

Биоинформатика последовательностей

Андрей Александрович Миронов

2021

Оглавление

1	Байесова вероятность	11
1.1	Что такое вероятность	11
1.1.1	Понятие вероятности.	12
1.2	Некоторые обозначения	14
1.3	Теорема Байеса и оценивание параметров	16
1.3.1	Примеры вычислений	17
1.3.2	Примеры априорных распределений	18
1.4	Оценки параметров.	20
1.4.1	L-оценка	21
1.4.2	МАР-оценка	21
1.4.3	Е-оценка	22
1.5	Распределение Дирихле	22
1.5.1	Оценки параметров	24
1.6	Другие распределения	25
1.6.1	Распределение Пуассона	26
1.6.2	Смеси Дирихле	28
1.7	Максимизация ожидания (ЕМ)	29
1.8	Заключительные замечания	30
2	Модели последовательностей	31
2.1	Модели природных последовательностей	31
2.1.1	Влияние модели на оценку	31
2.1.2	Природные модели последовательностей.	32
2.2	Бернуллевская модель	34
2.3	Марковские модели	35
2.3.1	Модели высших порядков	36
2.4	Оценка моделей	37
2.5	Частоты встречаемости слов	39
3	Выравнивания	43
3.1	Постановка задачи	43
3.1.1	Число выравниваний	44
3.2	Глобальное выравнивание	45
3.2.1	Редакционное расстояние	45

3.2.2	Алгоритм Нидльмана-Вунша	46
3.2.3	Алгоритм Миллера-Майерса	47
3.2.4	Выравнивание в полосе	49
3.3	Локальное выравнивание	49
3.3.1	Оптимальное локальное выравнивание	50
3.4	Штрафы за делеции	51
3.4.1	Общие штрафы	51
3.4.2	Аффинные штрафы	52
3.4.3	Невыравниваемые фрагменты	53
3.5	Статистики выравниваний	54
3.5.1	Наибольшая общая подпоследовательность	54
3.5.2	Наибольшее общее слово	55
3.5.3	Два типа поведения выравнивания	56
3.6	Матрицы сопоставления аминокислотных остатков	56
3.6.1	Матрицы PAM	58
3.6.2	Матрицы BLOSUM	59
3.7	Быстрые методы поиска сходства последовательностей	60
3.7.1	FASTA	60
3.7.2	BLAST	61
3.7.3	Выравнивание цепочек (Chain alignment)	63
3.7.4	Выравнивание геномов	64
4	Преобразование Барроуза-Уиллера	67
4.1	Суффиксный массив	67
4.2	Концепция	68
4.3	Быстрое восстановление исходного текста	70
4.4	Поиск образца с помощью преобразования BWT	72
4.5	Оптимизация памяти	75
4.5.1	Программистские приемы	76
4.6	Сжатие суффиксного массива	77
4.7	Итоги	79
5	Сборка геномов	81
5.1	Постановка задачи	81
5.1.1	Дополнительные трудности	82
5.1.2	Основные термины	84
5.2	Графовая постановка задачи. Гамильтонов путь	85
5.3	Графовая постановка задачи. Граф де-Брюина	87
5.4	Дополнительная информация и построение скаффолдов	88
6	Скрытые марковские модели	91
6.1	Вводные замечания	91
6.1.1	Пример	91
6.2	Скрытые Марковские Модели	93
6.2.1	НММ – Генеративная модель	93

6.2.2	Граф состояний	94
6.2.3	Декодирование. Алгоритм Витерби	95
6.2.4	Апостериорное декодирование. Алгоритм Вперед-назад (FB)	97
6.2.5	Итеративное предсказание	100
6.3	Биологические примеры	100
6.3.1	Сайты рестрикции	100
6.3.2	Сайты связывания транскрипционных факторов	102
6.3.3	Трансмембранные сегменты в белках	103
6.3.4	НММ – Лего	104
6.3.5	Предсказание генов в прокариотах	106
6.4	Оценка параметров модели	108
6.4.1	Обучение с учителем	108
6.4.2	Обучение без учителя	110
6.5	Пример применения НММ к эпигенетике	113
6.5.1	Вводные замечания	113
6.5.2	Алгоритм Chrom-НММ	114
6.6	Задачи	115
7	НММ выравнивание	117
7.1	Постановка задачи	117
7.2	Алгоритм Витерби для выравниваний	118
7.3	Максимизация правдоподобия	121
7.4	Апостериорное декодирование	124
7.4.1	Алгоритм Вперед-Назад	125
7.4.2	Субоптимальное выравнивание	128
7.5	Локальное выравнивание	131
8	Профили	135
8.1	Вероятность, энтропия, информация	135
8.1.1	Комбинаторная энтропия	135
8.1.2	Энтропия и информация	137
8.1.3	Взаимная информация	138
8.2	Профили	139
8.2.1	Консенсус	140
8.2.2	Регулярное выражение	140
8.2.3	Частотный профиль и правдоподобие	140
8.2.4	Информационное содержание и Лого	141
8.3	НММ профиль	142
8.3.1	Оценка параметров	144
8.3.2	Псевдо-счетчики	144
8.3.3	Взвешивание последовательностей	146
8.4	Поиск мотивов	151
8.4.1	Алгоритм MEME	153
8.4.2	Gibbs Sampler	154
8.4.3	Расширения, дополнения и обобщения	155

9 Множественное выравнивание	157
9.1 Постановка задачи	157
9.1.1 Качество выравнивания	158
9.2 Динамическое программирование	160
9.3 Прогрессивное выравнивание	161
9.3.1 Основной алгоритм	161
9.3.2 Улучшение выравнивания	163
9.3.3 t-Coffee	164
9.3.4 Muscle	164
9.3.5 ProbCons	165
9.3.6 MAFT	165
9.3.7 Dialign	166
10 Вторичная структура РНК	167
10.1 Функции РНК	167
10.2 Структура РНК	168
10.2.1 Элементы вторичных структур РНК	170
10.2.2 Представление структур РНК	171
10.3 Постановка задачи	172
10.4 Максимизация числа спаренных оснований. Алгоритм Нусси-нофф	173
10.4.1 Динамическое программирование	174
10.4.2 Восстановление структуры	175
10.4.3 Условно-оптимальная структура	176
10.5 Энергия вторичной структуры	178
10.5.1 Алгоритм Зукера	182
10.5.2 mFold – качество предсказания	184
10.5.3 Статистическая сумма	185
10.5.4 Субоптимальные структуры	187
11 Консенсусные структуры РНК	189
11.1 Консенсусные вторичные структуры	189
11.1.1 Метод ковариаций	189
11.1.2 Одновременное выравнивание последовательностей и структур	191
11.2 Порождающие грамматики	191
11.2.1 Иерархия грамматик	193
11.3 Контекстно-свободные грамматики и структура РНК	194
11.3.1 Разбор строки	195
11.3.2 Вероятностные грамматики	196
11.3.3 Вероятностные контекстно-свободные грамматики	198
11.4 Описание структуры РНК. Ковариационные модели	200

Предисловие

Биоинформатика последовательностей возникла практически сразу после того, как появились технологии чтения последовательностей. В начале научились читать аминокислотные последовательности белков, за что в 1958 году Фредерик Сэнгер получил свою первую Нобелевскую премию по химии. Чтение аминокислотных последовательностей тогда была довольно сложной биохимической процедурой. К 1965 году накопилось достаточно большое количество расшифрованных аминокислотных последовательностей. Впервые к анализу этих последовательностей был применен компьютер. Это сделала замечательная исследовательница Маргарет Дейхофф. Она создала первую базу данных (на самом деле просто сборник) последовательностей белков, которая содержала 65 расшифрованных последовательностей. В 1966 году она впервые использовала компьютер для построения выравнивания последовательностей. Эти выравнивания были использованы для построения филогении. Тогда использовался метод максимальной экономии.

В 1970 году появилась классическая работа Нидльмана и Вунша, где была сформулирована задача выравнивания и построен соответствующий алгоритм, хотя то, что мы сегодня называем алгоритмом Нидльмана-Вунша был предложен Дэвидом Санкофф в 1972 г.

Другая история связана с предсказанием вторичной структуры РНК. По аналогии с биохимическим методом определения аминокислотных последовательностей Роберт Холли разработал метод определения последовательности транспортных РНК. Параллельно с лабораторией Холли работала также группа Сэнгера, но Холли опередил. В то время определение последовательности в 70 нуклеотидов требовало большой биохимической работы и эта работа также была удостоена Нобелевской премии 1968г. Получив последовательность, Роберт Холли заметил, что эту последовательность с использованием Уотсон-Криковских взаимодействий можно уложить в структуру типа Клеверный лист. Эта работа была сделана вручную без использования компьютера, однако ее также можно отнести к пионерским работам по биоинформатике.

В 1977 году две независимые группы придумали два принципиально разных метода секвенирования ДНК. Это метод Максама-Гилберта, основанный на специфическом разрезании полинуклеотидной цепочки и метод Сэнгера, основанный на обрыве синтеза копии ДНК. Метод Сэнге-

ра выдержал испытание временем, а опубликованный несколько ранее метод Максама-Гилберта довольно быстро вышел из употребления. Использование, по тем временам высокопроизводительных, методов секвенирования позволило достаточно быстро получить целый ряд последовательностей. Первой опубликованной последовательностью была последовательность плазмиды pBR322. Потом были секвенированы бактериофаг ϕ X84, вирус SV40 и еще несколько полных последовательностей длиной порядка 4-5 тыс нуклеотидов. Последовательности, также как и сейчас, читались фрагментами и собирались вручную – информация с гелей распечатывалась на пишущей машинке, разрезалась на бумажные ленточки, которые раскладывались на полу и исследователи из этих фрагментов по перекрытиям собирали полные последовательности.

Датой рождения биоинформатики как отдельного направления молекулярной биологии я считаю январь 1982 года. В этот момент вышел первый номер журнала *Nucleic Acids Res* полностью посвященный компьютерному анализу последовательностей.

Предлагаемый Вашему вниманию текст представляет собой конспект лекций по биоинформатике последовательностей, читаемый на Факультете биоинженерии и биоинформатики МГУ им. М.В.Ломоносова. Курс читается в течение одного семестра и сопровождается домашними заданиями и журнальным клубом, на котором разбираются современные статьи по молекулярной биологии, где используются методы биоинформатики. Целью курса является введение в основные понятия и методы классической биоинформатики последовательностей, а также развитие умения понимать биоинформатические тексты. Здесь есть некоторое количество формул, однако, несмотря на то что иногда они достаточно громоздки, они достаточно простые. Хочется надеяться, что разбор этих формул разовьет необходимые навыки у читателя. В этом курсе нет практики использования различных программ, поскольку это является предметом другого курса – курса практической биоинформатики. Текст содержит следующие главы:

Байесова вероятность. Глава, посвященная Байесову подходу к вероятности является ключевой. Байесов подход применяется в тех случаях, когда количество наблюдений не велико, а это случается весьма часто в прикладных задачах, в том числе и в биоинформатике. В настоящее время этот подход получает все большее распространение.

Модели последовательностей. В этой главе дается представление о вероятностных моделях, в том числе о вероятностных моделях последовательностей. Дано представление о связи размера обучающей выборки и качества моделей. Показано, что даже в простейших бернуллиевых моделях есть нетривиальные эффекты.

Выравнивания. Выравнивание двух последовательностей – это одна из классических задач биоинформатики. В главе рассмотрены основные задачи выравнивания и алгоритмы построения выравниваний. Уделено внимание оценкам статистической значимости выравниваний.

Преобразование Барроуза-Уиллера. Это одна из самых сложных глав. Преобразование Барроуза-Уиллера применяется к некоторым специальным задачам выравниваний, а, точнее, к задаче поиска образца в тексте. Преобразование Барроуза-Уиллера применяется для картирования прочтений на известный геном и широко используется в программах обработки результатов секвенирования второго поколения.

Сборка геномов. Эта глава также имеет отношение к обработке данных секвенирования. В главе рассмотрены основные идеи, лежащие в основе алгоритмов и программ для сборки геномов из прочтений, полученных методом дробовика.

Скрытые марковские модели. Скрытые марковские модели это один из методов машинного обучения, получивших широкое распространение в анализе биологических последовательностей. В главе рассмотрены методы расшифровки последовательностей состояний и методы обучения моделей.

НММ выравнивание Здесь рассмотрены вероятностные подходы к задачам выравнивания – выравнивания рассматриваются как результат порождения некоторой моделью. Показаны преимущества вероятностного подхода.

Профили Профиль – это способ описать группу последовательностей, например, семейство белков или множество сайтов связывания транскрипционного фактора. Введены понятия энтропии, информации, взаимной информации. Описаны разные типы профилей, показаны методы обучения моделей профилей. Рассмотрены проблемы, связанные со взвешиванием последовательностей и выбором псевдо-счетчиков.

Множественные выравнивания Построение множественных выравниваний является важнейшей задачей при изучении белковых семейств. Рассмотрены основные идеи, применяемые в алгоритмах множественного выравнивания.

