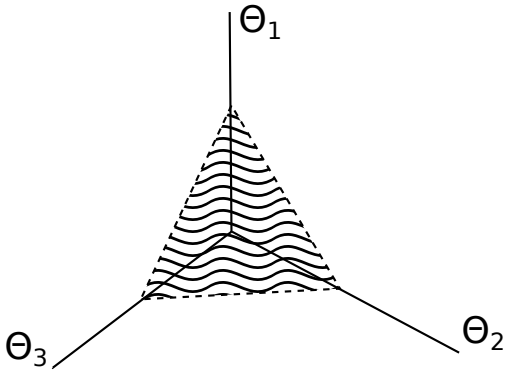
3.2 Априорные параметры

D — наблюдение. Θ — параметры, которые описывают априорные данные. Для монеты это 1 параметр, а для монеты — 5.

$P(\Theta|D)$ — апостериорная вероятность, $P(\Theta)$ — априорная вероятность,

$$P(\Theta|D) = \frac{P(D|\Theta)P(\Theta)}{\int P(D|\Theta)P(\Theta)d\Theta}$$

Трехмерное пространство параметров: $\Theta_1 + \Theta_2 + \Theta_3 = 1$ $\Theta_i \geq 0$ Это образует симплекс — многогранник с наименьшим числом граней в подпространстве на 1 меньшей размерности.

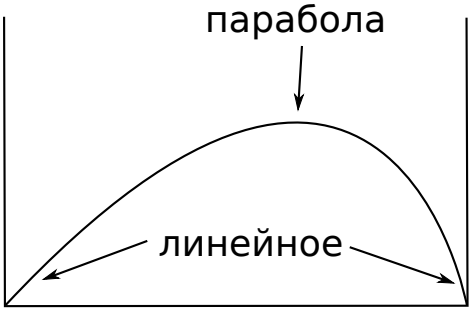


Параметр Θ для монеты:

1. по-хорошему должен быть δ-функцией (с пиком в 0.5)
2. размазано по (0,1): распределение Дирихле

$$Z\Theta^{\alpha_1}(1-\Theta)^{\alpha_2}$$

Z — нормировочный множитель



В n-мерном случае: $D(\Theta) = Z\prod\Theta_i^{\alpha_i}\delta(1-\sum\Theta_i)$

Сделали наблюдения, записали, сколько раз встретился какой исход. Как теперь оценить параметры Θ?

$$P(\Theta|D) = \frac{P(D|\Theta)P(\Theta)}{\int P(D|\Theta)P(\Theta)d\Theta}$$

$$P(\Theta|D) = \frac{\prod\Theta_i^{n_i}z_0\prod\Theta_i^{\alpha_i}}{Z...}$$

это выражение надо максимизировать, при условии, что $\sum D_i = 1$.

Метод неопределенных множителей Лагранжа:

$$\frac{\partial}{\partial\Theta_i}(\frac{z_0}{z}\cdot\prod\Theta_i^{n_i+\alpha_i} - \lambda(\sum\Theta_i - 1)) = 0$$

Система уравнений по всем i:

$$\frac{z_0}{z}\prod_{k!=i}\Theta_k^{n_k+\alpha_k}(n_i+\alpha_i)\Theta_i^{n_i+\alpha_i-1} - \lambda = 0$$

$$\frac{z_0}{z}\prod\frac{\Theta_k^{n_k+\alpha_k}}{\Theta_i^{n_i+\alpha_i}} - \lambda = 0$$

$$\Theta_i = \frac{A(n_i+\alpha_i)}{\lambda}$$

$$\sum\Theta_i = \sum\frac{A(n_i+\alpha_i)}{\lambda} = 1$$

$$\frac{A}{\lambda} = \frac{1}{\sum(n_i+\alpha_i)}$$

$\Theta_i = \frac{n_i+\alpha_i}{\sum(n_i+\alpha_i)}$ — MAP-оценка, максимум апостериорной вероятности

Почти как мы всегда считали, число успехов делить на число наблюдений, но добавляются «как бы-наблюдения» (псевдо-каунты), основанные на наших априорных предположениях.

E-оценка:

$$\Theta_i = \frac{n_i + \alpha_i + 1}{\sum(n_i + \alpha_i + 1)}$$

Оценка Θ — оценка природы вещей. Из наблюдений пытаемся оценить природу. Однако природа — великолепный шулер.

3.2.1 Задача

Выявить, подменил ли шулер монету — задача, сходная с задачей выявления кодирующей области. В обоих случаях происходит подмена статистики.

3.3 Скрытые марковские модели (HMM)

Мы видим серию исходов x_i. В каждый момент может с вероятностью $p(\text{п} \to \text{н})$ подменить правильную монету на неправильную. В каждый момент может с вероятностью $p(\text{н} \to \text{п})$ подменить неправильную монету на правильную.

У правильной и у неправильной монет есть вероятности выпадения орла и решки.

Общий случай:

1. есть алфавит A
2. есть набор состояний S_i
3. В каждом состоянии есть вероятности порождения символов из алфовита — эмиссионные вероятности $S_i : e^i_\alpha \alpha$ из A, $\sum e^i_\alpha = 1$
4. a_{ij} — переходные вероятности — вероятность перехода из i в j состояние. $\sum_{j=1} a_{ij}$

Похоже на конечный автомат, описывается конечным автоматом. На каждом шаге генерирует символ и переходит в другое состояние или остаемся в том же состоянии с некоторыми вероятностями. Также всегда есть возможность перехода в конечное состояние, ан котором мы прекращаем генерацию.

Бернулиевская модель генерации символов и модель с шуллером, у которого есть две монеты могут быть описаны с помощью HMM.

Примеры:

1. если есть два состояния, каждое из которых генерирует только один символ, мы всегда сможем предсказать, в каком состоянии находилась система в каждый момент времени.
2. Есть состояния a и b. Рассчитать вероятность предложенного пути, имея сгенерированную последовательность. Каждому пути соответствует вероятность. Ищем путь с максимальным весом. Свели задачу к поиску оптимального пути в графе. Поиск оптимального пути в графе для HMM — алгоритм Витерби. Декодинг (расшифровка): из набора наблюдений получаем набор состояний. Модель называется скрытой, так как мы не знаем, в каком состоянии была система, когда генерировала очередное слово.

3.3.1 Алгоритм Витерби

1. B — начало (begin)
2. E — end
3. L — длина последовательности

В каждой позиции i заводим витерби-переменные: $v^a(i)$

$$v^a(i) = \max_b v^b(i-1)a_{ba}e^a(x_i)$$

Начало рекурсии:

$$v^a(1) = \max_a a_{Ba}e^a(x_1)$$

$$v^E(L) = \max_z a_{zE}v^z(L-1)$$

3.3.2 Биологические приложения

Предсказание:

1. вторичной структуры белков
2. трансмембранных доменов TMHMM
3. сигнальных пептидов
4. CpG острова — C не метилируется, хотя обычно он метилируется в контексте с C

4 Задачи над HMM

Поиск сайтов узнавания

$$\text{Begin} \longrightarrow \text{Background}$$

1. e1
2. e2 — эмиссионные вероятности
3. e3

И так и порождаются, пока не начнется сайт. Сайт описывается профилем или весовой матрицей. Есть набор вероятностей для первой позиции сайта. Есть вероятности встретить букву для первой позиции. Вероятность перейти во вторую позицию сайта = 1. Точно так же есть вероятности для всех остальных позиций сайта. Когда сайт кончился, мы вернемся или в фоновую вероятность, или в другой сайт, или в конец.

Это простая HMM — внутри сайта все вероятности равны 1.

Самая простая HMM:

$$\text{Begin} \longrightarrow \text{End}$$

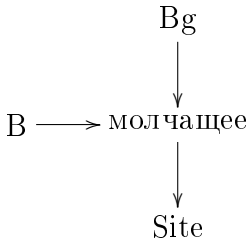
Чуть сложнее HMM — Бернулиевская.

4.1 Задача для Бернулиевской посл

Есть последовательность acggta Какова вероятность её в Бернуливской модели? $1/4^6$ Всего таких последовательностей 4^6. Сумма вероятностей равна 1. Но ещё есть последовательности других длин. Получается, что сумма вероятностей больше 1. Всё дело в том, что $1/4^6$ = вероятность при условии длины. $P(S) = P(S|L)P(L)$ Если просуммировать по всем последовательностям всех возможных длин, тогда полная сумма будет равна 1. Поэтому с вероятностями первая проверка всегда в том, будет ли сумма вероятностей равна 1.

4.2 Молчащие состояния

см тетрадь FIXME



4.3 Сайт узнавания бактериального sigma-фактора

см тетрадь

Два бокса (Gilbert и Pribnow), между которыми случайное соединение длины 15-21. Есть распределение расстояний. HMM для этого сайта узнавания (примерная):

```
Bg --- site --- Bg(спейсер) ---  site --- Bg(кодирующая)
```

Определяя вероятности остаться в Bg(спейсер) или выйти из него, мы можем добиться экспоненциального распределения длин спейсера. (доказать) Но мы хотим устроить какое-то распределение расстояний спейсера. Создадим много параллельных спейсеров (для каждой из возможных длин). С какой-то вероятностью мы входим в один из спейсеров. Внутри каждого спейсера последовательно определенное число Bg, между которыми вероятность перехода 1. Вероятность войти в каждый из спейсеров назначаем в соответствии с распределением длин спейсеров.

Это важный способ моделировать распределение длин. Может использоваться для моделирования кодирующих последовательностей. Самый длинных белок: около 4 000 остатков. Самый короткий белок: 7 остатков. Средняя длина одной закодированной полипептидной цепи: 300 у бактерий, 600 у человека. Размер гена у человека: 30 000 bp, средний размер экзона: 150, интрон: 200-300 (60 — 1.2M) Скорость транскрипции у бактерий: Скорость трансляции у бактерий: Общая длина ДНК человека: 1.5 м (гаплоидная)

4.4 HMM CPG островов

CpG-динуклеотид: сначала состояние, генерирующее букву C, потом с вероятностью 1 (в приближении) переходит в состояние, генерирующее G CpG-острова — не биологические объекты (экспериментально подтвержденные), а конструкты. На CpG

островах понижен уровень метилирования.

$$B \longrightarrow Bg \rightleftarrows \text{остров} \longrightarrow E$$

Части:

1. CpG
2. Bg

Считаем, что остров начинается и заканчивается на одной из CpG пар.
см тетрадь

4.5 HMM для открытой рамки считывания

см тетрадь

Однако это не совсем правильная схема, так как предполагается, что каждая позиция или фоновая или в ORF. Но ведь бывают пересечения рамок, причем рамки могут находиться на разных цепях.

Объекты:

1. старт-кодоны ATG
2. нормальные кодоны codons
3. стоп-кодоны stop

Нормальные кодоны:

1. AAA
2. AAC
3. ...
4. TAA — стоп
5. TTT

Каждый из кодонов, например AAA, — тоже HMM, внутри которого вероятности равны 1.

Stop:

1. TGA
2. TAA
3. TAG

Вероятности выйти из кодона: 1. Вероятности войти в конкретный кодон разные. Кодоны не равновероятны:

1. так как аминокислоты неравновероятны

2. синонимичные кодоны тоже неравновероятны — у организма есть предпочитаемые тРНК

Кстати, зачем есть предпочитаемые тРНК:

1. декодированием кода (сопоставлением кодонов к аминокслотам) занимается аминоацил-тРНК-синтетаза
2. аминоацил-тРНК-синтетаза может ошибаться.
3. Надо больше синтезировать те аминоацил-тРНК, при синтезе которым реже происходит ошибка.

Стоп:

1. стоп-кодоны имеют разную вероятность.
2. Один из стоп-кодонов может кодировать селено-цистеин.

4.6 Конечные автоматы

Любую HMM можно представить в виде конечного автомата. Генерируемые буквы надо ставить над стрелками перехода конечного автомата. FIXME

4.7 Биологическое применение HMM

Концепты HMM:

1. схема
2. численные значения параметров

Имея HMM, надо выдать вероятность, что, например, промотор действительно здест есть.

Есть наблюдение и набор состояний, которые могут здесь реализоваться. Натягиеваем модель на наблюдение — получаем большой граф. Вероятность пути по графу — произведение вероятности встретить букву. Ищем наиболее вероятный путь. Граф идет слева направо. Благодары этому он ациклический, никакой топологической сортировки делать не надо.

4.7.1 Витерби

1. $v_k(i)$ — наилучшая вероятность прийти из начала в состояние k в позиции i
2. $v_B(0) = 1$ (вероятность прийти в начало в нуле, если мы только начали) (инициация рекурсии)
3. $v_k(i) = \max\limits_l v_l(i-1)a_{lk}e_k(x_i)$ (продолжение рекурсии)
4. $P_{\text{пути}} = v_E(L{+}1) = \max\limits_l v_l(L)$ — полная вероятность пути (завершение рекурсии)

Вероятности путей будут очень маленькие

4.7.2 Forward-Backward

В данный момент мы не хотим найти промотор, а хотим выяснить, какова вероятность того, что промотор находится здесь. В таком случае вероятность должна быть близкой к 1.

К примеру, у нас есть Gilbert, а за ним два альтернативных Pribnow. Надо определить вероятность того, что мы находимся в Gilbert.

π_i — состояние в позиции i, x — последовательность

$$P(\pi_i = k|x) = \frac{P(\pi_i = k, x)}{P(x)}$$

P(x) в знаменателе нужен, чтобы сумма вероятностей была равна 1.

$$P(\pi_i = k, x) = P(x...x_i, \pi_i = k)P(x_{i+1}...x_E|\pi_i = k)$$

то есть, вероятность дойти до сюда и дойти отсюда до конца. То есть, даже дохлое состояние, если оно поддержано другими состояниями, может быть хорошим.

$f_k(i) = P(x_1...x_i, \pi_i = k)$ — сумма по всем путям дойти до k,

$$f_k(i+1) = \sum_l f_l(i)a_{lk}e_k(x_{i+1})$$

$$f_b(0) = 1$$

в конце: f_E — сумма по всем путям вероятностей этих путей, то есть, вероятность последовательности: $f_E(L) = P(x)$

$b(i)$ — сумма по всем путям направо $b_k(i) = \sum b_l(i+1)a_{kl}e_l(x_{i+1})$

1. f — forward — смотрим эмиссию предыдущей буквы
2. b — backward — смотрим эмиссию следующей буквы

$$b_E(L+1) = 1$$

$$b_B(0) = P(x)$$

Эти две рекурсии должны выдавать одно и то же значение. Что идти справа налево, что слева направо, должны получить сумму вероятностей всех путей, то есть вероятность последовательности.

$P(\pi_i = k|x) = \frac{f_k(i)b_k(i)}{f_E(L+1)}$

Этот алгоритм — апостериорное декодирование. До этого был алгоритм Витерби

4.8 Задача о монете

получим вероятности того, где монета кривая или нет. (Смотри предыдущую лекцию.)

5 Разрешение HMM

Схема HMM зафиксирована, надо найти параметры:

1. Есть последовательности с разметками. Разметка из экспериментальных данных. Найти параметры. Эти разметки — обучающая выборка (пока будем так считать). Пример: кодирующие области. Где кончается белок — понятно, на стоп-кодоне. Где начинается, не вполне ясно, так как ATG может встречаться и внутри ORF Экспериментальные данные, подтверждающие, что данный ATG — стартовый, есть:
   (a) N-концевое сиквенирование белков
   (b) измерение молекулярной массы белков
   (c) ...
2. Нет разметок, но есть представление о том, как устроена HMM. Более печальный случай. Есть последовательности, а разметки на них нет. Пример:
   (a) взяли транскрипционный фактор, связывающийся с ДНК, посадили его на колонку, прогнали через неё порезанную ДНК и нашли множество фрагментов ДНК, которые связываются с этим белком. Задача: определить сайт узнавания.
   (b) трансмембранные белки

5.1 Первый случай (разметка известна)

Есть вероятность последовательности при условии параметров

$$\prod_z P(x^z|\theta) \to \max_\theta$$

Разметка — это путь на нашем графе.

$$\prod_z P(x^z|\theta) = \prod_z \prod_i a_{kl} e_k(x_i) =$$

$$\prod_z \prod_{k,l,\alpha} a_{kl}^{A_{kl}} e_k^{E_k(\alpha)} \to \max_{a,e}$$

A и E — наблюденные количества эмиссий и переходов

То есть подбираем такие параметры схемы, что полученные последовательности генерировались с максимальной вероятностью

$$\sum_l a_{kl} = 1$$

$$\sum \alpha e_k(\alpha) = 1$$

см тетрадь

мы оптимизировали вероятность последовательности при условии параметров. Но это не совсем правильно. У нас есть веростность параметрво при условии наблюдений.

$$P(\theta|x) = P(x|\theta)P(\theta)/P(x)$$

см тетрадь

5.2 Второй случай

Для параметров нужна разметка, а для разметки — параметры.

5.2.1 Виттерби-обучение

1. берем случайные параметры, удовлетворяющие условиям.
2. находим разметки (например, с помощью Витерби)
3. По этой разметке делаем оценку параметров. Возвращаемся на 2, пока не сойдётся.

Иногда часть параметров известно заранее. Пример: поиск сайтов. Мы знаем, что сайты встречаются не очень часть, поэтому у нас есть представление о соотношении вероятности. Мы можем зафиксировать часть параметров и не пускать их в обучение. Обучать будем только матрицу.

5.2.2 Forward-Backward

С помощью Forward-Backward мы знаем вероятность $f_k(i)$ прийти в x_j и вероятность $b_l(i+1)$ уйти из x_{j+1} Таким образом, мы знаем вероятность перехода $k_i \to l_{i+1}$

$$P(k_i \to l_{i+1}) = f_k(i)a_{kl}e_l(x_{i+1})b_l(i+1)/P(x)$$

$P(\pi_i = k, \pi)$
см тетрадь 11
Этот вариант позволяет нам делать ошибки в разметке. Витерби — более жесткий.
Для эмиссионных:

$$P(x_i = \alpha, \pi_i = k|x, \theta) = f_k(i)e_k(x_i)b_k(i)/P(x)$$
$$E_k(\alpha) = if_k(i)e_k(x_i)b_k(i)/P(x)$$

Это более мягкий способ обучения, по Бауму-Велчу. Если выборка небольшая, разумнее работать по Бауму-Велчу

Мы находим локальный оптимум, а не глобальный. Поэтому надо стартовать случайным образом много раз и смотреть, какие параметры получаются. Кроме того, мы получаем вероятности и можем оценить правильность полученных параметров $P(\theta|x)$

5.3 Оценка качества обучения

Может применяться не только в данном случае, но и в случае с нейронной сетью.

Тестирующая выборка. Проверять на том же, на чем учились, нельзя. Выборку разделяют на две половинки: обучающая и тестирующая. Может применять и когда есть разметка, и когда нет. (когда нет — берем часть связавшихся фрагментов ДНК и по ним находим сайты, а потом ищем эти сайты в второй половине).

Есть True — истинные сайты и Pred — предсказанные сайты.

1. TP — True Positives
2. TN — True Negatives
3. FN — False Negatives
4. FP — False Positives

True — там где наврали (недопредсказали или перепредсказали), Positives — где мы предсказали

Далеко не всегда у нас есть хороший True. Например, нет соглашения, где считать границу альфа-спирали.

1. $Sp = \frac{TP}{TP+FP}$ — специфичность
2. $Sn = \frac{TP}{TP+FN}$ — чувствительность
3. $Q = \frac{\text{пересечение}}{\text{объединение}} = TPTP + FP + FN$ — качество

В методах, где есть порог:

1. большой порог: будем недопредсказывать
2. малый порог: будем перепредсказывать
3. идет борьба между чувствительностью и специфичностью

ROC-кривая — кривая, отражающая зависимость Sp и Sn. По ней можно найти точку баланса между Sp и Sn. Иногда можно согласиться на перепредсказание при условии, что все True будут предсказаны, или на недопредсказание, если точно не будет предсказан «мусор».

Между кодонами есть зависимости

1. в гидрофобном ядре концентрируются гидрофобные остатки
2. рибосома может связываться не только с самим кодоном, но и с предыдущим

Значит, число параметров растет. Не всегда хорошо делать подбираемых параметров слишком много. Тогда точность определения параметров будет плохой, так как на каждый из случаев будет приходиться очень мало наблюдений. Возникает плохая возможность переобучить. Тогда на тестирующей выборке будет проваливаться. Например, иногда лучше предсказывать по одиночным нуклеотидам, а не по динуклеотидам

6 Вероятности букв в колонке

Предположим, есть колонка выравнивания:

```
L
V
V
L
L
```

Хочется рассчитать вероятности встретить букву в этой позиции. Проще всего вероятность встретить букву положить частоте её встречаемости:

$$e_k(\alpha) = \frac{E_k(\alpha)}{\sum\limits_{\alpha} E_k(\alpha)}$$

Теперь добавим псевдокаунты:

$$e_k(\alpha) = \frac{E_k(\alpha) + \psi}{\sum\limits_{\alpha}(E_k(\alpha) + \psi)}$$

6.1 Правила добавления псевдокаунтов

1. Правило Лапласа: $\psi(\alpha) = 1$
2. $\psi(\alpha) \sim q(\alpha)$, где $q(\alpha)$ — объективная частота буквы

   Однако так мы не учитываем априорные вероятности
3. $\psi(\alpha) \sim \sum\limits_{\beta \in \text{наблюдаемая колонка}} P(\ \beta \xrightarrow{substitution} \alpha\ )$

В последних двух случаях присутствует некий коэффициент пропорциональности:

$$\psi(\alpha) = A \sum ...$$

Чему должен быть равен этот коэффициент? На самом деле, в выборе этого коэффициента воля исследователя (то есть он подборочный).

Однако при выборе значения коэффициента пропорциональности есть такой подход:

$$A = \kappa\sqrt{N}$$

где κ — подборочная константа, N — число наблюдений.

Объяснить, почему так делать правильно, непросто. Интуитивное объяснение заключается в том, что вариабельность (то есть аналог стандартного отклонения) пропорциональна $\sqrt{N}$. То есть, если представлять наблюдение как истинное значение плюс ошибка:

$$\begin{array}{ccc} E_k(\alpha) & + & \psi \\ \wr & & \wr \\ N & & \sqrt{N} \end{array}$$

6.2 Взвешенные последовательности

Теперь представим, что это эти последовательности взяты из разных организмов:

```
HUM    V
Shimp  V
D.m.   L
D.p.   L
C.e.   L
Ecoli  Q
```

Тут понятно, что Q из Ecoli иметь больший вес, чем другие последовательности. Так как HUM и Shimp — близкие организмы, то тот факт, что у них одна и та же буква, менее ценен (информативен), чем тот факт, что у Ecoli Q. Мы изучаем белок и нам важно, что тут может быть (для этого мы и смотрим частоты букв), поэтому нам очень важно, что тут может быть и Q тоже.

Давайте придадим веса нашим наблюдениям.

$$E(\alpha) = \sum 1 \to E(\alpha) = \sum w_k(\alpha)$$

где w_k — вес последовательности в выравнивании.

Последовательности получают веса, так как выборка не является равномерной.

6.3 Как взвешивать последовательности

1. Многогранники Вороного

   Ω — метрическое пространство последовательностей

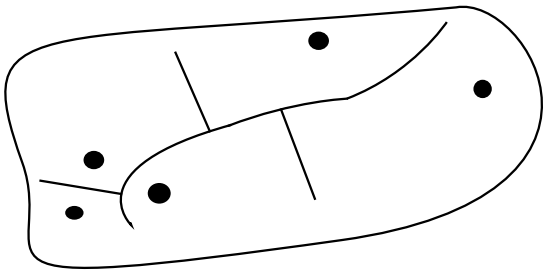
   (a) $\Omega \bigotimes \Omega \mapsto R^+$

   (b) $d(xy) > 0 \iff x \neq y$

   (c) $d(xy) = 0 \iff x = y$

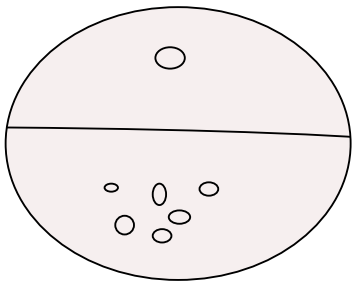
   (d) $d(xy) + d(yz) \geq d(xz)$

   Ω разбивается заборами на поместья. Забор ставится посередине между точками.



   Вес последовательности назначаем пропорцианальным весу (площади) поместья.

d — расстояние μ — мера (площадь поместья)



Отшельник будет иметь большой вес.

Практически считать можно так [метод Монте-Карло]: генерируем случайную последовательность в соответствии с моделью, смотрим, к какой из последовательностей она ближе и добавляем очко этой последовательности. Делаем там много раз. Если у нас 7 последовательностей, должно хватить 1000 бросков. Хорошо тут то, что у нас нет координат в пространстве, есть только расстояния. Можно было бы пытаться втащить последовательности в евклидово пространство, но лучше этим не заниматься.

2. Метод Г-С-Ч

   Есть взвешенное укорененное дерево.

```
 |---- 8 ----------------- D
 |
-|       |-- 5 ----------- C
 |-- 3 --|
         |       |-- 2 --- B
         |-- 3 --|
                 |-- 2 --- A
```

   Веса концевых веток полностью отходят своим послеледовательностям, веса остальных веток делятся поровну между всеми листьями, к которым они идут.

A	2	$+\frac{3}{2}$	$+1$	$=4.5$
B	2	$+\frac{3}{2}$	$+1$	$=4.5$
C	5		$+1$	$=6$
D	8			$=8$

   Алгоритм хорошо работает для ультраметрических деревьев, для перекошенных работает хуже.

6.4 Проблема множественного тестирования

Если был получен интересный результат, может встать вопрос, сколько неудачных попыток его получить было до этого. Если долго раз пытаться получить интересный результат, он рано или поздно может получится, но будет уже не так интересен.

Надо внести поправки на множественное тестирование.

1. Поправка Бонферони

$$p_{value} = Np$$

p — вероятность, N — число тестирований

Это жесткая поправка

2. FDR

ξ — случайная величина, $F(x)$ — функция распределения ξ, $f(x)$ — плотность ξ

Как понять, сколько попало в выборку из фонового распределения?