Вторичная структура РНК Предсказание вторичных структур РНК началось еще на заре молекулярной биологии. Впервые вторичная структура транспортной РНК была предсказана еще в 1965 году. Однако вторичную структуру могут образовывать практически все РНК. В главе рассмотрена проблема предсказания вторичной структуры по последовательности, описаны основные трудности существующих подходов.

Консенсусные структуры РНК Наиболее надежным способом предсказания биологически значимой вторичной структуры является поиск консервативных (консенсусных) структур. В главе рассмотрены две задачи. Первая – дано множество последовательностей, предсказать структуру. Вторая – дана структура, найти такую структуру в последовательности, например, найти в геноме все тРНК. Одним из универсальных методов решения такого типа задач является использование контекстно-свободных грамматик.

Предполагается, что читатель этого текста имеет следующие предварительные сведения и навыки: основы теории вероятностей; представления о молекулярной биологии, например знает аминокислоты; основы теоретической информатики. Для выполнения практических работ предполагаются навыки программирования.

Глава 1

Байесова вероятность

1.1 Что такое вероятность. Некоторые примеры

Зададимся вопросом – что такое вероятность? Есть определения разной степени строгости. Одно из них базируется на теории меры и σ -алгебрах. Другое, житейское, базируется на некотором здравом смысле – отношение числа успехов к числу испытаний, когда число испытаний стремится к бесконечности. Первое определение является основой аксиоматической теории вероятностей. Второе – апеллирует к интуиции. Однако не все так просто.

Парадокс двух конвертов. Предлагается простая игра – Вам предлагают два конверта с деньгами и говорят, что в одном конверте денег в два раза больше, чем в другом. Вы задумываетесь, какой из конвертов выбрать. Допустим, в левом конверте лежит неизвестное Вам количество денег X . Сколько денег лежит во втором конверте? Вы знаете теорию вероятностей и пытаетесь оценить математическое ожидание количества денег в правом конверте. Вероятность того, что Вы выбрали конверт с маленькими день-



гами, а во втором конверте количество денег вдвое больше ($2X$), равна 0.5. Вероятность, что Вы выбрали конверт с большими деньгами, а во втором конверте денег вдвое меньше ($X/2$), тоже равна 0.5. Считаем математиче-

ское ожидание количества денег во втором конверте.

$$E(B) = P(2X) \cdot (2X) + P(X/2) \cdot (X/2) = 0.5 \cdot (2X) + 0.5 \cdot (X/2) = 1.25X$$

Но те же рассуждения можно провести, стартуя с конверта **В** и получить, что в конверте **А** денег больше, чем в первом!

Парадокс Симпсона. Представим себе, что некоторая фармацевтическая компания разработала лекарство, например, от офигения. Она провела испытания и получила результаты, показанные в таблице 1.1. Считаем эффективность (таблица 1.2) и приходим к парадоксальному выводу – лекарство лечит и мужчин и женщин, но людей калечит!

Таблица 1.1: Результаты испытаний

	Мужчины		Женщины	
	Лечение	Контроль	Лечение	Контроль
Выздоровели	50	4	10	80
Всего	90	12	12	120

Таблица 1.2: Оценка эффективности

	Мужчины	Женщины	Все
Лечение	$\frac{50}{90} = \frac{5}{9}$	$\frac{10}{12} = \frac{5}{6}$	$\frac{60}{102} = \frac{20}{34}$
Контроль	$\frac{4}{12} = \frac{3}{9}$	$\frac{80}{120} = \frac{4}{6}$	$\frac{84}{132} = \frac{7}{11} = \frac{21}{33}$
Эффект	$\frac{5}{9} > \frac{3}{9}$ Лечит	$\frac{5}{6} > \frac{4}{6}$ Лечит	$\frac{20}{34} < \frac{21}{33}$ Калечит

1.1.1 Понятие вероятности.

Итак, математическое определение вероятности – это мера, определенная на пространстве событий. При этом ничего не говорится, откуда берется эта мера. Житейское определение – отношение числа успехов к числу испытаний. Представим себе, что Вам предлагают пройти по проволоке, натянутой через пропасть (рис.1.1). Сколько испытаний будет сделано? В этом случае Вы не сможете провести достаточно большое число испытаний, скорее всего хватит одного. Но все-таки есть, хоть и маленькая, вероятность того, что Вы пройдете до конца. В свое время великий Лаплас сказал (в вольном переводе) «*Вероятность – это здравый смысл и немного математики*». У теории вероятностей есть несколько прикладных источников.



Рис. 1.1: Какова вероятность того, что Вы пройдете по канату?

- Азартные игры.
- Страховое дело. При оценке страховой премии (платежа за страховку) Вам необходимо оценить вероятность страхового случая. Например, некто хочет застраховать автомобиль от падения метеорита. Ведь, хотя не было зарегистрировано ни одного случая, событие возможно. Какова должна быть страховая премия?
- Банковское дело. Выдача кредитов. Вам надо оценить вероятность того, что клиент вернет кредит и что не возникнет ситуации форс-мажора.

Что объединяет эти области деятельности? — **Деньги**. На самом деле вероятность — это каким количеством денег вы готовы рискнуть, чтобы получить результат, т.е. соотношение ожидаемого результата и количества вложенных денег. Отсюда следует, что вероятность понятие субъективное. По-видимому, единственное место, где существует объективная вероятность — это квантовая механика.

Еще одна важная особенность вероятности. Вероятность существует *до* события. Вероятность случившегося события равна 1. Здесь, кстати, опять есть переключка с квантовой механикой — наблюдение приводит к коллапсу волновой функции. Как перевести на этот язык частотную интерпретацию вероятности? Пусть после 1000 бросков монеты орел выпал 487 раз. Это позволяет предположить, что в *следующем броске* (которого еще нет) вероятность выпадения орла примерно 0.5 и, исходя из этого можно делать ставку, т.е. опыт позволил нам оценить свойство *монеты*. А если за 100 бросков орел выпал 100 раз. Какой вывод можно сделать? Есть два варианта — первый — монета кривая. Другой возможный вывод — смертельно не везет! Оба вывода имеют право на существование.

Кстати еще о деньгах. Допустим, ожидается некоторое событие – выборы президента (конечно, американского), футбольный матч, премия Оскар, ураган и пр. Для предсказания события создается биржа, где торгуются акции на разные исходы – победа республиканцев или демократов, счет в матче, победители премии и пр. Торговля происходит до события, а в момент события проигравшие акции аннулируются, а победившие забирают весь банк. Оказывается, что такого типа биржи дают предсказания с весьма высокой точностью. Например, котировка акций на момент голосования практически совпадает с количеством голосов, поданных за того или иного кандидата. Точность значительно превышает точность опросов. Ответ на вопрос в опросе мнений – действие безответственное, в то время как при игре на бирже человек рискует своими деньгами. Даже игрушечными – в США нельзя использовать настоящие деньги при игре на бирже предсказаний, за некоторыми исключениями.

Итак, теория вероятностей является априорным знанием. Все люди так или иначе используют вероятность бессознательно. Действительно, представим себе, что Вам предложили пройти по широкой (50см) доске, лежащей на полу. Нет проблем. А если доска находится на высоте, скажем, 3 метров. Страшновато, но пройти можно. А если доску поднять на высоту 100 метров. Очень страшно. Почему? Дело в том, что мы подсознательно оцениваем риски, а точнее – математическое ожидание потерь. На низкой доске потери минимальны, а на высокой – очень большие. Хотя вероятность падения и мала (доска широкая), но потери несовместимы с жизнью. Поэтому математическое ожидание потерь очень большое, и поэтому страшно. Теорию вероятностей знают и животные. Посмотрите, как страшно неопытной собаке пройти по буму. Про опыт собак и людей поговорим несколько позже.

1.2 Некоторые обозначения

Прежде, чем идти дальше, введем некоторые математические понятия и определения.

Гамма-функция. По определению,

$$\Gamma(x) = \int_0^{+\infty} t^{x-1} e^{-t} dt$$

Эта функция обладает рядом интересных свойств, а именно:

$$\Gamma(0) = 1; \quad \Gamma(1) = 1; \quad \Gamma(x+1) = x\Gamma(x) \quad (1.1)$$

Следствие:

$$\Gamma(n+1) = n!$$

Формула Стирлинга

$$\Gamma(x+1) \simeq \sqrt{2\pi x} x^x e^{-x}, \quad x \rightarrow +\infty$$

Иными словами Гамма-функция является обобщением факториала на не целые значения.

Дельта-функция. По определению,

$$\delta(x) = 0, \quad x \neq 0$$

$$\int_{-\infty}^{+\infty} \delta(x) dx = 1$$

чтобы представить себе эту функцию, рассмотрим плотность нормального распределения с математическим ожиданием 0:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

и устремим $\sigma \rightarrow 0$ (рис.1.2). Эта функция является так называемой обоб-

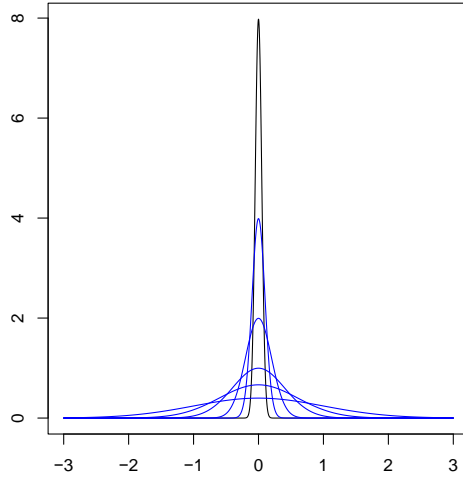


Рис. 1.2: Дельта-функция, как предел (в некотором смысле) плотностей нормального распределения

щенной функцией. Она обладает некоторыми свойствами, например, для любой непрерывной функции f :

$$\int_{-\infty}^{+\infty} \delta(x-a)f(x)dx = \int_{-t-a}^{+t-a} \delta(x-a)f(x)dx = f(a) \quad (1.2)$$

где t — любое положительное число. Действительно, дельта-функция везде равна 0, кроме нуля. Поэтому вклад в интеграл будет только от значения функции $f(x)$ при $x = a$.

Символ Кронекера. Это понятие определено для дискретных величин (например, букв). По определению:

$$\delta(m, n) = \delta_{m,n} = \begin{cases} 1, & m = n \\ 0, & m \neq n \end{cases}$$

Свойство: Если есть строка $\{s_i\}$, то

$$\sum_{i=0}^L \delta(s_i, \alpha) = n_\alpha$$

где α – некоторый символ, n_α – количество встреч символа α

1.3 Теорема Байеса и оценивание параметров

Представим себе, что при трех бросках монеты трижды выпал «орел». Какова вероятность выпадения орла в *следующем* броске. Если подходить с точки зрения нормальной статистики, вероятность выпадения орла в следующем броске равна отношению числа успехов к числу испытаний $p(o) = n_o/N = 3/3 = 1$. Это явно противоречит здравому смыслу. Мы сделали очень мало испытаний. Весь наш предыдущий опыт говорит, что здесь мы наблюдаем лишь флуктуацию, а монета скорее всего правильная, но некоторое сомнение все-таки есть.

Итак, из общих соображений мы можем предположить, что есть некоторое *распределение* вероятностей для вероятности орла. Т.е. есть большой мир монет и наша монета пришла из этого большого мира. И в этом мире монет есть распределение вероятностей для кривизны монет (рис.1.3) $p(p_o)$. Каждый представляет себе мир монет по-своему. У нас есть пара-

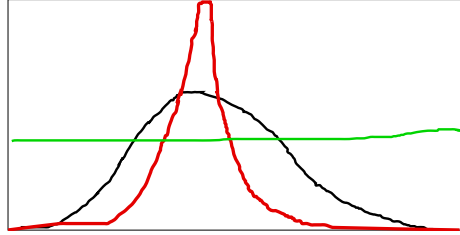


Рис. 1.3: Распределения свойств монет.