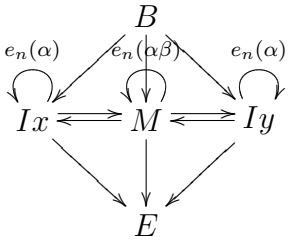
$$G(x) = \frac{n(\xi \geq x)}{N}$$
$$FDR = \frac{F(x)}{G(x)} = \frac{N}{n(\xi \geq x)} F(x)$$

7 HMM и выравнивание

7.1 Порождение пары последовательностей и выравниваний

Порождаем иногда символы, иногда пару символов

```
    m m m y y m x m
X:  a c c - - b b a
Y:  b b a c b d - a
    |           |
пара символов  1 символ
```



Соотношение $e_n(\alpha\beta)$ к I_α и I_β отвечает за длинны сопоставленных участков и делеций

M — match, пара букв

Ix и Iy — порождение буквы только в x или в y

Это HMM для построения(генерации) выравнивания.

7.2 Параметры

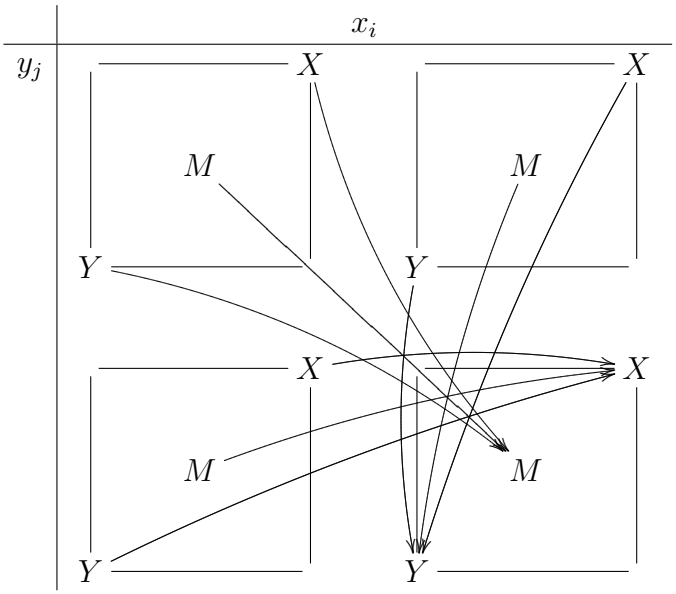
$M \in e(\alpha\beta)$ – вероятность пары букв

Ix и Iy : $e(\alpha)$ – есть переходные вероятности и вероятность уйти в конец и прийти из начала

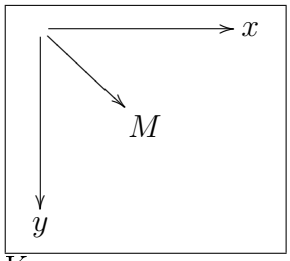
7.3 Задача декодирования

Теперь есть выравнивание. Надо решить обратную задачу: подсчитать все вероятности

С помощью алгоритма Витерби: Витерби для выравнивания



Каждая ячейка несет 3 состояния:



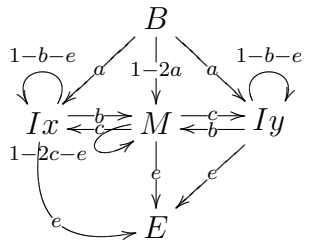
Какие есть переходы:

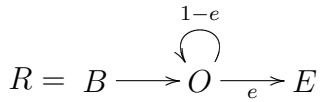
1. $M \to M$ по диагонали породили по букве в каждой из последовательностей, поэтому увеличили i и j
2. В M идут диагональные ребра
3. В x и y идут горизонтальные и вертикальные ребра соответственно.

$$\text{P(Путь)} = \prod a_{kl} \begin{cases} e_x(\alpha) \\ e_y(\beta) \\ e_M(\alpha\beta) \end{cases}$$

Считаем пока, что последовательности равноправные. Т.е. вероятности симметричные.

Пример выравнивания, где последовательности неравноправные: кДНК и геном.

$M =$ 



Мы пока использовали симметричные вероятности. Согласно Витерби считает max вероятность прийти в клеточку ij. Всего 3 Витерби-переменные(по числу состояний)

$$V_m(ij) = e(x_i y_j) \max \begin{cases} V_m(i-1, j-1)(1-2c-e) \\ V_x(i-1, j-1)b \\ V_y(i-1, j-1)b \end{cases}$$

$$V_x(ij) = e(x_i) \max \begin{cases} V_m(i-1, j)c \\ V_x(i-1, j)(1-b-e) \end{cases}$$

$$V_m(ij) = e(y_j) \max \ldots$$

$V_m(0,0) = 1 - 2a$
$V_x(0,0) = a$
$V_y(0,0) = a$

Находим оптимальный путь. Похоже на NW, однако там было сложение, а тут умножение. Это исправит логарифм. Переходим в логарифмическое простанство:

$$\log V_m(ij) = \ loge(x_i y_j) \max \begin{cases} \log V_m(i-1, j-1) + \log(1-2c-e) \\ \log V_x(i-1, j-1) + \log b \\ \log V_y(i-1, j-1) + \log b \end{cases}$$

Однако ln от вероятностей (< 1) отрицательный. Матрица в NW: $S_{\alpha\beta} = \log \frac{P(\alpha,\beta)}{a_\alpha a_\beta}$

7.3.1 Правдоподобие пути

Будем искать не P(путь), а правдоподобие пути.

$$L = \frac{P(x, y|M)}{P(x, y|R)} \to \max$$

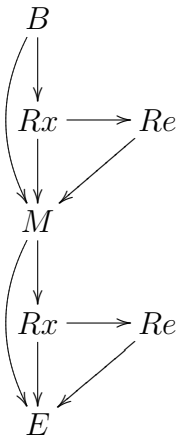
M — модель, R — случайные (бернулиевское независимое порождение x и y)

$$L_m = \frac{e(x_i y_j)}{e(x_i)e(y_j)} \max \begin{cases} L_m(i-1, j-1)\frac{1-2c-e}{1-e} \\ L_x(i-1, j-1)\frac{b}{1-e} \\ L_y(i-1, j-1)\frac{b}{1-e} \end{cases}$$

$e(x_i), e(y_j)(1-e)$ пришли от $P(x, y|R)$ После взятия log получили почти задачу максимизации веса в NW

7.4 Конструкторы

Из M и R строим схему:



Выравнивание:

```
xxx   xxxxx xxx
   xxxxxxxx
```

Пусть есть переход $R \to R$ $\log \frac{R \to R}{R \to R} = 0$ — Бернули
Этот 0 соответствует 0 из NW:

$$L_{ij} = \max \begin{cases} \dots \\ \dots \\ \dots \\ 0 \end{cases}$$

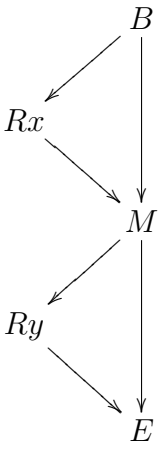
Такие блоки удобно реализовывать в программе. Этот подход дает нам участки выравнивания, которые нужно выравнивать, и участки, которые не нужно выравнивать.

Многодоменный белок:

```
~~~~      |||| ~~~     |||| ~~~~
    ~~~~~ ||||     ~~~ ||||      ~~~~~
  N-конец  кор  петля   кор  C-конец
```

Мы уже говорили в разговоре про выравнивание про необходимость возможности вносить невыравниваемы участки.

Как это реализовывать в HMM



7.5 Субоптимальное выравнивания

И Витерби и NW дают единственное оптимальное решение. Хочется иметь вероятности того, что данные два остатка сопоставлены. Причины:

1. 

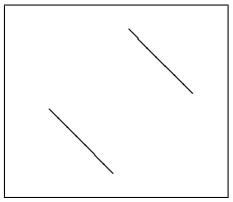
   остаток хочется переставить на другую сторону гена

2. двудоменные белки

   ```
   111111122222222
   222222211111111
   ```

   Тут 2 оптимальных выравнивания, но мы увидим только одно.

   Карта сходства:



Субоптимальное выравнивание — выравнивание с хорошим, но не самым лучшим весом. Итак, для каждых i и j мы хотим знать вероятность того, что мы находимся в состояниях M, Ix, Iy/ В случае монеты для этого используется Forward-Backward $P(\Pi_i = a) = \frac{f_a(a)b_a(i)}{P(x)}$ Применим FB к задаче выравнивания.

$$f_m(ij) = e(x_i y_j)(f_m(i-1, j-1)(1-2c-e) + f_x(i-1, j-1)b + f_y(i-1, j-1)b)$$

$$f_x(ij) = e(x_i)(f(i-1, j)(1-b-e) + f_m(i-1, j)c)$$
$$f_y(ij) = e(x_i)(f(i, j-1)(1-b-e) + f_m(i, j-1)c)$$

База:
$f_m(0,0) = 1 - 2a$
$f_x(0,0) = f_y(0,0) = a$
Конец:
$P(x, y|M) = f_e = f_m(e_1e_2)e + f_x(e_1e_2)e + f_y(e_1e_2)e$

Для backward аналогично:
$\diamond$ — сопоставление
$P(x_i \diamond y_j) = \frac{f_m(ij)b_m(ij)}{P(xy)}$
Пример:
Выравнивание:

```
xxx---xxxxx
xxxxxxxxxxx
==--------==
```

= высокие вероятности
− низкие вероятности
Положение гена не определено точно

Еще одно применение: имея $P(x_i \diamond y_j)$ подставляем их в качестве вероятностей сопоставления в алгоритм оптимального выравнивания. Это оптимальное апостериорное выравнивание. Заменяем $e(\alpha, \beta)$ на $P(x_i \diamond y_j)$

7.5.1 Построение субоптимального выравнивания

Хочется не просто слегка пошевелить ген, а порождать субоптимальное выравнивание, отличающееся от оптимального более координально. Строя оптимальное выравнивание, мы шли по обратным наилучшем переходам. Теперь же, идя назад, будем разыгрывать переход на ГСЧ, взвешивая на вероятность переходов.

$$P(m(ij) \to i-1, j-1) = \frac{f_m(i-1, j-1)b_m(ij)}{P(xy)}$$

$$P(m(ij) \to Ix(i-1, j-1)) = \frac{f_x(i-1, j-1)b_m(ij)}{P(xy)}$$

$$P(m(ij) \to Iy(i-1, j-1)) = \frac{f_y(i-1, j-1)b_m(ij)}{P(xy)}$$

Пропорционально этим вероятностям, выбираем путь и идем дальше (Монте-Карло)

7.6 Другие способы

Условно-оптимальное выравнивание – это оптимальное выравнивание, прошедшее через фиксированую точку. Это тоже стотостический метод. Есть и другие способы постороения субоптимального выравнивания.

8 Статфизика

P — состояний
n_i — число заполнений, т.е. число микросостояний попавших в это макросостояние
$\sum_i n_i = N$ Мы говорим о статистике Больцмана, т.е. частицы различимы (пронумерованы)

```
  микро      макро
|-------|
| 1 3 4 |      3
|-------|

|-------|
| 2   6 |      2
|-------|

|-------|
| 7   5 |      2
|-------|
```

Энтропия макросостояния $S = \log$ количества микросостояний, образующих данное макросостояние.

Кол-во микросостояний: $\frac{N!}{n_1!n_2!\ldots n_p!}$ т.е число способов распределить столько частиц по стольком ящикам.

$$n! \approx n^n e^{-n}$$

Кол-во микросостояний: $\frac{N^N e^{-N}}{n_1^{n_1} e^{-n_1} n_2^{n_2} e^{-n_2} \ldots n_p^{n_p} e^{-n_p}}$

$$log_{\text{кол-во}} = N\log N - n_1\log n_1 - n_2\log n_2 - \ldots - n_p\log n_p =$$
$$= n_1(\log N - \log n_1) - n_2(\log N - \log n_2) - \ldots - n_p(\log N - \log n_p) =$$
$$= -n_1\log\frac{n_1}{N} - -n_2\log\frac{n_2}{N} - \ldots =$$
$$= -N(f_1\log f_1 + f_2\log f_2 + \ldots) =$$
$$= -N\sum_i f_i\log f_i = \text{энтропия}$$

8.1 Информация

информация — потеря неопределенности, энтропии. Теперь у нас есть источник букв.

$$H_{\text{ист}} = -\sum_{\alpha\in A} p(\alpha)\log_2 p(\alpha)$$

Когда полностью прочитали, энтропия исчезает.
I — потеря неопределенности.

Но мы могли прочитать не полностью:
accNcat, N — неопределенность, поэтому часть энтропии осталось.

Пусть нуклеотиды равновероятны. Тогда каждый нуклеотид стоит 2 бита информации. N стоит 1 бит.

8.1.1 bit-score в BLAST

bit-score $= -\sum_{\text{pos}} p(S_i S_j)$, где S_i и S_j — сопоставлены

8.1.2 Если распределение частиц по ящикам не равновероятно:

полиномиальное распределение:

$$P_{\text{разм}} = \frac{N}{n_1! n_2! \dots n_p!} p_1^{n_1} \dots p_k^{n_k}$$

$$I(f|p) = \log P_{\text{разм}} = -N \sum f_i \log f_i + \sum n_i \log p_i =$$

$$= -N \sum f_i \log f_i + N \sum f_i \log p_i =$$

$$= -N \sum f_i \log \frac{f_i}{p_i}$$

Взаимная этропия двух распределений f и p.

8.1.3 Использование взаимной энтропии в биоинформатике

```
-A- -|
-A- -|
        Выделение групп
-G- -|
-G-  |
-G- -|
```

Смотрим взаимную энтропию первого множества относительно общего или второго. Взаимную энтропию называют расстоянием Кульбака, но оно не является расстоянием так как $d(ab) \neq d(ba)$. Доказано, что $I(f|p) \geq 0$, причем равна 0, только когда распределения полностью совпадают.

8.1.4 Теорема

$$I(f|p) = N \sum f_\alpha \log \frac{f_\alpha}{p_\alpha} \geq 0$$

$$I(f|p) = 0 \iff f_i = p_i$$

$$\sum f_i = 1$$

Воспользуемся методом неопределеных множителей Лагранжа

$$\begin{cases} \frac{\partial (I(f|p)) - \lambda(\sum f_i - 1)}{\partial f_i} \\ \dots \\ \sum f_i - 1 = 0 \end{cases}$$

получаем $n + 1$ уравнений

$$\frac{\partial}{\partial f_i} (\sum f_i \log f_i - f_i \log p_i - \lambda(\sum f_i - 1)) =$$

$$= \log f_i + 1 - \log p_i - \lambda = 0$$

$$\log \frac{f_i}{p_i} = \lambda - 1$$

$$f_i = e^{\lambda-1} p_i$$

Чтобы соблюсти $\sum f_i - 1 = 0$, надо $e^{\lambda-1} = 1 \implies$ экстремум достигается только при равенстве двух распределений

Пошевелили f_i:

$$f_i = p_i + \varepsilon_i$$

$$\sum \varepsilon_i = 0$$

$$\sum (p_i + \varepsilon_i) \log(1 + \frac{\varepsilon_i}{p_i}) = \sum (p_i + \varepsilon_i) \frac{\varepsilon_i}{p_i} = \sum \varepsilon_i + \sum \frac{\varepsilon_i^2}{p_i} \geq 0$$

9 Профили

```
  A F G T A
  A W G S S
  A F G T G
-----------
  A F G T X --- консенсус
```

1. Консенсусная последовательность определяется голосование.

   В большинстве описаний позиции независимы. Причина: наблюдений слишком мало, чтобы учесть даже взаимодействия соседей. Однако взаимодействия могут учитываться. Пример: сайты сплайсинга (алфавит малый, последовательностей много). Обычно позиции раасматриваются как независимые, т.к. не хватает данных.

   Чем определяется важность колонки: чем сильнее распределение отличается от глобального, тем больше информационное содержание колонки.

2. паттерн

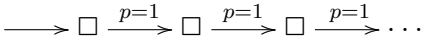
   ```
   A[FW]G[TS][ASG]
   ```

3. матрица частот

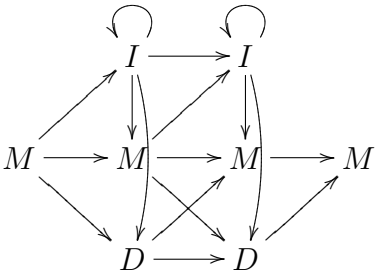
   ```
   A 3       1
   F   2
   W   1
   G     3   1
   T       2
   S       1 1
   ```

   обычно берут логарифмы частот, чтобы их складывать, а не умножать

Потенциально-весовая матрица — это HMM.



Однако множественное выравнивание (семейства, например) содержит много гепов (вставок-делеций). Для этого используется хитрый HMM:

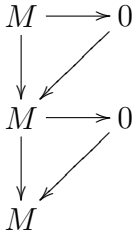


В HMM не каждое состояние должно обладать эмиссией. Состояние D (делеция) ничего не пораждает. Нужно, чтобы пропустить букву. Переходные вероятности зависят от позиции.

В плотном блоке выравнивания, мала вероятность найти на пути вставки и делеции. Нам дали новую последовательность. Определяем к какому семейству принадлежит эта последовательность.

Применяя алгоритмы HMM, считаем отношение правдоподобия, чтобы узнать, к какому семейству относится и с каким правдоподобием.

Когда мы прогоняем последовательность по профилю, это похоже на матрицу парного выравнивания:



Отличия:

1. ассиметричность
2. позиционно-зависимые вероятности

Параметров много, поэтому их определение может требовать творческого подхода.

9.1 Смеси Дирихле

Колонка может принадлежать TM-домену, α-спирали, петле и т.д. То есть, буквы могли прийти из разных источников (вероятностных пространств). Определение источника по наблюдению.

x — источник
f_i — распределение

$$P(\alpha|x) = \sum P(\alpha|f_i)P(f_i|x)$$

10 Множественное выравнивание

Что такое множественное выравнивание?

Это способ записать последовательности друг под другом так, чтобы гомологичные остатки образовывали столбик (стояли друг под другом). Напоминание: гомологичность — общность происхождения, т.е. эволюционный параметр, но в обычной ситуации мы не знаем истории последовательности. Существуют, конечно, и исключения: вирус гриппа, история эволюции которого ведется с 1918; вирус СПИДа.

Альтернативное определение: правильное выравнивание — это выравнивание, полученное из сопоставления пространственных структур. Существует золотой стандарт, состоящий из нескольких баз данных, например BALIBASE — benchmark alignment base. По этой базе данных проверяют программы множественных выравниваний.

Задача: лучше всего воспроизводить выравнивание из BALIbase. Если бы был такой алгоритм, есть надежда, что он хорошо бы строил и другие выравнивания.

10.1 Формализация задачи

Надо уметь оценивать качество выравнивания.

```
(a)bc...aca
(a)ba...cca    => число
(c)bc...cba
(a)cc...aba
```

Для парного: $\sum S_{\alpha\beta} - \sum D$ (S — сопоставления; D — делеции).

Для множественного выравнивания: сумма парных сопоставлений.

$$\sum_{col} = S_{aa} \cdot 3 + S_{ac} \cdot 3$$

(смотри первый столбец выравнивания).

Способ учета делеций: вместо делеций используем символ «-».

$$\lg \frac{P^3_{ac}}{P^3_a P^3_c}$$

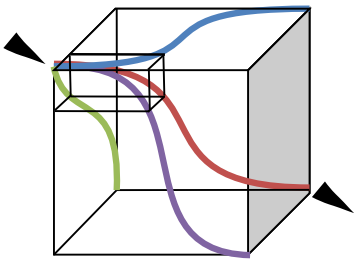
Правильнее было бы считать $\lg \frac{P_{3a1c}}{P^3_a P^3_c}$, то есть рассматривать все возможные колонки высоты h. Однако таким счетом не пользуются, т.к. слишком много было бы параметров (размер матрицы счетов за все колонки высоты h).

10.2 Сумма весов пар

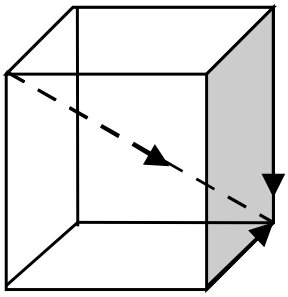
$$\sum_{i<j} \sum_k S(x^i_k x^j_k),$$

где i,j — номера последовательностей, k — номер колонки.

Для трех последовательностей:



красным обозначен какой-то путь, синим, зеленым и фиолетовым — его проекции на плоскости. Выделим элементарную ячейку (маленький кубик) и рассмотрим ее поподробнее.



В случае двух последовательностей (см. лекцию о парных выравниваниях) возможных переходов 3. В случае 3 — 7, 2 — 3, 3 — 7, 4 — 15, 5 — 31.

Если немного продолжить, то становится ясно, что количество переходов это 2^N, а количество ячеек LN, общее число операций $(2L)^N$. Возьмем среднестатистическое множественное выравнивание: 30 белков по 50 аминокислот:

$$(2 \cdot 50)^{30} = 100^{30} = 10^{60}\text{операций} = 10^{44}\text{лет}$$

Для сравнения время существования вселенной порядка 10^{10} лет. Так считать долго.

Допустим, что мы построили каким-то образом множественно выравнивание с заданным весом S_0, как мы его строили неважно, главное, что оно есть.

1. Спроецируем наш путь на двумерные грани (рис. 1);

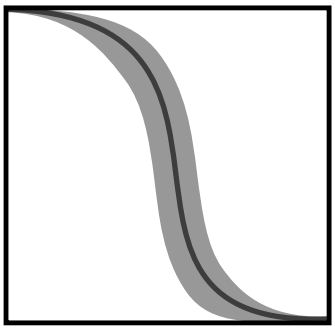
2. Вес множественного выравнивания(S0) равен сумме весов всех парных выравниваний (проекции).

Мы знаем, что есть оптимальное выравнивание, но не знаем, является ли наше с весом S_0 таковым, поэтому может потребовать только следующее условие:

$$S_{opt} \geq S_0$$

По аналогии вес оптимального выравнивания равен сумме весов оптимальных парных выравниваний. Из чего можно сделать вывод о том, что веса проекций правильного выравнивания не меньше весов проекций нашего выравнивания.

Возьмем одну из проекций и выделим в ней те клетки, если бы выравнивание проходило через них, то вес был бы не меньше, чем в нашем выравнивании.



Произведем это для всех проекций и посмотрим, где клетки из всех проекций имеют общую точку. В таких точках и будет проходить оптимальное множественное выравнивание. Но все осложняется тем, что если изначально мы взяли плохое выравнивание, то количество квадратов, которые необходимо обработать будет слишком велико, и в связи с этим программа будет работать долго.

Для поиска оптимальных путей в парном выравнивании (проекции) используется динамическое программирование. Итак, как нам выделить правильные клетки? Надо вычислять вес выравнивания и само выравнивание, проходящее через эту клетку. Вес оптимального выравнивания, проходящего через данную клетку: вес от начала до нашей клетки + вес от клетки до конца (как Forward-Backward).

$$w_{ij} = w^+_{ij} + w^-_{ij}$$

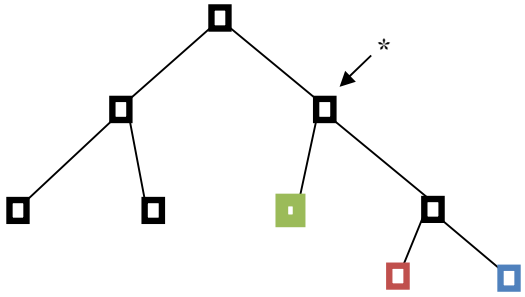
$$w^+_{ij} = max \begin{cases} w^+_{i-1,j-1} + S_{\alpha\beta}, \\ w^+_{i-1,j} - d, \\ w^+_{i,j-1} - d \end{cases}$$

$$w^-_{ij} = max \begin{cases} w^+_{i+1,j+1} + S_{\alpha\beta}, \\ w^-_{i+1,j} - d, \\ w^-_{i,j+1} - d \end{cases}$$

Таким образом, мы нашли те позиции i,j в которых $W_{ij} \geq S_0$

10.3 Идеология прогрессивного выравнивания

Предположим, что у нас есть бинарное дерево всех наших белков, которое можно построить по матрице расстояний, рассчитываемой по весам парных выравниваний. (Откуда берется дерево: по весам парных выравниваний можно построить мартицу расстояний, из которой можно получить дерево)



* – Выравнивание парного выравнивания с последовательностью.

Возьмем два самых близких белка (красный и синий), построим парное выравнивание, затем возьмем следующую последовательность, наиболее близкую к выровненным белкам и выровняем ее с нашим уже построенным парным выравниванием. И так далее будем добавлять по одной последовательности.

Как строить супервыравнивание (выравнивание двух выравниваний):

1. Сопоставить
2. Сделать разрыв (во всю колонку выравнивания)

Оптимизируется сумма:

$$\sum_{i<j} S^{ij}_{\alpha\beta} = \underbrace{\sum_{x<y<k} S^{xy}_{\alpha\beta}}_{(1)} + \underbrace{\sum_{x<k<y<n} S^{xy}_{\alpha\beta}}_{(2)} + \underbrace{\sum_{k<x<y<n} S^{xy}_{\alpha\beta}}_{(3)}$$

1. (1) и (3) — сумма внутренних пар
2. (2) — перекрестные пары

(3) и (1) мы рассчитали на этапе построения выравнивания. Обработка делеций вводит дополнительный алфавит: $\tilde{\text{A}} = \text{A} + \text{«-»}$, где гэп становится буквой. Аффинные штрафы за гэп не используются. Эта идея положена в основу программы ClustalW (кулинарный алгоритм).

К этой простой схеме есть следующее дополнение:

1. взвешивание последовательностей (в ClustalW алгоритм Сонхаммер)
2. контекст-зависимые веса за делецию: чем больше гэпов было в предыдущей колонке, тем дешевле будет вставка колонки гепов
3. Контекст-зависимые веса $S_{\alpha\beta}$

Недостатки:

1. Дерево может быть построено неверно
2. Дерево строится по парным, а не по множественным выравниваниям
3. Нет возможности пересмотра (уточечния)

   Пример, когда из-за отсутствия уточнения возникают ошибки:

```
x-
x-      x-
--      x- -- эти гепы пойдут до конца
     => --
xx      xx
xx      xx
```

10.4 Уточнение

1. Временно исключить часть последовательностей
2. Перестроить выравнивание
3. Возвращаем исключенные последовательности, подстраивая их под профиль

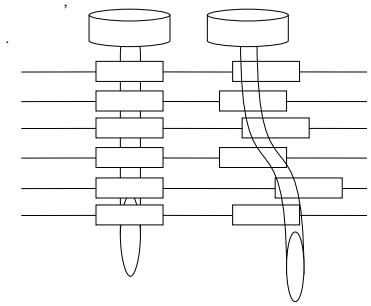
10.5 Алгоритмы

Алгоритмы для рассмотрения: T-coffe Muscle ClustalW Dialin ProbCones

Здесь перечислены основные программы построения множественных выравниваний. Нами будут рассмотрены три из них.

10.5.1 Dialin

Теперь перейдем к разбору алгоритма DIALIN. Хочется, чтобы выравнивание имело блочную структуру. В каждой последовательности находим консервативные блоки (например прогоняя ее против остальных). Закрепляем «гвоздями» консервативные блоки: Находят блоки слева и справа



Оставшееся подравнивают с помощью ClustalW. есть еще много улучшений алгоритма.

10.5.2 Probcons

Рассмотрим двеколонки выравнивания:

```
 1                    2
-V-                  XVY
-V-                  XLY
```

Про вторую можно сказать, что, хотя совпадение и хуже, она лучше, так как окружение совпадает.

Считаем апостериорные вероятности сопоставления:

$$P(x_i \diamond y_j)$$

Используем дерево как и в ClustalW (но строим без построения парных выравниваний). Вместо матрицы сопоставлений используем матрицы апостериорных вероятностей. Для каждой пары последовательностей: $FB \to D(x_i \diamond y_j)$ Еще одно ухищрение:

$$w(x_i \diamond y_j) = \sum_{z \in otherseq} (\sum_l imits_k p(x_i \diamond z_k) + \sum_l imits_k p(z_k \diamond y_j))$$

И так для всех x_i и y_j (контекстная буква из последовательности) получаем

$$w(x_i \diamond y_j)$$

При построении выравнивания двух выравниваний берем $w(x_i \diamond y_j)$.

Основной принцип: контекстно-зависимый score. Делеции не штрафуются. Алгоритм не вставляет делеции в ненужных месах, так как это сбило бы хорошие сопоставления.

В ситуации, приведенной ниже, ClustalW будет добавлять последовательности, неся за собой везде гэп. Probcons же подобной ошибки не допустит.

```
xxxxxxxxx-----------------xxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

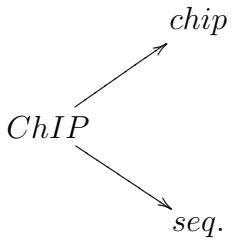
11 Распознавание сайтов связывания

Данные приходят с микрочипов. Однако микрочиповые данные содержат много(двухкратное) пере- или недо- предсказания.

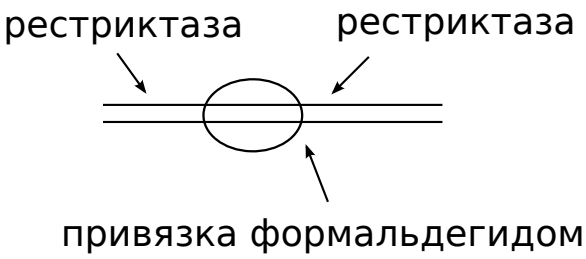
11.1 Способы поиска сайтов связывания

1. chip – белок не нужен
2. задержка в геле – белок нужен
3. SELEX –
   (a) белок закрепляется в колонке.
   (b) Затем случайная ДНК с известными концами

   ```
   XXXXXX--------XXXXXXX
   ```

   прогоняется через колонку.
   (c) Делаем ПЦР прилипших ДНК с ошибками.
   (d) Потом снова проганяем, причем условия отмывки каждый раз жестче.
   (e) Повторяем 5-6 раз.
4. 

   белок нужен.



Белок связан с антителом.

5. upstream перед генами-ортолагами: видимо регулируется одним и тем же белком

   Филогенетическая функция принтинга хорошо работает на бактериях

6. метаболический

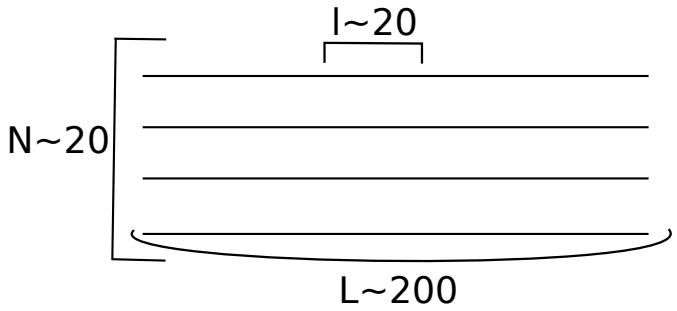
   $$M = \; o \longrightarrow o \longrightarrow o$$
   $$\qquad\qquad\quad\; \uparrow$$
   $$\qquad\qquad\quad\; o$$

   где M – метаболический путь

   Гены одного метаболического пути могут находиться под общим регулятором. Делаем аналогично предыдущему способу.

11.2 Постановка задачи:

Есть последовательности, про которые известно, что они содержат сайт.



Сайтов может быть несколько, а может не быть вообще. Надо увидеть устройство сайта (правило, с помощью которого можно было бы искать сайт в других последовательностях).

11.3 Модель сайтов

1. PWM + порог, позиционно-весовая матрица белок взвешивается с каждым нуклеотидом независимо

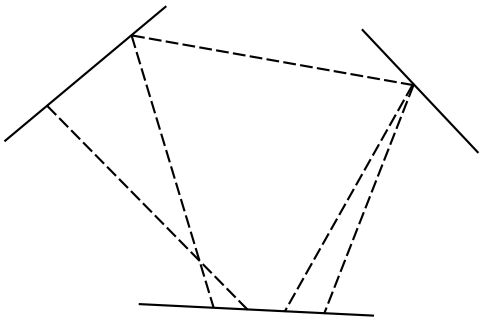
2. геометрия ДНК, это есть PWM на динуклеотидах + порог имеет больше параметров, ее труднее обучать

3. Magic Word + максимальное число замен, то есть есть идеальный сайт, а все наблюдаемые – отличие от него

4. логическое правило: логическая формула, связывающего нуклеотиды на разных позициях

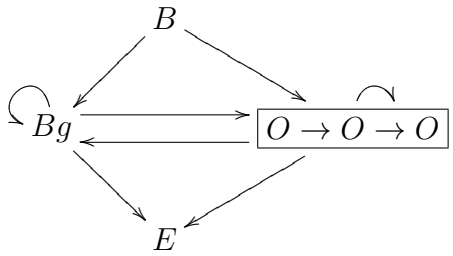
5. Непонятно что (кулинарный подход): использует нейронные сети. Не формулируется ни в один из типов $1-4$, ближе всего к 4

11.4 Граф



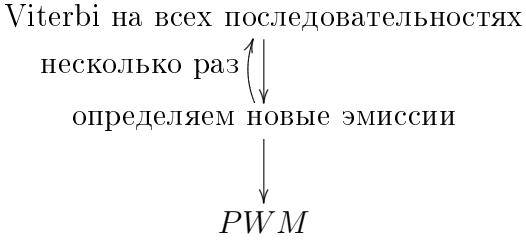
Итак если у нас есть 3 последовательности. Ребра проводим, если похожи. Имеем 3-дольный граф. Если нашли в этом графе клику, значит нашли множество похожих слов.

11.5 HMM



Обучение HMM $\to$ PWM
Как проводить обучение HMM:

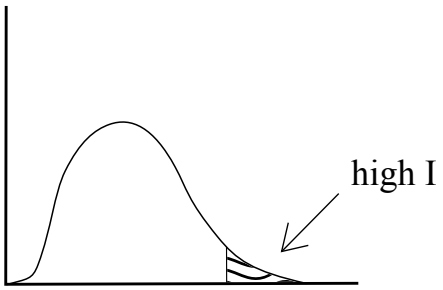
1. Viterbi-обучение. Задаем веса эмиссии по какому-то слову (берем PWM вида $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ и добавляем псевдокаунты)



2. или обучение по Baum-Welch

И делаем так для всех слов. Получим столько PWM, сколько букв в этих последовательностях.

Оценим PWM, для этого оценим информационное содержание PWM. Смотрим распределение I:



Высоким значениям I соответствуют интересные PWM.

Baum-Welch: делаем FB, при подсчете верочтностей считаем не число попаданий, а оцениваем апостериорные вероятности наблюдений.

Алгоритм Baum-Welch + I = MEME – алгоритм максимизации ожиданий.

11.6 MEME Алгоритм максимизации ожидания

Есть последовательности. В них есть сигнал – непрерывный участок (слово). Буквы в сигнале могут быть с вариациями.

Верим, что есть сигнал, описываемый PWM. Значит, в последовательностях есть слово, имеющее высокий вес относительно PWM (пока неизвестной нам).

Как искать PWM:

1. берем первое слово из первой последовательности
2. предполагаем, что оно и есть сигнал
3. строим PWM, находящую эту последовательность.

Классическая PWM:

$$PWM(S) = \begin{pmatrix} 1 & 0 & 0 & \\ 0 & 0 & 1 & \cdots \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{pmatrix}$$

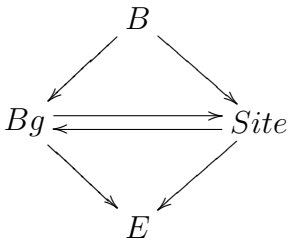
Боле правильный способ построения PWM (с псевдокаунтами):

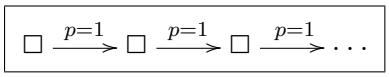
$$w_\alpha^j = \frac{C_\alpha^j + \psi_\alpha^j}{\sum C_\alpha^j + \psi_\alpha^j}$$

$$\Rightarrow \begin{matrix} 1.75 \\ 0.25 \\ 0.25 \\ 0.25 \end{matrix}$$

Псевдокаунты нужны, чтобы учесть неуверенность.
Алгоритм:

1. $i = 0, j = 0$ — первое слово в первой последовательности
2. i-ое слово в j-ой последовательности $\Rightarrow PWM(S_i^j)$
3. Опишем сигнал как HMM:



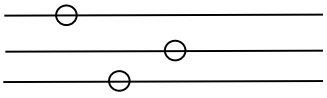
где $Site =$ 

Находим наилучшее вхождение во всех последовательностях (Витерби). Получаем обучающую выборку

4. Из обучающей выборки находим новую PWM Затем переходим к шагу (3) и так несколько раз. Уточнение реализации: есть порог для счета наилучшего вхождения. Этот порог может меняться от итерации к итерации.
5. $i{+}{+} \Rightarrow$ следующее слово, перейти к (2)
6. $j{+}{+} \Rightarrow$ следующая последовательность, перейти к (1)

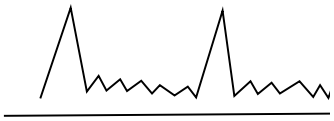
11.7 Второй подход к поиску сайтов

Сайты = координаты сайтов.
Выберем координаты случайно:



Затем подправляем положение сайтов. Позиция -1 означает, что в этой последовательности нет сайта (если это допускается).
Алгоритм:

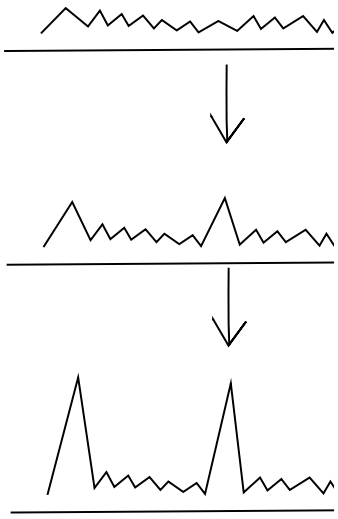
1. Выбираем произвольные позиции в последовательностях
2. вынимает ожну последовательнbость, по остальным стоим PWM
3. Затем на вынутой последовательности предсказываем сайты исходя из распределения вероятностей



Новый сайт выбираем сдучайно, с вероятностями из этого распределения.

4. Выбираем другую последовательность и делаем с ней такой же розыгрыш. И так много раз.

Как улучшается распределение:



Как строится рапределение:

$$P(pos) = Z^{-1}\frac{(p(pos|site)}{p(pos|bg)}$$

$Z = \sum \frac{(p(pos|site)}{p(pos|bg)}$ — нормировочная сумма, bg — не $site$

$$p(pos|site) = \prod_{i=pos}^{pos+l} \underbrace{\sum_{\alpha} f_{\alpha}^{i}\delta(S_i, \alpha)}_{\text{только соответствующие буквы}}$$

$$p(pos|bg) = \prod_{i=pos}^{pos+l} f_{\alpha}^{bg}\delta(S_i, \alpha)$$

(f^{bg} — фоновое распределение букв; если нуклеотиды распределены равномерно, это 0.25)

Алгоритм хорошо работает на прокариотах, плохо на эукариотах.

Подводные камни:

1. Заранее объявляется длина сайта (l в наших формулах). Можно прогадать, что будет вносить шум (сайт оказался меньше, чем предполагали) или отсекать часть информации (сайт оказался больше).

2. В алгоритме присутствует случайность. Это может приводить в сдвигу сайта в предсказании.

3. Сайт может быть повтором или палидромом (особенно в эукариотах):

   $\longrightarrow \longrightarrow$ или $\longrightarrow \longleftarrow$

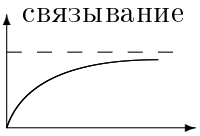
   В таком случае можно улучшить алгоритм: стартовать на со случайных позиций, а только с палидромов (или повторов), допуская некоторое количество ошибок. Тогда результат будет лучше.