метр $\theta = p_o$, который мы хотим оценить. Для этого у нас есть: результат опыта D и представление о свойствах монет вообще $p(\theta)$. Мы хотим что-то узнать про *нашу* монету. Для этого нам подходит теорема Байеса.

$$\Pr(\theta | D) = \frac{\Pr(D | \theta) \cdot \Pr(\theta)}{\Pr(D)} \quad (1.3)$$

здесь $\Pr(D|\theta)$ – это вероятность того, что мы видим. Это не совсем вероятность, поскольку событие D (в нашем случае – вероятность выпадения трех орлов) уже произошло. Эту величину называют *правдоподобием* – это вероятность события D , если оно как бы еще не произошло. Для нашего случая $\Pr(D|\theta) = \theta^3$. Распределение $\Pr(\theta)$ – наше представление о мире монет, считаем его известным. Это распределение называется *априорным* распределением. Величина $\Pr(\theta|D)$ называется *апостериорным* распределением. Это распределение вероятностей для параметра (в нашем случае вероятности выпадения орла p_o) для *нашей* монеты, полученное в результате опыта. Остался неизвестным знаменатель $\Pr(D)$. Для его оценки заметим, что $(\theta|D)$ – полное пространство событий, поэтому

$$\begin{aligned} \int_0^1 \Pr(\theta|D) d\theta &= 1 = \\ \int_0^1 \frac{\Pr(D|\theta) \cdot \Pr(\theta)}{\Pr(D)} d\theta &= \\ \frac{\int_0^1 \Pr(D|\theta) \cdot \Pr(\theta) d\theta}{\Pr(D)} \end{aligned}$$

Отсюда сразу получаем:

$$\Pr(D) = \int_0^1 \Pr(D|\theta) \cdot \Pr(\theta) d\theta$$

Окончательно получаем:

$$\Pr(\theta|D) = \frac{\Pr(D|\theta) \cdot \Pr(\theta)}{\int_0^1 \Pr(D|\theta) \cdot \Pr(\theta) d\theta} \quad (1.4)$$

1.3.1 Примеры вычислений

Рассмотрим простую задачу. Есть редкая болезнь – число заболевших 1 на миллион. Болезнь в запущенной форме очень тяжелая, а лечение имеет побочные осложнения. Некоторая компания разработала тест-систему, которая для носителя дает абсолютную точность, но на здоровых ошибается в одном случае из тысячи. Казалось бы тест хороший. Стоит ли его проходить? Мы хотим оценить вероятность того, что при положительном тесте человек здоров. Итак у нас есть:

$\Pr D = 10^{-6}$ – вероятность того, что человек болен

$\Pr H = 1 - 10^{-6}$ – вероятность того, что человек здоров

$\Pr(+|D) = 1$ – вероятность положительного теста при условии, что человек болен

$\Pr(+|H) = 10^{-3}$ – вероятность положительного теста при условии, что человек здоров

Вероятность того, что человек здоров при условии положительного теста:

$$\begin{aligned}\Pr(H|+) &= \frac{\Pr(+|H) \cdot \Pr(H)}{\Pr(+|H) \cdot \Pr(H) + \Pr(+|D) \cdot \Pr(D)} \\ &= \frac{10^{-3} \cdot (1 - 10^{-6})}{10^{-3} \cdot (1 - 10^{-6}) + 1 \cdot 10^{-6}} \simeq 1 - 10^{-3}\end{aligned}$$

Очевидно, что тест не эффективен.

1.3.2 Примеры априорных распределений

Вернемся к нашему примеру – выпадение трех орлов в трех испытаниях.

Равномерное априорное распределение. Мы вообще не представляем как устроены монеты, которые нам подсовывают. Тогда полагаем, что априорное распределение вероятностей вероятности орла распределено равномерно: $p(\theta) = 1$. Подставляем в формулу (1.4) и получаем:

$$\Pr(\theta | D) = \frac{\theta^3 \cdot 1}{\int_0^1 \theta^3 \cdot 1 d\theta} = \frac{\theta^3}{1/4} = 4\theta^3$$

Априорное распределение – дельта-функция. Пусть, теперь мы уверены, что монета прямая. Это значит, что априорное распределение – дельта-функция, сосредоточенная в позиции 0.5: $\delta(x - 0.5)$. Тогда уравнение Байеса предстанет в виде:

$$\begin{aligned}\Pr(\theta | D) &= \frac{\Pr(D | \theta) \cdot \Pr(\theta)}{\int_0^1 \Pr(D | \theta) \cdot \Pr(\theta) d\theta} = \\ &= \frac{\theta^3 \cdot \delta(\theta - 0.5)}{\int_0^1 \theta^3 \cdot \delta(\theta - 0.5) d\theta} =\end{aligned}$$

Поскольку дельта-функция равна 0 везде, кроме точки 0.5, то применив свойство (1.2), получаем:

$$\Pr(\theta | D) = \frac{0.5^3 \cdot \delta(\theta - 0.5)}{0.5^3} = \delta(\theta - 0.5) \quad (1.5)$$

Т.е. если мы абсолютно уверены, что все монеты прямые, то по результату опыта мы еще раз убедились, что наша монета тоже прямая, несмотря на то, что три раза выпал орел.

Априорное распределение – более общий случай. Наверное, распределение вероятностей вероятности выпадения орла устроено примерно как красная или черная линия на рис.1.3. Какое это может быть распределение. Первое, что приходит в голову – это нормальное распределение

с математическим ожиданием 0.5. Но оно не годится, поскольку носитель нормального распределения – все числовая ось $(-\infty, +\infty)$, а наше распределение сосредоточено на отрезке $[0, 1]$. Подходящим распределением является бета-распределение:

$$f(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} \quad (1.6)$$

где $B(\alpha, \beta)$ – бета-функция (нормировочный множитель).

$$B(\alpha, \beta) = \int_0^1 x^{\alpha-1} (1-x)^{\beta-1} dx = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \quad (1.7)$$

На рис.1.4 показано плотность бета - распределения для разных значений

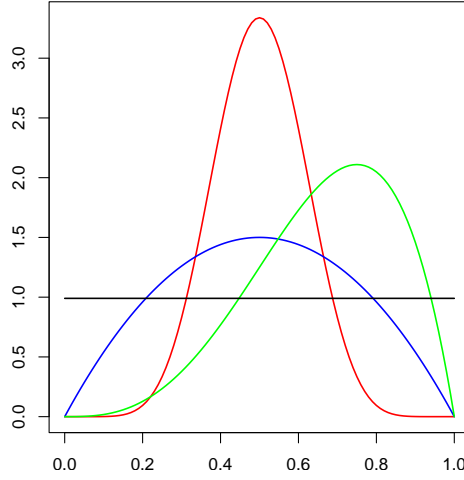


Рис. 1.4: Бета-распределение при разных параметрах. $(\alpha = 1, \beta = 1)$ – черный; $(\alpha = 2, \beta = 2)$ – синий; $(\alpha = 4, \beta = 2)$ – зеленый; $(\alpha = 10, \beta = 10)$ – красный

параметров. При $\alpha \rightarrow \infty, \beta \rightarrow \infty, \alpha = \beta$ это распределение будет (в некотором смысле) стремиться к дельта-функции $\delta(x - 0.5)$. Таким образом, этот класс распределений вполне подходит в качестве априорного распределения. Если априорное распределение – бета-распределение, то формула Байеса для нашего случая (три орла в трех испытаниях) примет вид:

$$\begin{aligned} \Pr(\theta | D) &= \frac{\Pr(D | \theta) \cdot \Pr(\theta)}{\int_0^1 \Pr(D | \theta) \cdot \Pr(\theta) d\theta} \\ &= \frac{\theta^3 \cdot \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1}}{\int_0^1 \theta^3 \cdot \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1} d\theta} = \\ &= \frac{\theta^{\alpha+2} (1-\theta)^{\beta-1}}{\int_0^1 \theta^{\alpha+2} (1-\theta)^{\beta-1} d\theta} \end{aligned} \quad (1.8)$$

На рис.1.5 показано, как меняется распределение вероятностей после испытаний при разных параметрах априорного распределения. Видно, что наибольшему изменению подверглось равномерное распределение, в то время как распределение с большими значениями параметров α , β изменились слабо. С накоплением данных происходит изменение распределения веро-

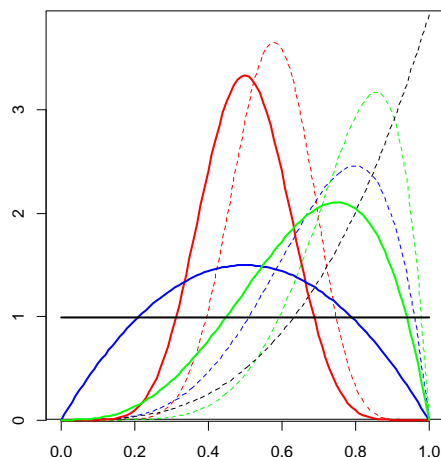


Рис. 1.5: Априорные (сплошные линии) и апостериорные (пунктирные линии) распределения при разных параметрах. ($\alpha = 1, \beta = 1$) — черный; ($\alpha = 2, \beta = 2$) — синий; ($\alpha = 4, \beta = 2$) — зеленый; ($\alpha = 10, \beta = 10$) — красный

яностей для вероятности успеха — чем больше успехов было, тем более вероятным представляется успех в будущем. При проекции на поведение людей и животных это можно интерпретировать, как накопление опыта. Хорошо дрессированная собака, которая сотни раз проходила по буму уже не боится — произошла переоценка вероятности неудачи. Статистика показывает, что самыми аварийными водителями являются водители со стажем около трех лет. У них три года все было в порядке (тем более, что в начале они ездили очень осторожно) и они перестали бояться. Потом накапливается настоящий опыт — уже они видели много аварий и сами попадали в неприятные ситуации (я не говорю уже о Дарвиновском отборе).

1.4 Оценки параметров.

Мы научились получать апостериорные распределения по результатам опытов. Однако, вместо распределений (функций) гораздо удобнее иметь дело с числами. Для этого можно из распределений получать разные числа, например математическое ожидание или моду. Рассмотрим по-прежнему ис-

пытания монеты. Пусть у нас было N испытаний и выпало n орлов и m решек. Тогда апостериорная вероятность будет

$$\begin{aligned} \Pr(\theta | D) &= \frac{\Pr(D | \theta) \cdot \Pr(\theta)}{\int_0^1 \Pr(D | \theta) \cdot \Pr(\theta) d\theta} \\ &= \frac{C_{m+n}^m \theta^n (1 - \theta)^m \cdot \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1 - \theta)^{\beta-1}}{\int_0^1 C_{m+n}^m \theta^n (1 - \theta)^m \cdot \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1 - \theta)^{\beta-1} d\theta} \quad (1.9) \\ &= \frac{\theta^{n+\alpha-1} (1 - \theta)^{m+\beta-1}}{\int_0^1 \theta^{n+\alpha-1} (1 - \theta)^{m+\beta-1} d\theta} \end{aligned}$$

Можно сделать несколько численных оценок значения параметра θ , который, как мы помним, есть оценка вероятности выпадения орла в следующем испытании.

1.4.1 L-оценка

Первая оценка – это традиционная оценка максимума правдоподобия $\theta = \arg \max \Pr(D | \theta)$ (*L-оценка*). Для определения этой оценки нам надо найти значение θ , которое максимизирует правдоподобие. Для этого вычислим производную от правдоподобия:

$$\begin{aligned} \frac{\partial \Pr(D | \theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} C_{n+m}^n \theta^n (1 - \theta)^m \\ &= C_{n+m}^n (n \theta^{n-1} (1 - \theta)^m - m \theta^n (1 - \theta)^{m-1}) \\ &= C_{n+m}^n \theta^{n-1} (1 - \theta)^{m-1} (n(1 - \theta) - m\theta) = 0 \end{aligned}$$

Решая это уравнение, получаем:

$$\begin{aligned} n(1 - \theta) - m\theta &= n - \theta(m + n) = 0 \\ \theta &= \frac{n}{n + m} \end{aligned} \quad (1.10)$$

Мы получили стандартную оценку вероятности – число успехов делить на число испытаний. При этом мы не принимали во внимание априорное распределение.

1.4.2 MAP-оценка

Теперь посмотрим на оценку моды (максимума распределения) апостериорного распределения вероятностей выпадения орла (1.9), и найдем максимум этой функции (*MAP-оценка*).

$$\begin{aligned} \frac{\partial \Pr(\theta | D)}{\partial \theta} &= \frac{1}{\int_0^1 \theta^{n+\alpha-1} (1 - \theta)^{m+\beta-1} d\theta} \frac{\partial}{\partial \theta} (\theta^{n+\alpha-1} (1 - \theta)^{m+\beta-1}) = 0 \\ \theta^{n+\alpha-1} (1 - \theta)^{n+\beta-1} ((n + \alpha - 1)(1 - \theta) - (m + \beta - 1)\theta) &= 0 \end{aligned}$$

Откуда сразу получаем:

$$\theta = \frac{n + \alpha - 1}{n + m + \alpha + \beta - 2} \quad (1.11)$$

Сравним уравнения (1.10) и (1.11). Они очень похожи, единственное, в числитель и в знаменатель добавились некоторые значения: $\alpha - 1$ и $(\alpha - 1) + (\beta - 1)$. Это как если мы добавили к нашим наблюдениям $\alpha - 1$ успехов и $\beta - 1$ неудач. Эти добавки (мы их увидим еще много раз) называются псевдо-счетчиками (pseudocounts).