11.8 Отступление про палиндромы

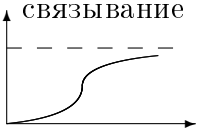
Часто бывает такое:

$\longleftarrow \longrightarrow \longleftarrow \longrightarrow$

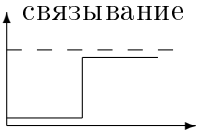
Так бывает если связывающий белок — димер. Зачем этому белку быть димером? Рассмотрим зависимость связывания от концентрации:



— Михаэлисовская зависимость (для белка-мономера)



— чувствительность лучше (для белка-димера)



— идеальный варинт (самая лучшая чувствительность)

В последнем случае связывания нет при концентрациях ниже пороговой, а после достижения порога связывание полное. При таких условиях достигается наилучшая чувствительность. Как видно из графиков, зависимость для димера ближе к идеальной, чем зависимость для мономера.

11.9 Распознавание образов

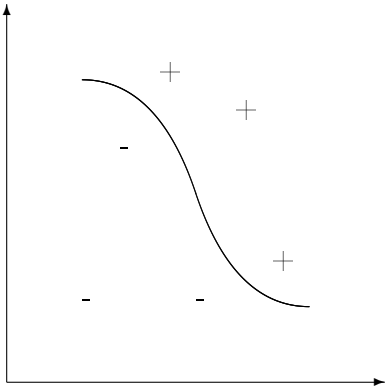
Способоы:

1. построить логическую формулу (множество опытов дискретно)

2. SVM (непрерывное пространство)

11.9.1 SVM (метод опорных векторов)

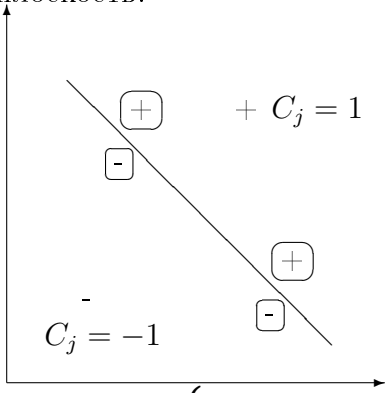
Есть декартово пространство. Есть набор наблюдений (точек в этом пространстве), при которых опыт дал положительный результат, и тех, при которых отрицательный. (Например, при каких условиях произошел инфаркт; была обнаружена нефть; распознаваемая буква оказалась буквой А)



цель: провести такую линию (разделяющую поверхность)

P — количество распознающих параметров (описывающих разделяющую поверхность), N — число наблюдений. Правило: $P \leq \sqrt{N}$

Проведем поверхност, характеризующуюся наименьшим количеством параметров — плоскость.



Ввели $C_j = \begin{cases} 1 & \text{для } + \\ -1 & \text{для } - \end{cases}$

W — пространство

Уравнение плоскости: $\sum w_i a_i - b = 0$

Правило:

$$\sum (w_i a_i - b) c_i > P$$

Абсолютное значение P не имеет значения, так как можно пропорционально увеличить a и b, поэтому:

$$\sum (w_i a_i - b) \geq 1$$

Ещё требуем $\sum a_i^2 \Rightarrow min$, чтобы плоскость шла посередине, а не перекашивалась в одну из сторон.

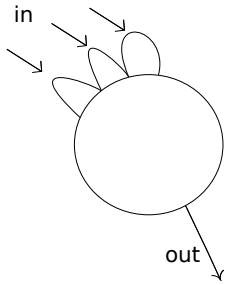
Это задача квадратичного программирования. Плоскость лежит на опорных векторах (на схеме взяты в рамку).

Наш PWM для распознавания сайтов имеет тот же вид: мы суммируем наблюдения с коэффициентами. Однако кроме положительных наблюдений (сайты) у нас

ещё много отрицательных; их очень много, что сильно увеличивает набор рассматриваемых объектов.

11.9.2 Искусственные нейроны

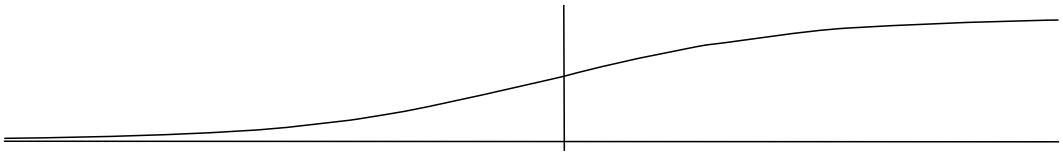
Исследования механизмов обработки изображений в глазу вдохновили на создание искусственного нейрона:



$$out = f(\sum x_i w_i)$$

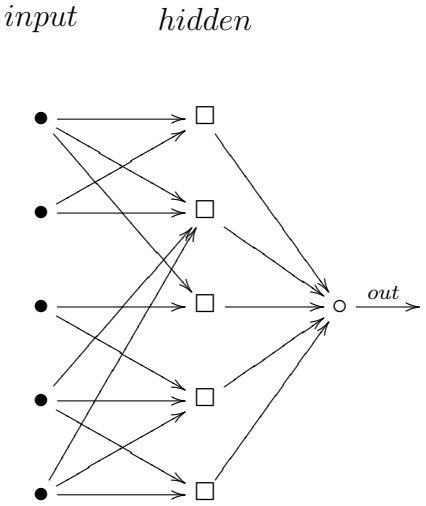
где f — логистическая кривая:

$$f = \frac{1}{1 + e^{-x}}$$



Надо подобрать w_i так, чтобы нейрон хорошо распознавал.

Описанная система состоит из одного нейрона. Более продвинутая система:



Трехслойная нейронная сеть широко используется. Нейронные сети были реализованы даже на уровне микропроцессоров. (Интересно, что они продолжают работу при сломанных 1/3 внутренних нейронов.)

Кажется, недостаток нейронных сетей в том, что неясен смысл коэффициентов (в отличие от HMM или PWM).

12 РНК

12.1 Функции РНК

1. тРНК
2. мРНК
3. рРНК
4. малые ядерные
5. малые ядрышковые
6. микро
7. si
8. switch – рибопереключатели
9. pri – праймеры для репликации ДНК
10. pico
11. рибозимы
12. геномы вирусов
13. регуляция промоторов
14. теломерная
15. хроматин-ассоциированная
16. антисмысловая
17. самовырезающиеся интроны I и II
18. гидовая
19. репарация
20. IRES
21. транспортно-матричная

12.2 Сведения о РНК

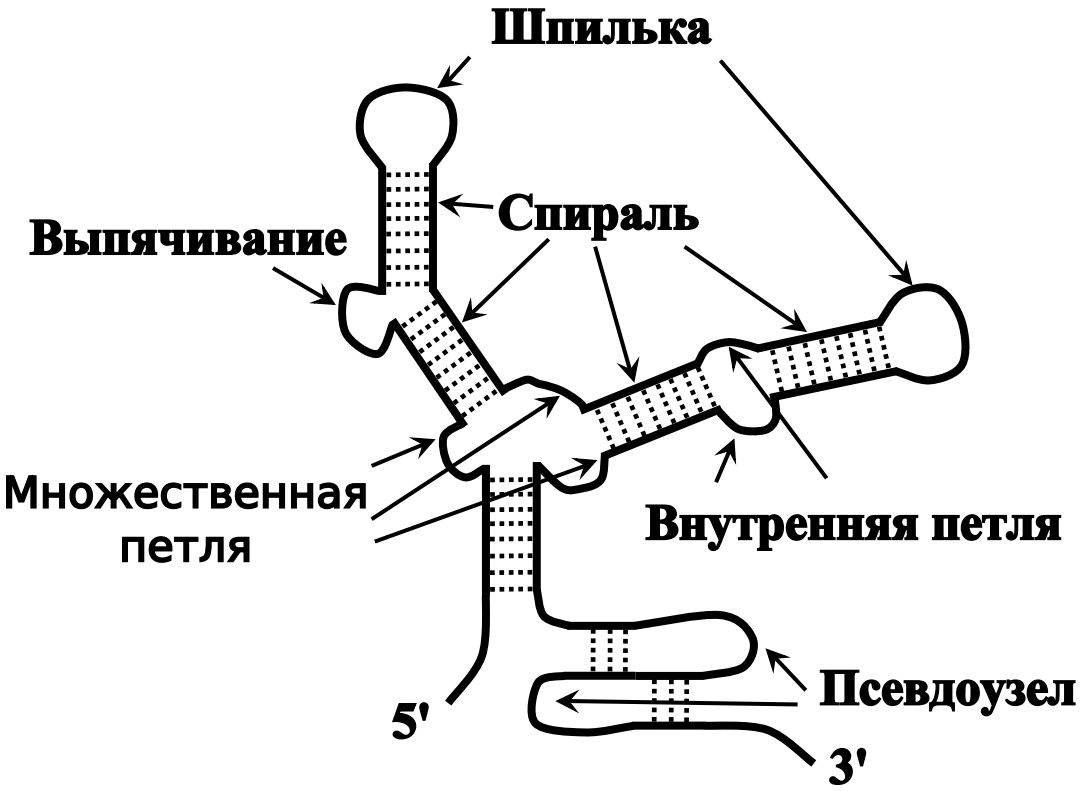
В геноме человека кодирующая часть составляет 3%, но функциональными являются 30-50%. Бóльшая часть РНК одноцепочечная, что делает её структуры очень разнообразными.

В РНК в А-форме на виток приходится 10 оснований, В ДНК в В-форме на виток приходится 11 оснований.

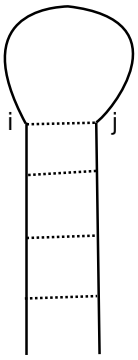
РНК более стабильна структурно, но менее устойчива химически, поэтому наследственная информация хранится в ДНК.

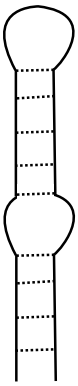
12.3 Вторичная структура РНК

Исходя из кривых плавления, основной структурой для РНК является вторичная, а не третичная.



Иногда шпильками называют такие структуры:



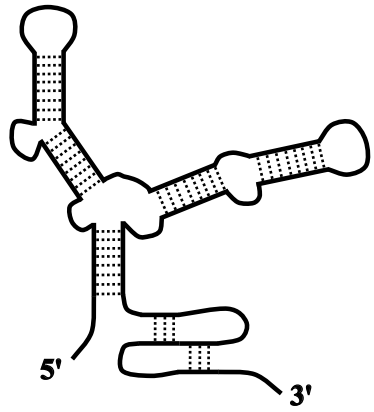


или

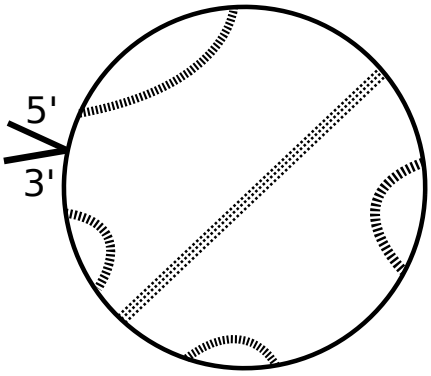
но это не соответствует строгому определению.

12.3.1 Как хранят вторичную структуру

1. топологическая схема



2. Перечисление всех спиралей и псевдоузлов
3. Массив спаренных оснований
4. Круговая диаграмма



На круговой диаграмме последовательность РНК нанесена на окружность, а хорды соответствуют спариваниям между основаниями. Пересечения хорд соответствуют псевдоузлам. Круговая диаграмма удобна для построения алгоритмов над вторичной структурой.

12.4 Предсказание вторичной структуры РНК по последовательности

Rfam – база для проверки предсказаний вторичной структуры. Золотым стандартом является тРНК: тРНК много, для многих есть рентген, кроме того, тРНК консервативны.

Ещё можно использовать рРНК, но они длиннее (порядка 1000, в то время как тРНК порядка 100), поэтому с вычислительной точки зрения в качестве стандарта удобнее использовать тРНК.

Часто псевдоузлы не рассматривают, так как:

1. нет эффективных алгоритмов, учитывающих псевдоузлы;
2. нет оценки энергии для псевдоузлов

12.4.1 Комбинаторный алгоритм, разрешающий псевдоузлы

Всего может быть образовано $\frac{L^2}{2}$ взаимодействий. Построим граф, у которого вершины соответсвуют взаимодействиям, а ребро проводится между совместимыми взаимодействиями. Совместимые взаимодействия – это взаимодействия, которые не противоречат друг другу. Пример несовместимых взаимодействий:

Эти взаимодействия несовместимы, так как в них участвует одно и то же основание.

Вторичной структуре соответствует клика в этом графе.

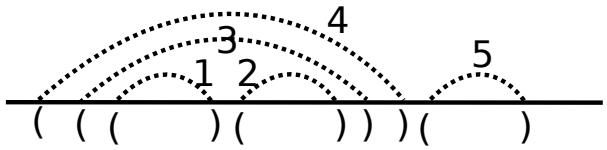
Недостатки:

1. поиск клик – NP-полная задача;
2. если граф целиком не является кликой, клик в нем может быть очень много, приходится решать, какая из них лучше.

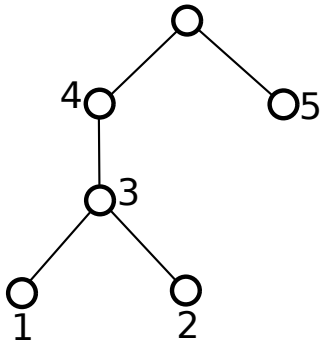
Преимущество этого алгоритма в том, что он разрешает псевдоузлы. К алгоритму может быть применена оптимизация: рассматривать взаимодействия не отдельных остатков, а групп из 3 остатков. За счет этого сокращается число вершин в графе и упрощается поиск клик.

12.4.2 Оценка количества структур без псевдоузлов

Количество вторичных структур без псевдоузлов для бернулиевской последовательности длины L: 1.8^L. Вторичная структура без псевдоузлов может быть представлена в виде правильной скобочной структуры:



Если есть псевдоузлы, количество «(» всё равно будет равно количеству «)», но скобочная структура не будет правильной. Скобочной структуре соответствует дерево:



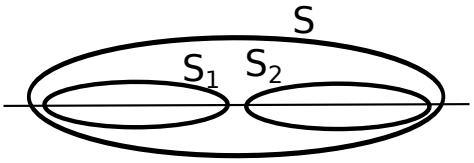
Количество правильных скобочных структур: 1.8^L.

12.4.3 Алгоритм Нуссинофф, не разрешающий псевдоузлы

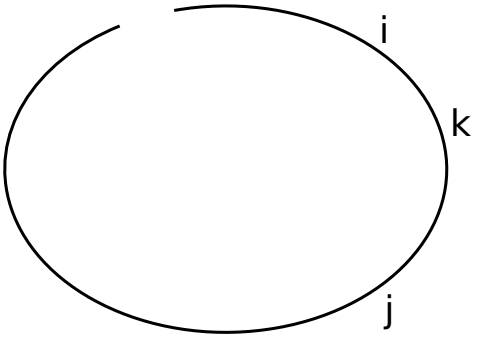
Задачу предсказания удобно сводить к задаче оптимизации. Будем оптимизировать число спаренных оснований. (Можно вводить веса за спаривания: спаривание GC крепче, чем AT.)

Структура S – набор спаренных оснвоаний. $|S|$ – мощность S, количество спаренных оснований.

Если $S = S_1 \cup S_2$ (то есть, S может быть разбита на непересекающиеся структуры S_1 и S_2), $|S| = |S_1| + |S_2|$:



S_{ij} – оптимальная структура на сегменте $[i, j]$.



Возможны следующие случаи:

1. $i \diamond j + S_{i+1,j-1}$ (если i и j спарены)
2. $S_{kj} + S_{i,k-1}$ (если j и k спарены)
3. $S_{i,j-1}$ (j ни с кем не спарена)

Заметим, что ни в одном из случаев не требуется информация о структуре бóльшего размера, что позволяет использовать динамическое программирование, начиная с коротких отрезков $[i, j]$ и постепенно их увеличивая.

$$S_{ij} = \max \begin{cases} 1 + S_{i+1,j-1} & i \diamond j \\ S_{i,j-1} & \\ \max\limits_{k}(1 + S_{k+1,j-1} + S_{i,k-1} & \end{cases}$$

(1 – вес спаривания, вместо него можно использовать более сложный вес, например, учитывающий, что пара GC крепче AT.)

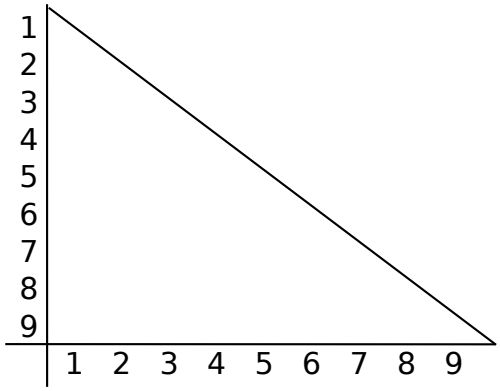
i доводят до 0, j – до конца. Сложность алгоритма: $L^2 L = L^3$

12.4.4 Восстановление структуры по матрице спариваний

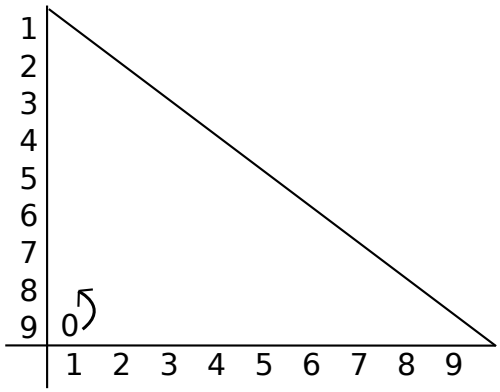
Находится вес вторичной структуры с максимальным количеством (или весом) спаренных оснований. Чтобы найти саму эту структуру, надо на каждом шаге $[i, j]$ запоминать, с кем спаривается i, то есть k. Эти числа составляют матрицу обратных переходов:

π_{ij} – с кем спарен j или 0, если ни с кем не спарен.

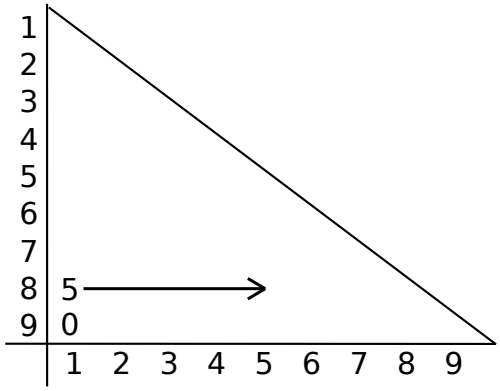
Эта матрица треугольная.
Рассмотрим пример.

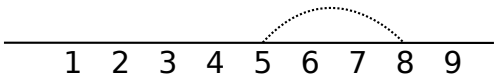


Начнем просматривать матрицу с левого нижнего угла. Эта ячейка соответствует паре первого и последнего оснований.

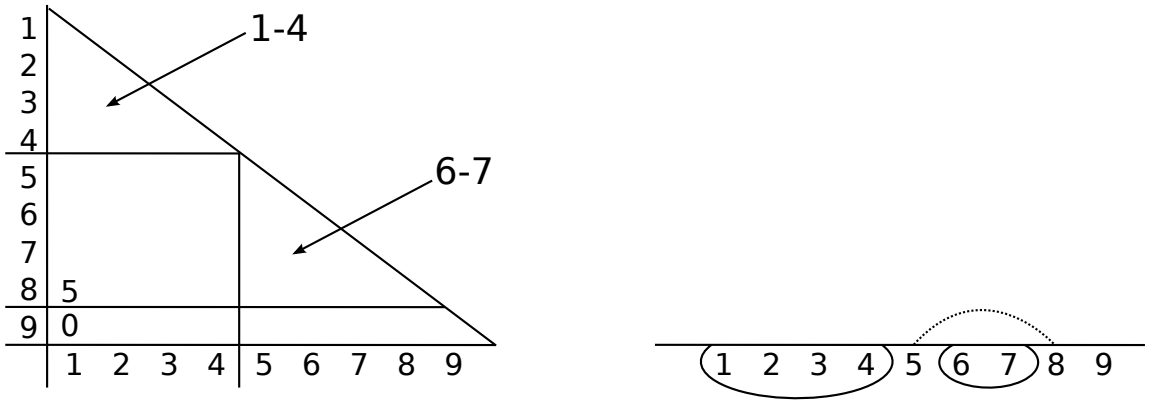


В этой ячейке стоит 0, значит первое и последнее основания не спарены. Поднимаемся вверх на один шаг.

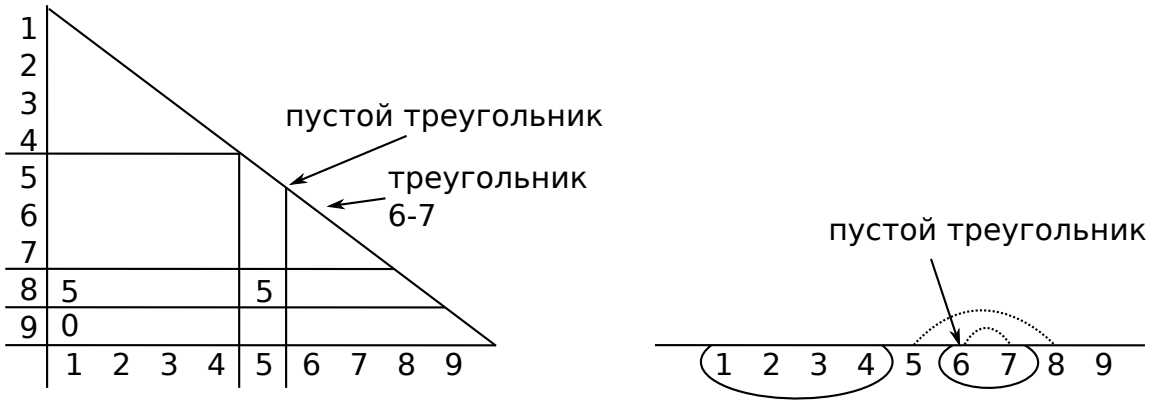




В этой ячейке записано число 5, что говорит, что основание 5 спарено с основанием 8. Структура разбивается на две непересекающиеся группы: 1-4 и 6-7. Этим группам в матрице соответствуют треугольники:



Впоследствии эти группы рассматриваются независимо. Рассмотрим дальнейшее «раскручивание» группы 6-7:



Выяснилось, что основание 6 спарено с оснвоанием 7. Данный случай можно рассматривать, как предыдущий: в данном случае один из треугольников пустой, а второй включает только сами основания 6 и 7. Оба треугольника не нуждаются в дальнейшей обработке.

Псевдокод:

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

12.5 Энергия вторичной структуры

Энергия считается как сумма энергий элементов структуры. Энергии элементов структуры можно узнать из эксперимента.

12.5.1 Спирали

В спирали энергия содержится в основном в стэкинг-взаимодействиях. Стекинг существует даже в однонитиевом состоянии. Энергии стэкинга всех возможных пар можно занести в таблицу:

	GC	AU	...
GC			
AU			

12.5.2 Петли

Есть ли энергия у петель?

В качестве энергии рассматривается $\Delta G = \Delta H - T\Delta S$. $\Delta S \sim$ логарифм количества микросостояний, которые реализуют это макросостояние.

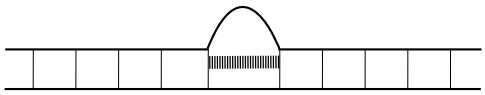
Энергия запасена в концах петли. Если отпустить концы петли, высвободится энергия (возрастет подвижность, то есть количество микросостояний).

Энергия петли:

$$\Delta G = B + \frac{3}{2} k T_B \ln(L)$$

где L – длина петли, значение B различается для разных петель.

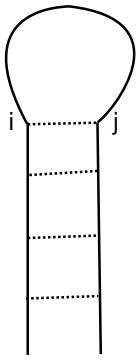
В выпячивании иногда сохраняется стэкинг:



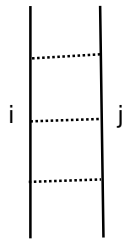
12.5.3 Zucker

Доступные нуклеотиды – это нуклеотиды, не заслоненные спариванием.

1. Вся петля доступна для пары i, j:



2. Для этих все недоступно



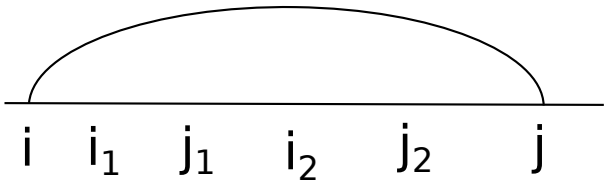
$$\Delta G_{structure} = \sum \Delta G_{loop}$$

Однако алгоритм рассматривает структуры от меньшей к большей:

это спаривание не выгодное, поэтому и все последующие не могут образовываться.

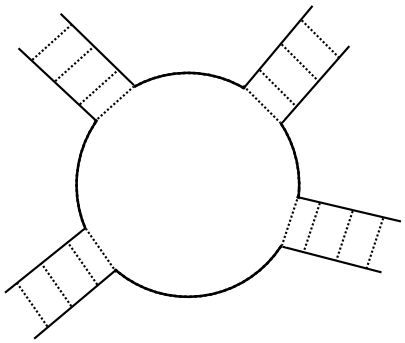
W_{ij} – энергия наименьшей структуры на ij, V_{ij} – энергия наименьшей структуры на ij при условии, что $i \diamond q$

$$W_{ij} = \min \begin{cases} W_{i+1,j} & \text{j — noncoupled} \\ W_{i,j-1} & \text{j — noncoupled} \\ V_{ij} & \\ \min\limits_{k}(W_{ik} + W_{k+1,j}) & \text{j — coupled} \end{cases}$$



$$V_{ij} = \min_{i<i_1<j_1<i_2<j_2<j} \Delta G^{loop} + \sum V_{i_k j_k}$$

Сложность 2^L.



Проблема возникает именно в мультипетлях:
Чтобы снизить сложность алгоритма, можно не учитывать ΔG^{loop}.

12.5.4 mFold

Программа mFold – стандарт для нахождения вторичной структуры. Находит правильно $60-80\%$. У mFold есть возможность заранее спарить два нуклеотида (условное выравнивание, то есть заранее попросить выровнить два остатка). Однако есть опасность получить неправильную структуру, подав mFold неправильные ограничения. Причиной такого низкого качества сворачивания является возможность стабилизации РНК в клетке после достижения правильной структуры.