1.4.3 Е-оценка

Еще одной оценкой параметра является оценка математического ожидания апостериорного распределения, называемая *Е-оценкой*:

$$E(\theta) = \int_0^1 \theta \cdot \Pr(\theta | D) d\theta = \frac{\int_0^1 \theta \Pr(D | \theta) \Pr(\theta) d\theta}{\int_0^1 \Pr(D | \theta) \Pr(\theta) d\theta}$$

Для случая бернуллиевских испытаний (монета) и для бета-распределения в качестве априорного распределения получаем:

$$E(\theta) = \frac{\int_0^1 \theta^{n+1} (1 - \theta)^m \cdot \theta^{\alpha-1} (1 - \theta)^{\beta-1} d\theta}{\int_0^1 \theta^n (1 - \theta)^m \cdot \theta^{\alpha-1} (1 - \theta)^{\beta-1} d\theta} = \frac{\int_0^1 \theta^{n+\alpha} (1 - \theta)^{m+\beta-1} d\theta}{\int_0^1 \theta^{n+\alpha-1} (1 - \theta)^{m+\beta-1} d\theta}$$

Интегралы в числителе и в знаменателе есть бета-функции (1.7):

$$\begin{aligned} E(\theta) &= \frac{B(n + \alpha + 1, m + \beta)}{B(n + \alpha, m + \beta)} \\ &= \frac{\Gamma(n + \alpha + 1) \Gamma(m + \beta)}{\Gamma(n + \alpha + 1 + m + \beta)} \frac{\Gamma(n + \alpha + m + \beta)}{\Gamma(n + \alpha) \Gamma(m + \beta)} \end{aligned}$$

Принимая во внимание свойство гамма-функции (1.1), получим:

$$E(\theta) = \frac{n + \alpha}{n + m + \alpha + \beta} \quad (1.12)$$

Мы видим почти такую же оценку, что и для случая МАР-оценки (1.11), только псевдо-счетчики на 1 больше.

1.5 Распределение Дирихле

Вторым по значимости объектом теории вероятностей является игральная кость. Здесь при испытании возможны 6 исходов. А если мы анализируем нуклеотидные последовательности, то возможны 4 исхода. Эти случаи отличаются от монеты, поскольку мы оцениваем не одну вероятность, а

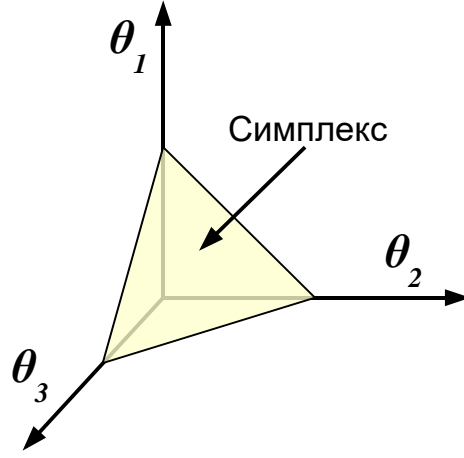


Рис. 1.6: Симплекс в трехмерном пространстве







сразу несколько. Поэтому величина, которая оценивается по итогам испытаний – это целый вектор θ . При этом есть естественное ограничение $\sum \theta_i = 1$; $\theta_i \geq 0$. Эти ограничения говорят о том, что носителем распределения для θ является симплекс (рис.1.6) В качестве априорного распределения для набора вероятностей используют обобщение бета-распределения – распределение Дирихле:

$$f(\theta) = \frac{1}{B(\alpha)} \prod \theta_i^{\alpha_i - 1}$$

$$B(\alpha) = \int_{simplex} \prod \theta_i^{\alpha_i - 1} d\theta = \frac{\prod \Gamma(\alpha_i)}{\Gamma(\sum \alpha_i)} \quad (1.13)$$

При целых значениях параметров α_i гамма-функция превращается в факториал, и распределение $f(\theta)$ замечательным образом превращается в полиномиальное с числами наблюдений $n_i = \alpha_i - 1$:

$$f(\theta) = \frac{(\sum (\alpha_i - 1))!}{\prod (\alpha_i - 1)!} \prod \theta_i^{\alpha_i - 1} = \frac{(\sum (n_i))!}{\prod n_i!} \prod \theta_i^{n_i}$$

Пусть мы сделали испытание и получили n_i исходов каждого типа. Например число выпавших       при бросании игральной кости. Тогда апостериорное распределение параметров θ_i (вероятности выпадения гра-

ней игральной кости) будет

$$\begin{aligned} \Pr(\theta | D) &= \frac{\Pr(D | \theta) \cdot \Pr(\theta)}{\int_{\text{simplex}} \Pr(D | \theta) \cdot \Pr(\theta) d\theta} \\ &= \frac{\frac{(\sum n_i)!}{\prod n_i!} \prod \theta_i^{n_i} \cdot \frac{1}{B(\alpha)} \prod \theta_i^{\alpha_i-1}}{\int_{\text{simplex}} \frac{(\sum n_i)!}{\prod n_i!} \prod \theta_i^{n_i} \cdot \frac{1}{B(\alpha)} \prod \theta_i^{\alpha_i-1} d\theta} = \frac{\prod \theta_i^{n_i+\alpha_i-1}}{\int_{\text{simplex}} \prod \theta_i^{n_i+\alpha_i-1} d\theta} \end{aligned} \quad (1.14)$$

здесь $\frac{(\sum n_i)!}{\prod n_i!} \prod \theta_i^{n_i}$ – полиномиальное распределение.

1.5.1 Оценки параметров

Л-оценка Правдоподобие в (1.14) – это

$$L = \frac{(\sum n_i)!}{\prod n_i!} \prod \theta_i^{n_i}$$

максимизируем L по θ при ограничении $\sum \theta_i = 1$. Для этого применим метод неопределенных множителей Лагранжа. Строим функционал

$$\Phi = L - \lambda \left(\sum \theta_i - 1 \right) = A \prod \theta_i^{n_i} - \lambda \left(\sum \theta_i - 1 \right)$$

где $A = (\sum n_i)! / \prod n_i!$. Дифференцируем Φ по одной из переменных θ_k . При этом используем выражение для производной степенной функции $(x^n)' = nx^{n-1} = nx^n/x$. Получаем уравнения для поиска экстремума:

$$\frac{\partial \Phi}{\partial \theta_k} = \frac{n_k}{\theta_k} A \prod \theta_i - \lambda = \frac{n_k}{\theta_k} L - \lambda = 0$$

Отсюда получаем

$$\theta_k = \frac{n_k L}{\lambda}$$

чтобы найти λ вспоминаем ограничение $\sum \theta_i = 1$. Суммируя эти уравнения, получаем

$$\sum n_k \frac{L}{\lambda} = 1; \quad \lambda = L \sum n_k$$

Окончательно получаем Л-оценку

$$\theta_k = \frac{n_k}{\sum n_i} \quad (1.15)$$

Во-первых, эта оценка достаточно очевидна – все то же число успехов делить на число испытаний. Во-вторых, она обобщает Л-оценку для случая двух исходов.

МАР-оценка. Апостериорная вероятность для случая полиномиального распределения есть (1.14):

$$\Pr(\theta | D) = \frac{\prod \theta^{n_i + \alpha_i - 1}}{\int_{\text{simplex}} \prod \theta^{n_i + \alpha_i - 1} d\theta}$$

Для максимизации этой величины при ограничениях на $\theta \sum \theta_i = 1$ те же приемы, что и при максимизации правдоподобия. В результате получим

$$\theta_k = \frac{n_k + \alpha_k - 1}{\sum_i (n_i + \alpha_i - 1)} \quad (1.16)$$

Это уравнение также обобщает МАР оценку для случая двух исходов и здесь также появляются псевдо-счетчики, которые являются параметрами распределения Дирихле. Обратим внимание, что при $n_i \rightarrow \infty$ вклад псевдо-счетчиков исчезает и апостериорная оценка стремится к оценке максимума правдоподобия.

Е-оценка. Оценим математическое ожидание параметров (вероятностей исходов) апостериорного распределения.

$$E(\theta_k) = \frac{\int_{\text{simplex}} \theta_k \prod \theta^{n_i + \alpha_i - 1} d\theta}{\int_{\text{simplex}} \prod \theta^{n_i + \alpha_i - 1} d\theta}$$

Воспользуемся формулой (1.13) для вычисления интеграла по симплексу. Тогда, аналогично случаю для двух исходов, получаем:

$$\begin{aligned} E(\theta_k) &= \frac{\prod_{i \neq k} \Gamma(n_i + \alpha_i) \cdot \Gamma(n_k + \alpha_k + 1)}{\Gamma(1 + \sum n_i + \alpha_i)} \cdot \frac{\Gamma(\sum (n_i + \alpha_i))}{\prod \Gamma(n_i + \alpha_i - 1)} \\ &= \frac{\prod_{i \neq k} \Gamma(n_i + \alpha_i) \cdot \Gamma(n_k + \alpha_k) \cdot (n_k + \alpha_k)}{\Gamma(\sum n_i + \alpha_i) \cdot (\sum n_i + \alpha_i)} \cdot \frac{\Gamma(\sum (n_i + \alpha_i))}{\prod \Gamma(n_i + \alpha_i)} \\ &= \frac{n_k + \alpha_k}{\sum (n_i + \alpha_i)} \end{aligned}$$

Здесь мы получили оценку, аналогичную оценке для случая двух исходов. Надо просто к наблюдаемым счетчикам добавить псевдо-счетчики. И для Е-оценки они на 1 больше, чем для МАР-оценки.

1.6 Другие распределения

Весь предыдущий анализ касался оценки вероятностей исходов. Вероятности исходов являются параметрами биномиального или полиномиального распределения. Но кроме биномиального распределения мы знаем еще множество других распределений, у которых тоже есть свои параметры, которые могут иметь другой смысл.

1.6.1 Распределение Пуассона

Пусть мы имеем некоторое количество наблюдений пуассоновского процесса. Например, это может быть количества прочтений, покрывающих данный ген в нескольких репликах, или в нескольких клетках при одноклеточном секвенировании транскриптома. Даже если уровень экспрессии гена постоянен, по случайным причинам мы будем иметь в разных *одинаковых* экспериментах разный уровень покрытия. В простейшем случае можно считать, что число прочтений, покрывающих данный ген, подчиняется распределению Пуассона:

$$\Pr(D) = \frac{\lambda^n}{n!} e^{-\lambda} \quad (1.17)$$

Мы хотим определить уровень экспрессии этого гена λ . Напишем формулу для Байесовой оценки параметра:

$$\Pr(\lambda | D) = \frac{\Pr(D | \lambda) \cdot \Pr(\lambda)}{\Pr(D)}$$

Посмотрим, что означают вероятности, входящие в это уравнение. Во-первых, правдоподобие $\Pr(D | \lambda)$ — это произведение вероятностей наблюдений n_i при этом было сделано M экспериментов и суммарное количество прочтений равно $\sum n_i = N$:

$$\Pr(D | \lambda) = \prod \frac{\lambda^{n_i}}{n_i!} e^{-\lambda \cdot M} = \frac{\lambda^N}{\prod n_i!} e^{-\lambda \cdot M}$$

Теперь зададим априорное распределение. Уровень экспрессии гена в пространстве всех генов наверное устроен как-то так (рис.1.7А). Для этого хорошо подходит двух-параметрическое гамма-распределение:

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

Параметры этого распределения можно оценить из анализа уровней экспрессии разных генов в разных экспериментах. Теперь можно написать формулу Байеса для этого случая:

$$\begin{aligned} \Pr(\lambda | D) &= \frac{\Pr(D | \lambda) \cdot \Pr(\lambda)}{\Pr(D)} \\ &= \frac{\frac{\lambda^N}{\prod n_i!} e^{-\lambda \cdot M} \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta \lambda}}{\int_0^\infty \frac{\lambda^N}{\prod n_i!} e^{-\lambda \cdot M} \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta \lambda} d\lambda} \\ &= \frac{\lambda^N e^{-\lambda \cdot M} \lambda^{\alpha-1} e^{-\beta \lambda}}{\int_0^\infty \lambda^N e^{-\lambda \cdot M} \lambda^{\alpha-1} e^{-\beta \lambda} d\lambda} \\ &= \frac{\lambda^{N+\alpha-1} e^{-(M+\beta)\lambda}}{\int_0^\infty \lambda^{N+\alpha-1} e^{-(M+\beta)\lambda} d\lambda} \end{aligned}$$

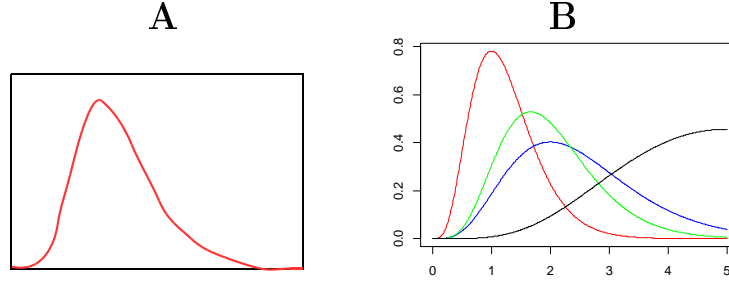


Рис. 1.7: А – предполагаемое распределение уровней экспрессии генов. В – Гамма распределение при разных параметрах

т.е. апостериорное распределение опять стало гамма-распределением с новыми параметрами $\alpha' = N + \alpha$, $\beta' = M + \beta$. Для гамма-распределения известны мода и математическое ожидание (см., например, википедию). Поэтому можно сразу сделать MAP- и Е-оценки:

$$\lambda_{MAP} = \frac{\alpha' - 1}{\beta'} = \frac{N + \alpha - 1}{M + \beta}$$

$$\lambda_E = \frac{\alpha'}{\beta'} = \frac{N + \alpha}{M + \beta}$$

Любопытно, что новое распределение не зависит от того, как распределились отдельные наблюдения, а важны лишь суммарное количество прочтений и количество опытов. На рис.1.8А показаны априорные и апостериорные распределения для наблюдений $n_i = \{3, 6, 0\}$ и разных параметрах априорного распределения. Однако, если увеличить количество экспериментов, то апостериорные распределения станут более узкими (рис.1.8В) и меньше будут зависеть от параметров априорного распределения. Заметим, что дисперсия распределения λ равна:

$$D(\lambda) = \frac{\alpha' - 1}{\beta'^2}; \quad \sigma(\lambda) = \frac{\sqrt{\alpha' - 1}}{\beta'} = \frac{\sqrt{N + \alpha}}{M + \beta}$$

Отсюда следует важный практический вывод. Если Вы хотите точнее определить экспрессию гена, то делайте много реплик, пусть даже в каждой реплике будет меньшее покрытие.