Теория: правильная стуктура имеет не только наименьшую энергию, но еще отстоит от других структур по энергии. (У ДНК этого зазора нет.)

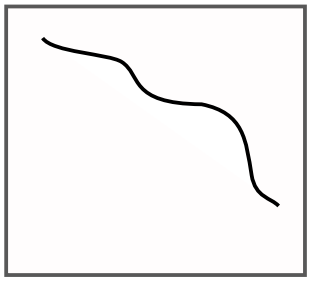
$0,6$ккал/моль $= kT$ при нормальных T, это ошибка из-за тепловых флуктуаций. Если оптимальная структура 35,1 ккал/моль, правильная может иметь энергию 35.

12.5.5 Субоптимальная структура

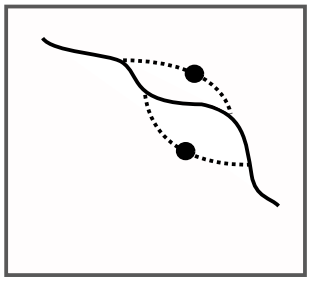
Рассчитаем для РНК статсумму Z: $Z = \sum_{\text{все состояния}} e^{-\frac{\Delta G}{kT}}$
$Z_{ij} = \sum_{\text{все состояния, где } i \diamond j}$, где Z_{ij} — условная статсумма

Хотя структур много, рассчитать Z несложно (сравнимо с расчетом количества путей в графе).
$p(i \diamond j) = \frac{Z_{ij}}{Z}$ — вероятность того, что $i \diamond j$.

Будем абстрактно изображать структуру, как путь:



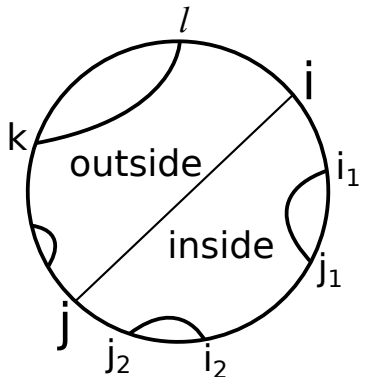
Затем, имея $p(i \diamond j)$, с помощью Монте-Карло выбираем пару ij и проводим через них субоптимальную структуру



Можно заказать оптимальную и несколько субоптимальных.

12.5.6 Как искать условные статсуммы

Полная статсумма: $Z = Z_{oL}$
$e^{\Delta G_1 + \Delta G_2} = e^{\Delta G_1} e^{\Delta G_2}$
Возможны опрерации сложения и умножения, поэтому статсумма является удобным объектом.



$$Z_{ij} = \sum e^{-\Delta G^{in}} e^{-\Delta G^{out}} =$$
$$= \underbrace{\sum e^{-\Delta G^{in}}}_{I_{ij}} \underbrace{\sum e^{-\Delta G^{out}}}_{O_{ij}}$$

$$I_{ij} = \sum_{i<i_1<j_1<i_2<j_2<j} e^{-\frac{\Delta G^{loop}}{kT}} \prod_k I_{i_k<j_k} =$$

$$= \sum e^{-\frac{\Delta G^{loop}}{kT}} \prod_k I_{i_k j_k}$$

$$O_{ij} = \sum_{kl} I_{jk} I_{li} O_{lk} e^{-\frac{\Delta G^{loop}}{kT}}$$

I считается изнутри наружу. O – снаружи внутрь (сравни с FB).

$$O_{0L} = I_{0L} = Z$$

Однако не очень удобно получать несколько структур.

12.6 Консервативная вторичная структура (алгоритм Санкофф)

Одновременно строит выравнивание и вторичную структуру. Однако $O(L^6)$, поэтому не используется.

Предположим, что множественное выравнивание есть.

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

Колонки не консервативны, но они связаны, они образуют комплементарную пару.

1. Выявить согласованные пары колонок
2. Строим структуру с максимальным количеством согласованных пар (подобно алгоритму Нуссинофф)

Не требуется полная согласованность т.к.:

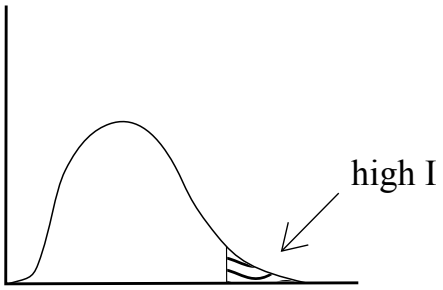
1. последовательность могла быть не с той же функцией
2. выравнивание неправильное
3. неканоническое спаривание

Поэтому считаем предпочтения этих пар. Считаем взаимную информацию двух колонок:

$$I_{ij} = \sum_{\alpha\beta} f_{\alpha\beta}(ij) \log \frac{f_{\alpha\beta}(ij)}{f_\alpha(i) f_{\beta(j)}}$$

I_{ij} – взаимная информация, f – частоты

I_{ij} большое $\Rightarrow$ образуются комплементарные пару. Чтобы знать, какое значение I велико, надо знать распределение I для случайных колонок:



Чтобы получить случайное распределение, перемешивают основания внутри колонок.

13 Вторичная структура РНК

Марков изучал структуру языков, таким образом, возникли Марковские модели (ММ). Хомский в 50х годах придумал другую систему описанию языков.

Грамматики – набор правил генерации текста. (Аналогично Марковским моделям, сначала просто, а затем и скрытым).

Приведем несколько определений.

1. Терминальный символ – это буква, которая может появиться в тексте (a, b, c, ε).

2. Нетерминальный символ, любой другой.

13.1 Пример. Регулярная грамматика

$S \to W_1$
$W_1 \to aW_2$(нетерминальный символ – несгенерированный фрагмент текста)
$W_2 \to bW_3$
$W_2 \to \varepsilon$

Т.е. еще раз, терминальные символы непосредственно генерируемые символы текста, а нетерминальные те, что обозначают определенные правила перехода между терминальными символами. Символы начала и окончания текста являются терминальными. Иногда, если в нескольких правилах находятся одни и те же нетерминальные символы, то такие строчки можно объединить в одну.
$W_1 \to aW_2$
$W_1 \to bW_2 \to (W_1 \to [a, b, c]W_2)$
$W_1 \to cW_2$

Как видно, в данной грамматике мы можем порождать символы только слева. Это является отличительным признаком регулярной грамматики (порождать символы либо только слева, либо только справа).

Правила типа $W \to aW'$ – приписываем слева нетерминальный символ (эквивалентно регулярному выражению). Есть правила, порождающие целую строку. (Например, если нетерминальный символ переходит в такой же нетерминальный.)

1. aba – подходит данной грамматике
2. aaa – не подходит (a b должны чередоваться)
3. ab – не подходит (так как заканчивается на a)

13.2 Контекстносвободные грамматика

$W \to aWaWWt$ – любая комбинация терминальных и нетерминальных символов. Регулярная грамматика является частным случаем контекстносвободной. С помощью данной грамматики мы можем порождать символы одновременно и влево и вправо. (Дальше увидим зачем нам это нужно.)

1. Также существует еще более общая грамматика – контекстнозависимая: $A_1WA_2 \to A_1W'A_2$
2. Еще более общая грамматика – свободная: $A_1WA_2 \to$ любая комбинация символов

Вспомним способ записи вторичной структуры РНК. Это скобочная формула. Итак, эту формулу можно породить с помощью контекстносвободной грамматики.

$$W_1 \to (W_2)$$

Распишем самую простую грамматику, описывающую шпильку (Z):

1. $S \to W_1$ – начало порождения текста
2. $W_1 \to [acgt]W_1$ – порождение хвоста произвольной последовательности
3. $W_1 \to aW_2t$
4. $W_2 \to aW_2t$
5. $W_2 \to cW_2g$
6. $W_2 \to [acgt]W_3$

7. $W_3 \to [acgt\varepsilon]W_3$

Z – набор правил, образующих полную шпильку. Эта грамматика лишь упрощенный вариант, здесь длина спирали неограниченна.

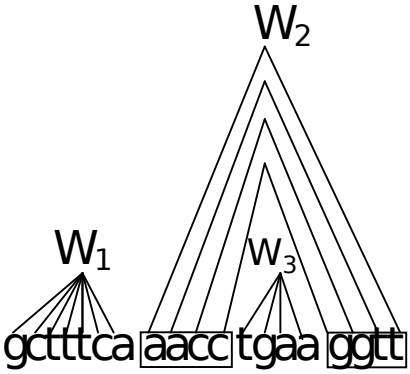
Данную грамматику можно применить только для порождения инвертированных «повторов» (комплементарных в данном случае). Прямые повороты таким образом получить нельзя. Можно применить для поиска консервативных структур РНК в геноме.

$Z_4 \to Z_2Z_1Z_3$ – это объединение трех шпилек в одну. В части $\to \ldots$ не может быть больше двух символов (каноническая форма грамматики). Во всех алгоритмах используется канонический вид.

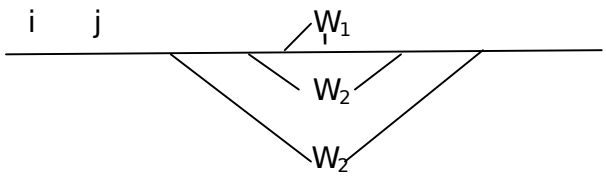
И такое правило: $W_1 \to aW_2t$ – можно переписать в каноническом виде:

$$W_1 \to aW_2$$

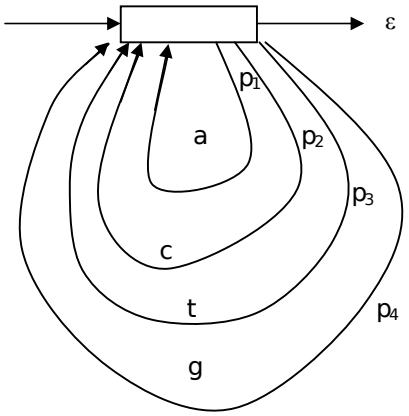
$$W_2 \to W_3t$$



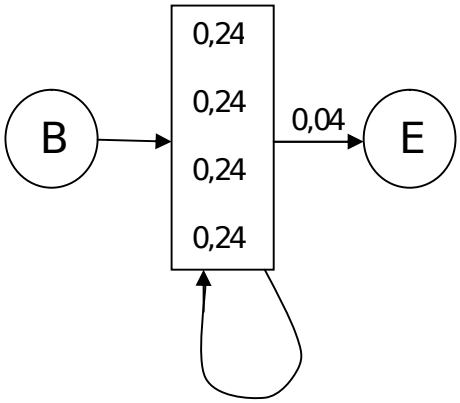
Напоминает скобочную формулу. Называется дерево разбора. Теперь рассмотрим алгоритм поиска. Подбор грамматики. Есть последовательность. Начинаем в каждой букве последовательности пытаемся закрепить правило W1. Над каждой точкой получаем небольшую притирку, если место выбрано хорошо, то выше можем строить следующее правило.



На каждое из правил можем повесить вероятность, получаем стохастическую грамматику.

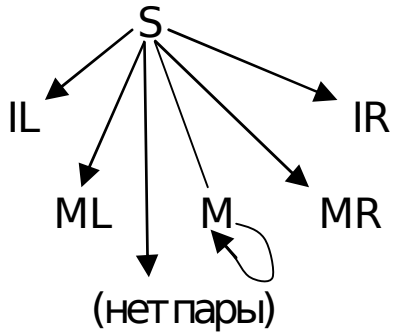


P	
0.24	$w \to aw$
0.24	$w \to cw$
0.24	$w \to gw$
0.24	$w \to tw$
0.24	$w \to \varepsilon$

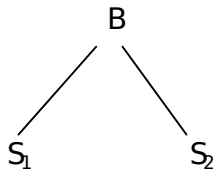


Стохастическая (вероятностная) контекстносвободная грамматика аналогична конечным автоматам и HMM. По сути контекстносвободная грамматика – это обобщение HMM.

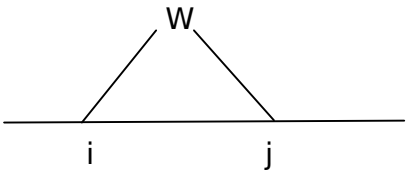
1. $w \to aw$ – 4 вероятности на каждую букву
2. $w \to twa$ - 16 вероятностей
3. $w \to wa$ – 4
4. $w \to ww$ – 1 вероятность двух нетерминальных символов
5. $w \to \varepsilon$ – 1 окончание
6. $s \to w$ – 1 начало, вероятность = 1

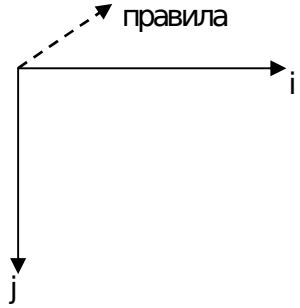


1. IL – символ слева
2. IR – символ справа
3. M – символы с двух сторон
4. MR – справа от «повторов» (?)
5. ML – слева от «повторов» (?)



Нужно найти наилучшее вхождение строки в грамматику. В матрицу вероятность того, что правило сработало. В итоге время кубическое.





$$P(V, i, j) = \max[p(x, i, k) \cdot p(y, k + 1, j) \cdot t_v]$$

$P(v, 0, k) \rightarrow$ максимизируем вероятность (аналог Витерби).