Формулы апостериорного распределения для полиномиального распределения с распределением Дирихле в качестве априорного и для распределения Пуассона при априорном гамма-распределении удивительным образом свернулись к простым выражениям. Это не случайно. Если апостериорное распределение $\Pr(\theta|D)$ принадлежит тому же семейству вероятностных распределений, что и априорное распределение $\Pr(\theta)$ (т.е. имеет тот же вид, но с другими параметрами), то это семейство распределений называется *сопряжённым* семейству функций правдоподобия $\Pr(D|\theta)$.

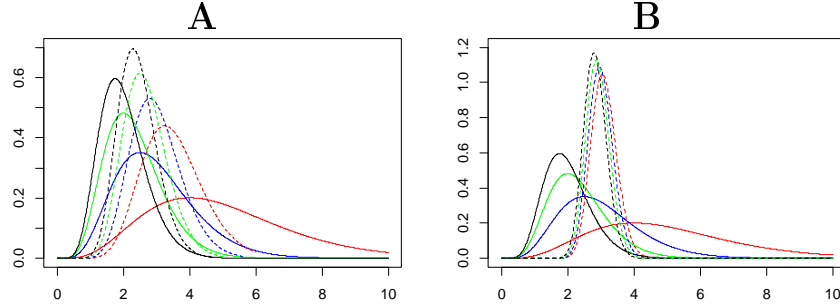


Рис. 1.8: А – Априорные (сплошные линии) и апостериорные (пунктирные линии) гамма-распределения для наблюдений $n_i = \{3, 6, 0\}$. Параметры априорных гамма-распределений: (5,1) – красная, (6,2) – синяя, (7,3) – зеленая, (8,4) – черная. В – то же, но если $M = 20$, $N = 60$

1.6.2 Смеси Дирихле

Представим себе, что есть две фабрики монет, которые делают монеты с разной кривизной. Пусть распределения кривизны монет, изготовленных на разных фабриках известны ($f_r(\theta)$), и известна доля каждой фабрики в общей выборке q_r . Тогда априорное распределение примет вид:

$$\Pr(\theta) = \sum_r q_r f_r(\theta); \quad \sum_r q_r = 1$$

Такое смешанное распределение называется смесью Дирихле. В биоинформатике такие смеси встречаются достаточно часто. Например, если есть колонка в множественном выравнивании аминокислотных последовательностей. Эта колонка может принадлежать трансмембранному сегменту или гидрофобному ядру или поверхности белка. Для каждого из типов белковой глобулы есть свои априорные распределения типа распределений Дирихле $f_r(\theta) = \frac{1}{Z} \prod (\theta_i)^{\alpha_i^r} - 1$. В этом случае априорное распределение имеет вид:

$$\Pr(\theta) = \sum_r \frac{1}{Z_r} q_r (\theta_i)^{\alpha_i^r - 1} = \sum_r \frac{\Gamma(\sum_i \alpha_i^r)}{q} \prod_i \Gamma(\alpha_i^r) (\theta_i)^{\alpha_i^r - 1}$$

Апостериорное распределение имеет вид:

$$\Pr(\theta|D) = \frac{\prod \theta_i^{n_i} \sum_r q_r \frac{\Gamma(\sum_i \alpha_i^r)}{\prod_i \Gamma(\alpha_i^r)} (\theta_i)^{\alpha_i^r - 1}}{\int_{simplex} \prod \theta_i^{n_i} \sum_r q_r \frac{\Gamma(\sum_i \alpha_i^r)}{\prod_i \Gamma(\alpha_i^r)} (\theta_i)^{\alpha_i^r - 1} d\theta} \quad (1.18)$$

Для случая анализа колонки множественного выравнивания задача выглядит так. Глядя на колонку выравнивания понять какие аминокислотные

остатки ожидаются в данной позиции и с какой вероятностью. При этом мы не знаем принадлежит данная позиция поверхности белка, трансмембранному сегменту или гидрофобному ядру белка. Из анализа большого количества разных белков мы знаем долю поверхностных остатков, остатков, принадлежащих ядру и пр. (q_r). Мы также знаем распределения частот аминокислотных остатков в соответствующих типах позиций (α_i^r). Уравнение (1.18) дает нам распределение вероятностей встречи аминокислотных остатков в *данной* позиции.

Можно также поставить задачу об определении вероятности принадлежности к тому или иному классу q_r при условии наблюдения:

$$\Pr(q_r|D) = \frac{\Pr(D|q_r) \Pr(q_r)}{\Pr(D)}$$

Смеси Дирихле достаточно часто встречаются в биоинформатике в целом ряде задач, не только связанных с оценкой вероятностей встречи тех или иных аминокислот, но также встречаются в анализе экспрессии генов, задачах молекулярной диагностики, задачах кластеризации и прочих.

1.7 Максимизация ожидания (ЕМ)

Рассмотрим задачу. Пусть у нас есть серия наблюдений $X = \{x_i\}$ случайной величины ξ . Причем эта случайная величина является смесью Дирихле двух случайных величин, про которые мы только предполагаем, что они нормально распределенные с параметрами (μ_1, σ_1^2) и (μ_2, σ_2^2) , причем мы не знаем эти параметры. Мы предполагаем, что часть наблюдений пришла из первого распределения, а часть – из второго. Вопрос: какие наблюдения из какого распределения пришли и каковы параметры этих распределений?

Итак, у нас есть наблюдения и гипотеза о том, что наблюдения пришли из смеси распределений и мы хотим определить какие значения из какого распределения пришли. Казалось бы задача не разрешимая. Однако, если нам кто-то скажет каковы параметры распределений, то мы можем разложить наше множество наблюдений X на два подмножества $X = X' \cup X''$. Для каждого разбиения можно написать логарифм правдоподобия:

$$\begin{aligned} L(X', X'' | \mu_r, \sigma_r) &= \ln \prod e^{-\frac{(x'_i - \mu_1)^2}{2\sigma_1^2}} + \ln \prod e^{-\frac{(x''_i - \mu_2)^2}{2\sigma_2^2}} \\ &= -\sum \frac{(x'_i - \mu_1)^2}{2\sigma_1^2} - \sum \frac{(x''_i - \mu_2)^2}{2\sigma_2^2} \rightarrow \max_{X', X''} \end{aligned} \quad (1.19)$$

Далее, можно максимизировать эту величину по всем разбиениям. Для этого упорядочим все x_i . Ясно, что разбиение будет устроено так – часть наблюдений от 1 до k будет принадлежать $X' = \{x_1, \dots, x_k\}$, а остальные – $X'' = \{x_{k+1}, \dots, x_n\}$. Можно просто перебрав все возможные границы определить оптимальное разбиение. С другой стороны, если нам известны разбиения, то мы можем оценить параметры распределений (μ_1, σ_1^2) и

(μ_2, σ_2^2) :

$$\begin{aligned} (\mu_1|X', X'') &= \frac{\sum_{i=1}^k x_i}{k}; & (\sigma_1^2|X', X'') &= \frac{\sum_{i=1}^k (x_i - \mu_1)^2}{k-1} \\ (\mu_2|X', X'') &= \frac{\sum_{i=k+1}^n x_i}{n-k}; & (\sigma_2^2|X', X'') &= \frac{\sum_{i=k+1}^n (x_i - \mu_2)^2}{n-k-1} \end{aligned} \quad (1.20)$$

Теперь можно сформулировать алгоритм. Он состоит из итераций, на каждой из которых выполняется два действия – вычисляются параметры распределений (1.20, шаг Expectation). Затем оптимизируется разбиение (1.19, шаг Maximization). Итерации прекращаются, когда параметры распределений перестали меняться, или по достижении заданного количества итераций. В качестве начального приближения используется случайное разбиение. Идеи такого подхода лежат в основе популярного алгоритма кластеризации k-средних (k-means).

В общем случае алгоритм максимизации ожидания применяется для поиска параметров, описывающих наблюдения. На шаге максимизации происходит максимизация правдоподобия, на шаге ожидания подбираются параметры. Позже мы несколько раз встретим этот метод.

1.8 Заключительные замечания

Как уже упоминалось, Байесову вероятность понимают даже животные. Например, собака с большой осторожностью впервые идет по буму, поскольку она имеет априорное представление об опасности. Однако после того, как она несколько раз пройдет – происходит переоценка вероятностей и апостериорная вероятность становится априорной для дальнейшей оценки исходов при последующих испытаниях. Именно поэтому после каждого успеха априорное представление об опасности становится более оптимистичным, а заодно и навыки выполнения упражнения тоже совершенствуются. Одно время в YouTube была серия роликов про собачку Ozzy, которая с удовольствием и весьма лихо умела ходить по канату. Часто такая переоценка своих возможностей приводит к печальным последствиям. Известно, что чаще всего в ДТП попадают водители со стажем 3-5 лет, поскольку априорная оценка опасности сильно смещена в сторону безопасности. То же можно сказать про парашютистов и альпинистов. Самый опасный уровень – средний, когда оценка опасности занижена, а навыков ещё маловато.

Глава 2

Модели последовательностей

2.1 Модели природных последовательностей

В анализе данных важную роль играют модели данных, которые позволяют решать многие задачи. Правильная модель данных должна во-первых, хорошо отражать данные, а с другой стороны иметь вероятностную природу. Еще, хорошо бы, чтобы модель сопровождалась математическим аппаратом.

2.1.1 Влияние модели на оценку

Рассмотрим геометрический пример. Какова вероятность того, что произвольная хорда в круге длиннее, чем сторона вписанного правильного треугольника (короче, чем $\sqrt{3}R$) (рис.2.1)?

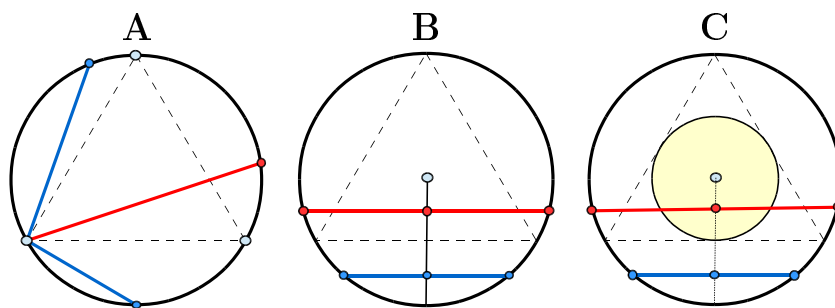


Рис. 2.1: Длина хорды. Красная — длинная хорда, синяя — короткая хорда

Оценка №1. Выберем две случайные точки на окружности и проведем через них хорду. Совмещаем с одним концом хорды вершину вписанного правильного треугольника (рис.2.1А). Если второй конец пересечет противоположную сторону треугольника, то это будет длинная (красная) хорда, иначе – это короткая хорда. Ясно, что вероятность, что хорда – длинная равна:

$$\Pr(long) = \frac{1}{3}$$

Оценка №2. Проведем произвольный радиус. На радиусе выбираем произвольную точку и проводим хорду перпендикулярно радиусу (рис.2.1В). Вписанный треугольник поместим так, чтобы одна из его сторон была перпендикулярна радиусу. Это сторона разделит радиус пополам. Если точка, определяющая хорду, лежит ближе к центру, то это будет длинная хорда, иначе – короткая. Ясно, что вероятность, что хорда – длинная равна:

$$\Pr(long) = \frac{1}{2}$$

Оценка №3. Возьмем произвольную точку внутри круга. Проведем через нее радиус и, также как и в предыдущем случае, определим хорду как перпендикуляр к этому радиусу (рис.2.1С). Ясно, что, если точка лежит ближе, чем на $R/2$ к центру, то она определяет длинную хорду, иначе – это короткая хорда. Площадь области, определяющей длинные хорды равна $\pi R^2/4$, в то время как площадь круга равна πR^2 . Поэтому вероятность того, что хорда длинная равна:

$$\Pr(long) = \frac{1}{4}$$

Какая оценка правильная? Все оценки – правильные, просто здесь мы использовали разные разумные *модели* случайной хорды и получили разные оценки.

2.1.2 Природные модели последовательностей.

Итак, рассмотрим некоторые варианты природных моделей биологических последовательностей.

Модель №1. Рассмотрим последовательности в банке данных (например в GenBank) (рис.2.2А). Пронумеруем их. Затем с помощью датчика случайных чисел выберем номер последовательности и вытащим последовательность из банка данных с этим номером. Эта модель имеет вероятностную природу. Хорошо ли она отражает реальные (природные) данные? На самом деле не очень, если только мы не собираемся исследовать особенности банка данных. Действительно, там много разных последовательностей из разных источников, но явно перепредставлены последовательности из генома человека. Если бы там были только человеческие последовательности, то эта



Рис. 2.2: Природные модели последовательностей. А – Gene-банка; В – болото

модель отражала бы свойства человека. Но если говорить о представленности природных последовательностей, там явно недопредставлены последовательности бегемотов. По-видимому, представленность последовательностей из банка данных характеризует структуру финансирования проектов секвенирования. На бегемотов явно дают меньше денег, чем на человека – вот и последовательностей меньше.

Модель №2. Зачерпнем воды из болота и секвенируем какие-нибудь ДНК оттуда. Что мы получим? – Скорее всего там будут перепредставлены бактерии и бактериофаги. Будут еще амёбы, инфузории и что-нибудь в этом роде, но практически не будет последовательностей из лягушек, хотя *очевидно*, что именно они населяют болото. Представленность будет уже достаточно равномерная, правда более характерная для *этого* болота, но не для мира. Это тоже будет некоторая вероятностная модель мира (в данном случае болота). Здесь вероятность отражает количество той или иной ДНК.

Модель №3. Для того же болота секвенируем РНК. Что это будет? скорее всего там будет перепредставлены разнообразные рибосомные РНК. Здесь вероятность последовательности отражает количество РНК. Пример с болотом – прямой аналог рассуждений про хорду и треугольник.

Перечисленные вероятностные модели характеризуют разные особенности и надо хорошо понимать объект исследования и подбирать адекватные модели. Для статистического анализа каких-либо явлений необходимы «фоновые» или нулевые модели. Довольно часто для получения «фоновых» моделей используют случайные выборки, например случайные фрагменты генома человека. Однако в геноме много разных особенностей, например есть большое количество повторов, кодирующая часть генома занимает очень небольшую фракцию и фоновая модель должна как можно ближе соответствовать исследуемому объекту.

Допустим, вы обнаружили, что некоторые (например альтернативные) интроны обладают каким-то свойством (например, там избегаются CpG динуклеотиды). Для того, чтобы доказать, что свойство выбранных вами интронов статистически достоверно отличаются от других интронов, надо взять выборку всех не таких (например, конститутивных) интронов, но хорошо бы исключить первые интроны в генах, поскольку они часто содержат регуляторные элементы, например CpG острова, что создаст определенный перекося фоновой модели. Такой перекося может статистически подтвердить вашу гипотезу, но это будет ложное подтверждение. Построение адекватных фоновых моделей часто представляет собой нетривиальную задачу.

2.2 Бернуллевская модель

Однако все предыдущие модели очень трудны для математического анализа. Мы не имеем возможности экстраполировать данные. Например, если мы секвенировали новую последовательность, мы не можем определить вероятность того, что она соответствует той или иной модели. Самой простой моделью генерации последовательностей является модель испытаний Бернулли. Мы задаемся некоторым распределением частот букв (может быть она была известна из независимых источников) $p(\alpha)$. Далее, с помощью датчика случайных чисел выбираем очередную букву и помещаем в последовательность. Продолжаем это пока не надоест. В результате мы получим последовательность букв, которая является представителем случайной последовательности. Для любой последовательности x можно написать вероятность ее возникновения:

$$\Pr(x) = \prod p(x_i)$$

Но это неправильная вероятность. Возьмем все возможные последовательности данной длины L . Суммарная вероятность всех последовательностей заданной длины равна 1. Это легко доказать по индукции. Действительно, для последовательностей длины 1 суммарная вероятность равна сумме вероятностей порождения всех символов и равна 1. Допустим, что эта сумма верна для множества последовательностей длины $L - 1$, тогда

$$\begin{aligned} \sum_x \Pr(x) &= \sum_{x_1, \dots, x_L} \prod \Pr(x_i) \\ &= \sum_{x_1, \dots, x_L} \Pr(x_1) \Pr(x_2, \dots, x_L) \\ &= \sum_{x_1} \Pr(x_1) \sum_{x_2, \dots, x_L} \Pr(x_2, \dots, x_L) = 1 \cdot 1 = 1 \end{aligned}$$

Сумма вероятностей по последовательностям всех возможных длин стремится к бесконечности. Поэтому модель должна включать в себя распределение длин последовательностей $p(L)$. Тогда вероятность генерации после-

довательности $x = \{x_1, \dots, x_L\}$ будет

$$\Pr(\{x_1, \dots, x_L\}) = \prod_{x_1, \dots, x_L} p(x_i)p(L)$$

Таким образом, суммарная вероятность всех последовательностей будет равна

$$\begin{aligned} \sum_x \Pr(\{x_1, \dots, x_L\}) &= \sum_x p(x_i)p(L) \\ &= \sum_x \prod_{x_1, \dots, x_L} p(x_i) \sum(p(L)) \\ &= 1 \cdot \sum(p(L)) = 1 \cdot 1 = 1 \end{aligned}$$

Бернулевская генерация последовательности состоит из двух этапов – 1) выбор длины последовательности; 2) генерация L символов.

Контрольный вопрос. Пусть все буквы равновероятны. Дано две последовательности:

```
gcgttggcgagtttagtttgttgccatttcgcac
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Какая из двух представленных последовательностей более вероятна?

Для оценки параметров бернуллиевской модели можно использовать оценки правдоподобия или Байесовы оценки.

2.3 Марковские модели

Бернуллиевская модель генерации последовательностей не совсем адекватная. Например, в геноме человека редко встречается динуклеотид CpG. Здесь буква p означает фосфат. Эта запись используется для того, чтобы отличать CG как комплементарную пару от CG, расположенных на одной цепи.

Великий русский математик Андрей Марков обратил внимание на то, что в русском языке после буквы 'у' никогда не следует буква 'ь'. Более того, после гласной чаще встречается согласная буква, а после согласной буквы чаще идет гласная. Это наблюдение привело его к мысли, что при генерации слов русского языка, вероятности встречаемости букв зависят от предыдущей буквы.

Эта ситуация аналогична той, что наблюдается в геноме. Марковская модель генерации последовательностей заключается в следующем. Даны распределения вероятностей для первых символов $q(\alpha)$ – ведь последовательность надо с чего-то начинать – перед первой буквой нет другой буквы.

Для дальнейшей генерации есть переходные вероятности $p(\beta | \alpha)$:

$$\begin{aligned} q(\alpha); \quad \sum_{\alpha} q(\alpha) &= 1 \\ p(\alpha | \beta); \quad \sum_{\alpha} p(\alpha | \beta) &= 1 \end{aligned}$$

Если дан алфавит A , то число параметров, необходимых для описания марковской модели равно $|A| + |A|^2$ при $1 + |A|$ ограничениях, итого $|A|^2 - 1$ независимых параметров. Генерация происходит по следующему алгоритму. Задаемся длиной последовательности L . Генерируем первый символ из распределения $q(\alpha)$. Затем исходя из текущего символа генерируем следующее состояние с использованием распределения $p(\beta | \alpha)$. Вероятность (правдоподобие) последовательности, сгенерированной марковской цепью равна:

$$\Pr(x_1, \dots, x_L) = q(x_1) \prod_{i=2, \dots} p(x_i | x_{i-1}) p(L)$$

Допустим, у нас двух-буквенный алфавит (a,b), и даны распределения вероятностей:

$$\begin{aligned} q(\alpha) &= (1, 0) \\ p(\alpha | \beta) &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{aligned}$$

Такая марковская цепь будет порождать последовательности вида (abab...). Если все элементы матрицы переходных вероятностей отличны от 0, то марковская цепь обладает свойством *эргодичности*. Это значит, что статистические свойства одной последовательности при $L \rightarrow \infty$ будут такими же, как для многих разных последовательностей. Эта модель называется марковской моделью первого порядка, поскольку очередная буква зависит только от одного предыдущего символа. Для оценки вероятностей первых букв нам надо иметь много последовательностей.

Распределение вероятностей первого символа получаем из частот встречаемости первых символов. Однако, если наша модель обладает свойством эргодичности, для оценки вероятностей первого символа можно использовать просто частоты встречаемости символов в последовательностях. Для оценки переходных вероятностей можно подсчитать частоту встречаемости пар букв в последовательностях $f(\alpha\beta)$. Далее, используя теорему Байеса, оценим вероятности

$$\Pr(\alpha | \beta) = \frac{f(\alpha\beta)}{f(\beta)}$$

2.3.1 Модели высших порядков

Марковская модель первого порядка легко обобщается на случай, когда генерация следующего символа зависит от нескольких предыдущих символов

$\Pr(\alpha | \beta\gamma \dots)$. Тогда для генерации последовательности необходимы стартовые вероятности и переходные вероятности:

$$\begin{aligned} q(\alpha); \quad \sum_{\alpha} q(\alpha) &= 1 \\ q(\alpha | \beta); \quad \sum_{\alpha} q(\alpha | \beta) &= 1 \\ \dots \\ p(\alpha | \beta\gamma \dots); \quad \sum_{\beta\gamma \dots} p(\alpha | \beta\gamma \dots) &= 1 \end{aligned}$$

Для марковской модели порядка k мы имеем $|A| + |A|^2 + \dots + |A|^{k+1}$ параметров при наличии $1 + |A| + \dots + |A|^k$ ограничений. Итого $|A|^{k+1} - 1$ независимых параметров.

2.4 Оценка моделей

Пусть у нас есть некоторый набор последовательностей $X = \{x^k\}$ и несколько моделей. Как узнать, какая модель лучше? Для этого можно для каждой модели посчитать правдоподобие:

$$\Pr(X | M_r) = \prod_k \Pr(x^k | M_r)$$

Обычно правдоподобие – величина очень маленькая, поскольку она является произведением большого количества вероятностей. Для того, чтобы понять, какая модель лучше описывает данные X обычно вычисляют логарифм отношения правдоподобия:

$$L(M_1, M_2) = \log \frac{\Pr(X | M_1)}{\Pr(X | M_2)}$$

Если величина $L(M_1, M_2)$ положительна, то первая модель лучше описывает данные, иначе вторая лучше. Если мы будем применять очень сложные модели, содержащие большое число параметров, которые мы подбираем, то ясно, что правдоподобие от этого будет только увеличиваться. Значит ли это, что более сложная модель лучше описывает множество последовательностей? Это можно проверить, создав валидационную выборку. Разбиваем все множество объектов X (в нашем случае последовательностей) на две части – обучающую X_{tr} и валидирующую X_v . На первой выборке обучаем (подбираем параметры) обеих моделей, а на второй – проверяем, какая модель лучше. Обычно, увеличение количества параметров до определенного предела увеличивает качество описания, но, начиная с некоторого момента качество описания начинает падать. Для оценки качества модели часто используют не валидационные выборки, а информационные критерии. Первый критерий, AIC вычисляется так:

$$AIC(M) = 2k - 2 \ln \hat{L} \quad (2.1)$$

где k – число параметров модели, а $\hat{L} = \prod \Pr(X_i|M)$ – оптимальное значение правдоподобия. Среди всех моделей отбирают такую модель, которая имеет минимальное значение AIC , что обеспечивает баланс между качеством описания (\hat{L}) и количеством параметров. Этот критерий в 1971 году предложил Хиротугу Акаи. Основываясь на несколько других предположениях был предложен Байесов информационный критерий:

$$BIC(M) = k \ln(n) - 2 \ln \hat{L} \quad (2.2)$$

В этом критерии учитывается также n – размер обучающей выборки. Критерии этого типа используются в весьма широком диапазоне задач – от степенной регрессии до анализа моделей последовательностей.

Пример вычислений

Рассмотрим как это работает на примере марковской модели 1-го порядка. Пусть даны N последовательностей заданной длины L . Тогда можно оценить параметры модели (стартовые и переходные вероятности):

$$\begin{aligned} \Pr_0(\alpha) &= \frac{n_0(\alpha)}{N} && \text{стартовые вероятности} \\ \Pr(\beta|\alpha) &= \frac{\Pr(\alpha, \beta)}{\Pr(\alpha)} = \frac{n(\alpha\beta)}{n(\alpha)} && \text{переходные вероятности} \end{aligned}$$

здесь $n_0(\alpha)$ – число встреч буквы α в первой позиции последовательности; $n(\alpha\beta)$ – число пар $\alpha\beta$ на всей выборке; $n(\alpha)$ – число букв α на всей выборке. В предположении эргодичности можно считать, что стартовые вероятности просто равны частотам встречаемости букв в тексте.

Логарифм правдоподобия вычисляется через счетчики встречаемости букв и пар букв и оценивается как

$$\begin{aligned} \ln L &= \ln \left(\prod_{\alpha} \Pr_0(\alpha)^{n_0(\alpha)} \prod_{\alpha\beta} \Pr(\beta|\alpha)^{n(\alpha\beta)} \right) \\ &= \sum_{\alpha} n_0(\alpha) \cdot \ln \left(\Pr_0(\alpha) \right) + \sum_{\alpha\beta} n(\alpha\beta) \ln \left(\Pr(\beta|\alpha) \right) \\ &= \sum_{\alpha} n_0(\alpha) \cdot \ln \left(\frac{n_0(\alpha)}{N} \right) + \sum_{\alpha\beta} n(\alpha\beta) \ln \left(\frac{n(\alpha\beta)}{n(\alpha)} \right) \end{aligned}$$

Позже мы увидим, что эти формулы тесно связаны с информационным содержанием текста. Если последовательности в выборке разной длины, то при оценке правдоподобия следует учесть распределение длин последовательностей. В качестве размера обучающей выборки для ВИС оценки берем суммарную длину последовательностей.

Аналогично можно получить оценки и для случая марковской модели более высокого порядка. Для более сложных и изощренных моделей также все будет сводиться к информационным формулам вида $n \ln n/N$.

2.5 Частоты встречаемости слов

Рассмотрим простой пример. Пусть у нас есть двух-буквенный алфавит $A = \{a, b\}$. Рассмотрим все слова длиной 4. Таких слов 16. Подсчитаем количество вхождений aa и ab в эти слова:

	aa	ab		aa	ab
aaaa	3	0	baaa	2	0
aaab	2	1	baab	1	1
aaba	1	1	baba	0	1
aabb	1	1	babb	0	1
abaa	1	1	bbaa	1	0
abab	0	2	bbab	0	1
abba	0	1	bbba	0	0
abbb	0	1	bbbb	0	0

и построим распределение числа встреч слов (рис.2.3А):

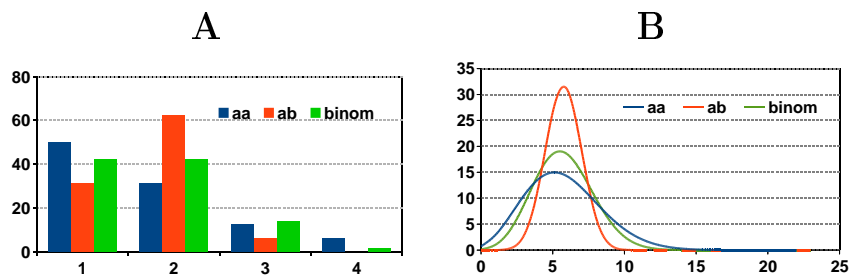
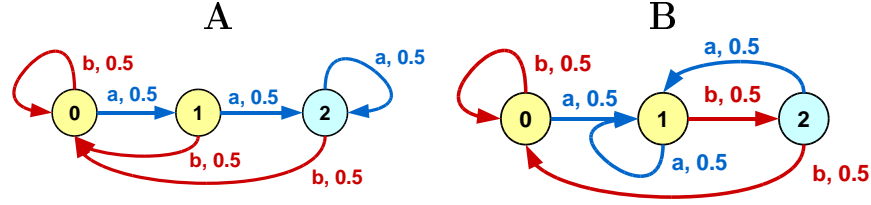


Рис. 2.3: Распределения вероятностей числа встреч слов aa и ab среди всех слов длиной 4 (А) и среди всех слов длиной 24 (В). Зеленый график соответствует биномиальному распределению.

Сравнение распределений показывает большую разницу. В чем причина? Дело в том, что в отличие от отдельных букв, события появления слова в разных местах текста не являются независимыми событиями. Так, если в позиции i встретилось слово ab , то в позиции $i + 1$ это слово заведомо не может появиться, в то время как для слова aa вероятность встречи слова в следующей позиции равна просто вероятности появления буквы a .

Здесь мы возвращаемся к истории, которая была в курсе алгоритмов. Поиск слова может быть описан как конечный автомат. При этом для слова aa и слова ab эти автоматы будут разными. При анализе частот встречаемости слов автомат является еще вероятностным, т.е. на каждой стрелке кроме символа из алфавита еще написана вероятность появления этого символа (рис.2.4). При чтении случайного текста автомат бродит между своими состояниями, время от времени заходя с допускающее состояние 2.

Важно, что распределения числа встреченных слов зависят от структуры слова, а именно от того, есть ли суффиксы, совпадающие с префикса-

Рис. 2.4: Конечные автоматы для поиска слов aa (A) и ab (B).

ми и как они расположены. Для того, чтобы оценить основные параметры распределения частот слов, вводят так называемый автокорреляционный многочлен $K_W(x) = \prod k_i x^i$. Если нам дано слово W длиной n , то автокорреляционный многочлен выглядит так:

$$K_W(x) = \prod_{i=0}^{i=n-1} k_i x^i; \quad k_i = \begin{cases} 0, & \text{suffix}_i(W) \neq \text{prefix}_i(W) \\ 1, & \text{suffix}_i(W) = \text{prefix}_i(W) \end{cases}$$

где $\text{suffix}_i(W)$ – суффикс слова W , начинающийся с позиции i , а $\text{prefix}_i(W)$ – префикс слова W длиной $n-i$. Например для слова АСАСАС коэффициенты этого многочлена будут $\{1, 0, 1, 0, 1, 0\}$:

A	C	A	C	A	C			k_i
A	C	A	C	A	C			1
		A	C	A	C	A	C	0
			A	C	A	C	A	1
				A	C	A	C	0
					A	C	A	1
						A	C	0

Математическое ожидание количества вхождений слова в текст не зависит от структуры слова. Однако, дисперсия зависит и вычисляется через автокорреляционный многочлен. В простейшем случае, когда все буквы в тексте равновероятны ($p_\alpha = p$) можно написать (вывод этих формул находится за пределами курса):

$$\begin{aligned} E(N) &= N \cdot p^n; & D(N) &= N \cdot p^n(2K_W(p) - 1 - (2n - 1)p^n) \\ E_{\text{binom}}(N) &= N \cdot p^n; & D_{\text{binom}}(N) &= N \cdot p^n(1 - p^n) \end{aligned} \quad (2.3)$$

Здесь второй строкой для сравнения даны формулы для биномиального распределения, которое возникло бы в предположении, что появления слов независимы. Например, для слова АСАСАС автокорреляционный многочлен выглядит так: $K_W(x) = 1 + x^2 + x^4$. Предполагая равную вероятность встреч нуклеотидов ($p = 0.25$), получаем $K_W(p) = 1 + p^2 + p^4 = 1.06$. Дисперсия будет равна $0.00028N$. Биномиальная оценка будет $0.00024N$. Но для слова АААААА дисперсия равна $0.00041N$, что намного больше биномиальной оценки. Наиболее сильно выражено отличие на коротких словах.

Для случая, когда буквы не равновероятны, формулы немного изменятся:

$$\begin{aligned} E(N) &= N \cdot p_W; & D(N) &= N \cdot p_W(2K_W(p) - 1 - (2n - 1)p_W) \\ p_W &= \prod_{i=0}^n p(W_i); & K_W(p) &= 1 + \sum_{l=1}^{n-1} k_l \prod_{i=1}^l p(W_i) \end{aligned} \quad (2.4)$$

Оказывается, что для марковских цепей формулы аналогичны. Вклад самопересечения слов довольно активно используется при поиске сигналов и оценке статистической значимости находок.

Глава 3

Выравнивания

3.1 Постановка задачи

В биологии гомологичными объектами называются объекты, имеющие общее происхождение. Например плавники китов гомологичны конечностям наземных млекопитающих, но не гомологичны плавникам рыб. Аналогично говорят, что две последовательности (нуклеотидные или аминокислотные) гомологичны, если они имеют общую предковую последовательность. Задача поиска гомологии последовательностей состоит в том, чтобы установить их общность происхождения.

Если при анализе морфологических особенностей животных мы можем среди ископаемых найти предка, то при анализе последовательностей нам, как правило, предки не известны. Исключением здесь являются некоторые вирусы, для которых известна их история. Например, если нам известен факт однократного заражения человека вирусом ВИЧ, то все вирусы в его организме являются потомками исходного вируса, а поскольку этот вирус очень быстро мутирует, то мы получаем популяцию несколько отличающихся друг от друга геномов, произошедших от общего предка.

Другой пример – эксперимент по длительной эволюции *E.coli* в пробирке. Здесь нам тоже достоверно известно общее происхождение геномов. Однако в большинстве случаев нам предки не известны и *не известен факт общего происхождения*. Задача выравнивания последовательностей звучит так. Написать две последовательности друг под другом так, чтобы *гомологичные* остатки располагались бы друг под другом. К сожалению, обычно нам неизвестен путь эволюции последовательностей. Поэтому мы используем *гипотезу* о том, что если структура двух последовательностей похожа, то *гомологичные* остатки будут занимать аналогичные позиции в структуре.

Алгоритм выравнивания последовательностей должен уметь написать две последовательности друг под другом, чтобы гомологичные остатки располагались друг под другом. Для проверки качества работы алгоритма будем использовать сравнения пространственных структур. Если алгоритм

будет хорошо строить выравнивания на этом множестве последовательностей, то есть *надежда*, что для последовательностей, для которых не известна пространственная структура, выравнивание тоже будет правильным. Множество выравниваний, полученных из сравнения пространственных структур назовем *золотым стандартом*.

3.1.1 Число выравниваний

Сколько всего существует возможных выравниваний для двух последовательностей? Для ответа на этот вопрос построим последовательность из 0 и 1 следующим образом (рис.3.1). Читаем выравнивание по колонкам сверху-

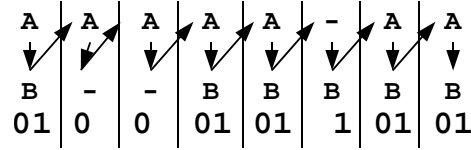


Рис. 3.1: Преобразование выравнивания в строку.

вниз. Если есть буква в первой строке, то ставим 0 иначе ничего не ставим; Если есть буква во второй строке – пишем 1, иначе ничего не пишем. Заметим, что это представление обратимо – по строке нулей и единиц можно восстановить выравнивание. В этой строке ровно столько 0, какова длина первой последовательности и ровно столько 1, какова длина второй последовательности. Количество возможных выравниваний равно числу способов разместить в строке n нулей и m единиц:

$$N_{al} = C_{n+m}^n = \frac{(n+m)!}{n!m!}$$

Полагая $n = 1$ и применяя формулу Стирлинга, получаем:

$$N_{al} = C_{2n}^n = \frac{(2n)!}{(n!)^2} \simeq \frac{\sqrt{2\pi} \cdot 2n (2n)^{2n} e^{-2n}}{2\pi n \cdot n^{2n} e^{-2n}} = \frac{2^{2n}}{\sqrt{\pi n}}$$

т.е. количество возможных выравниваний растет практически экспоненциально с увеличением длины последовательностей. В этих рассуждениях есть маленький изъян. Есть вырожденный случай – два разных выравнивания, показанных ниже формально разные, но имеют одинаковую кодировку:

ABBAABV	ABVV-ABV
ABVBAVB	ABV-AABV
01010101010101	01010101010101

Это означает, что полученная оценка несколько занижена, а число выравниваний еще больше.

3.2 Глобальное выравнивание

3.2.1 Редакционное расстояние

Простейшим соображением для построения выравнивания является идея редакционного расстояния. Есть эволюционный процесс, который может заменять, удалять и вставлять буквы. Каждая такая операция называется *операцией редактирования*. Наверное, природа устроена так, что этих событий должно быть минимальное количество.

Редакционным расстоянием между двумя текстами назовем минимальное количество операций редактирования, преобразующих один текст в другой. Для решения этой задачи построим граф. Вершины графа – различные пары префиксов текстов. Этот граф можно представить в виде прямоугольной решетки (рис.3.2). Ребра определим следующим образом. Гори-

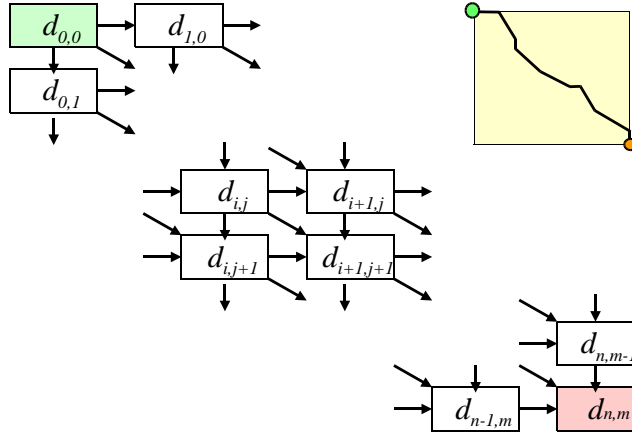


Рис. 3.2: граф редакционного расстояния.

зонтальные и вертикальные ребра соответствуют операциям вставки/удаления символа, диагональные ребра – сопоставление символов. На ребрах пишем веса. Вставка/удаление – одна операция редактирования, сопоставление – если символы совпали, то нет операции редактирования, если не совпали – то есть операция замены:

$$\begin{aligned} e((i, j) \rightarrow (i + 1, j)) &= 1 \\ e((i, j) \rightarrow (i, j + 1)) &= 1 \\ e((i, j) \rightarrow (i + 1, j + 1)) &= 1 - \delta(x_i, y_j) \end{aligned}$$

Таким образом, из выравнивания более коротких префиксов получаем выравнивание более длинных. Путь по этому графу от начала (пустые префиксы, левый верхний угол) до конца (префиксы совпадают с полными

последовательностями) соответствует некоторому набору операций редактирования. Для определения оптимального (кратчайшего) пути можно применить динамическое программирование, т.е. будем строить оптимальное выравнивание от более коротких префиксов к более длинным. На вершинах определим вес d_{ij} . Оптимальный вес вершины определим рекурсивно:

$$\begin{aligned} d_{0,0} &= 0 && \text{-- инициализация} \\ d_{i,j} &= \min \begin{cases} d_{i-1,j} & +1, & i > 0 \\ d_{i,j-1} & +1, & j > 0 \\ d_{i-1,j-1} & +1 - \delta(x_i, y_j), & i, j > 0 \end{cases} && \text{-- рекурсия} \\ D(x, y) &= d(n, m) && \text{-- завершение} \end{aligned}$$

Для восстановления собственно выравнивания надо запомнить оптимальные переходы $\pi_{i,j} = [k, l]$ – координаты ячейки на которой реализуется минимум

$$\pi_{i,j} = \arg \min \begin{cases} d_{i-1,j} & +1, & i > 0 \\ d_{i,j-1} & +1, & j > 0 \\ d_{i-1,j-1} & +1 - \delta(x_i, y_j), & i, j > 0 \end{cases}$$

Совершая обратный проход с использованием $\pi_{i,j}$ мы узнаем оптимальный набор операций редактирования.

3.2.2 Алгоритм Нидльмана-Вунша

В биоинформатике используют не минимизацию редакционного расстояния, а уровень сходства. Для того, чтобы перейти от одной задачи к другой, заменим расстояния $d_{i,j}$ на $-d_{i,j}$. Тогда операцию \min надо заменить на \max . Далее прибавим $1/2$ к $-d_{i,j}$ и получим функцию сходства $w_{i,j} = 1/2 - d_{i,j}$. При этой подмене мы получаем такую схему оценки: совпадение – премия $1/2$, замена – штраф $1/2$, вставка/делеция – штраф 1. Такое преобразование позволяет заметно обобщить задачу.

Новая задача выглядит так: Даны две последовательности и система оценки качества выравнивания – премия за совпадение *match*; штраф за замену *mism* и штраф за вставку/делецию *del*. Задача: построить выравнивание, максимизирующее вес выравнивания. Для решения этой задачи строим алгоритм максимизации веса выравнивания:

$$\begin{aligned} w_{0,0} &= 0 && \text{-- инициализация} \\ w_{i,j} &= \max \begin{cases} w_{i-1,j} & -del, & i > 0 \\ w_{i,j-1} & -del, & j > 0 \\ w_{i-1,j-1} & +match, & i, j > 0, x_i = y_j \\ w_{i-1,j-1} & -mism, & i, j > 0, x_i \neq y_j \end{cases} && \text{-- рекурсия} \\ W(x, y) &= d_{n,m} && \text{-- завершение} \end{aligned} \tag{3.1}$$

Для восстановления оптимального пути также запоминаем оптимальные

переходы $\pi_{i,j}$:

$$\pi_{i,j} = \arg \max \begin{cases} w_{i-1,j} & -del, & i > 0 \\ w_{i,j-1} & -del, & j > 0 \\ w_{i-1,j-1} & +match, & i, j > 0, x_i = y_j \\ w_{i-1,j-1} & -mism, & i, j > 0, x_i \neq y_j \end{cases}$$

Представленный алгоритм рассматривает только факт совпадения или замены. Однако для аминокислот известно, что некоторые из них имеют схожие физико-химические свойства, в то время как другие сильно отличаются друг от друга. Поэтому вместо строк $w_{i-1,j-1} + match$, $w_{i-1,j-1} - mism$ используют матрицу замен аминокислотных остатков $M(\alpha, \beta)$. Эта матрица учитывает степень сходства или различия свойств аминокислотных остатков – на похожих аминокислотах ее значения положительны, а на различных – отрицательны. В этом случае рекурсия будет иметь вид:

$$w_{i,j} = \max \begin{cases} w_{i-1,j} & -del, & i > 0 \\ w_{i,j-1} & -del, & j > 0 \\ w_{i-1,j-1} & +M(x_i, y_j), & i, j > 0, \end{cases} \quad (3.2)$$

Рекурсия (3.2) называется алгоритмом *Нидльмана-Вунша*.

Для построения оптимального пути нам необходимо заполнить все ячейки матрицы, и при их заполнении мы выполняем конечное количество операций. Поэтому время работы алгоритма равно $T = O(mn)$. Рекурсия (3.2) требует памяти для заполнения матрицы. Однако, в этой рекурсии можно обойтись и линейной памятью – для вычислений достаточно помнить лишь предыдущую строчку. Как это запрограммировать – хорошее упражнение по программированию. Однако избежать заполнения матрицы обратных переходов не удастся. Поэтому для работы алгоритма необходима память порядка $M = O(mn)$.

3.2.3 Алгоритм Миллера-Майерса

Можно построить алгоритм, которому необходима лишь линейная память. В современном мире особых проблем с памятью нет, однако идеи, лежащие в основе этого алгоритма применяются в ряде других алгоритмов.

Алгоритм Миллера-Майерса состоит в следующем. Разбиваем одну из последовательностей на две равные части, тем самым разбиваем матрицу выравнивания на две равные части. Для каждой точки $(m/2, j)$ линии раздела находим веса оптимальных выравниваний из начала в $(m/2, j)$ и из конца в $(m/2, j)$ (рис.3.3А):

$$W^+(m/2, j), \quad W^-(m/2, j)$$

При этом, во-первых, при динамическом программировании мы не запоминаем обратный проход π , а во-вторых, мы используем линейную по памяти

схему поиска оптимума. Для вычисления W^- мы делаем динамическое программирование из конца в начало. Вес оптимального выравнивания, проходящего через точку $(m/2, j)$, равен:

$$W(x) = W^+(m/2, j) + W^-(m/2, j)$$

Вес оптимального пути из начала в конец есть (рис.3.3В)

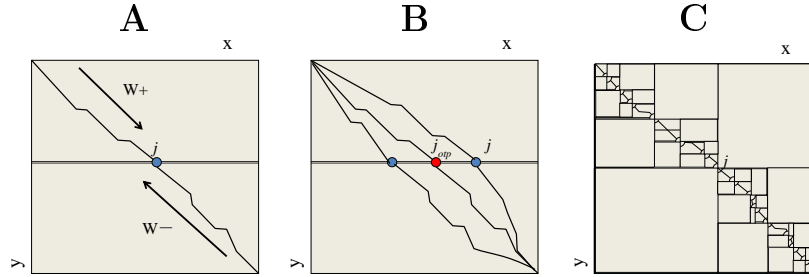


Рис. 3.3: Алгоритм Миллера-Майерса

$$W(x, y) = \max_j (W^+(m/2, j) + W^-(m/2, j))$$

Найдя максимум и определив позицию j_{opt} , где он достигается, мы находим одну точку на оптимальном пути выравнивания $(i_1, j_1) = (m/2, j_{opt})$. Теперь все пространство выравнивания разбито на два прямоугольника $(0, 0) : (i_1, j_1)$ и $(i_1, j_1) : (m, n)$ (рис.3.3С). На этих прямоугольниках тоже можно применить тот же прием и получить еще две точки, через которые проходит путь оптимального выравнивания $(i_2, j_2), (i_3, j_3)$. Теперь каждый из прямоугольников еще раз разобьется на две подматрицы, где можно будет найти еще 4 точки на оптимальном пути. Так будем продолжать, пока размер прямоугольников не схлопнется.

Казалось бы, за возможность не запоминать обратный проход мы делаем очень много лишних операций. Но это не совсем так. При каждом делении мы сокращаем объем просматриваемых матриц в 2 раза. Поэтому время работы алгоритма будет:

$$T = O(mn + nm/2 + mn/4 + mn/8 + \dots) = O(2mn)$$

т.е. время работы всего только удвоилось.

Поиск условно-оптимального выравнивания. Прием, когда всё пространство можно разбить на две части, которые мы просматриваем в противоположных направлениях, позволяет искать условно-оптимальные решения. Пусть нам достоверно известно, что остатки r, s должны быть сопоставлены в выравнивании. Как найти оптимальное выравнивание при этом

условии? Делаем проход сверху-вниз, как показано в формуле (3.1) и запоминаем матрицу обратных проходов $\pi = \pi^+$. Потом делаем такую же рекурсию в обратном направлении и получаем новую матрицу обратных проходов π^- . Далее, стартуя с точки r, s с помощью матрицы π^+ находим оптимальный путь $(0, 0) \rightsquigarrow (r, s)$ и с помощью матрицы обратного хода π^- находим оптимальный путь $(r, s) \rightsquigarrow (m, n)$

3.2.4 Выравнивание в полосе

Выравнивания с большими делециями, как правило, не имеют биологического смысла (рис.3.4). Ясно, что выравнивания D1 и D2 не представляют интереса, поскольку содержат в основном делеции, а разумные выравнивания (A) лежат в полосе. Это позволяет модифицировать алгоритм

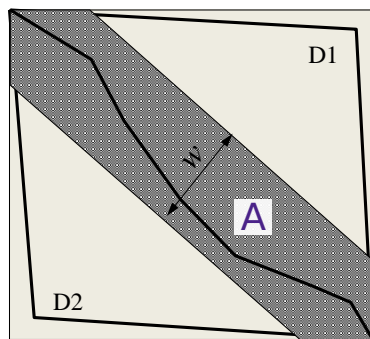


Рис. 3.4: Выравнивание в полосе

Нидльмана-Вунша: задаемся шириной полосы w и просматриваем только те вершины графа, что лежат в указанной полосе. Этот прием весьма широко используется в тех случаях, когда нам известно примерно, где находится искомое выравнивание. Сложность алгоритма теперь будет $T = O(wm)$.

3.3 Локальное выравнивание

Алгоритм Нидльмана-Вунша строит *глобальное выравнивание*, требуя, чтобы все остатки были по-возможности сопоставлены. Но надо понимать, что обычно источником аминокислотных последовательностей являются формальные трансляции нуклеотидных последовательностей, сделанные по предсказаниям генов — это не есть реальные аминокислотные последовательности, полученные экспериментально. При предсказании генов часто не точно предсказывается старт кодирующей области. Кроме того есть и биологические особенности, такие как эволюционная подвижность стартов и концов кодирующих областей. Более того, бывает слияние генов, когда в одном организме есть два белка, а в другом они слиты в один белок. Эти и

многие другие особенности говорят о том, что если концы аминокислотных последовательностей не выравниваются, то не надо за это штрафовать. Всё это приводит к идее *локального выравнивания*. Локальным выравниванием будем называть такое выравнивание *фрагментов* последовательностей $x[r \dots s]$, $y[t \dots v]$, которое имеет наибольший вес. Никакое изменение этих фрагментов (позиций r, s, t, v) не может увеличить вес выравнивания.

3.3.1 Оптимальное локальное выравнивание

Задачу построения локального выравнивания можно сформулировать как задачу поиска пути в графе наибольшего веса. Для этого из начальной вершины B проведем во все вершины ребра с нулевым весом, а также из всех вершин в конец E также проведем ребра нулевого веса (рис.3.5). Тогда

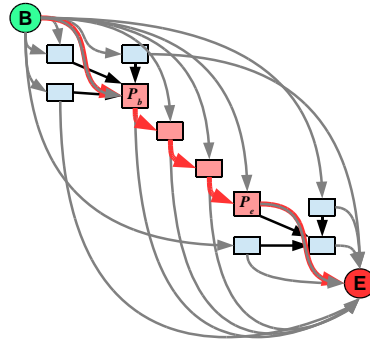


Рис. 3.5: Поиск пути максимального веса. Серые стрелки – ребра нулевого веса. Красные стрелки – некоторый путь в графе.

путь из B в E в таком графе будет состоять из ребра $B \rightarrow P_b$, пути $P_b \rightarrow P_e$ и ребра $P_e \rightarrow E$. Поскольку веса ребер $B \rightarrow P_b$ и $P_e \rightarrow E$ равны 0, то вес пути из начала в конец $B \rightarrow E$ будет равен весу пути $P_b \rightarrow P_e$. Поэтому поиск пути максимального веса есть поиск пути максимального веса из начала B в конец E . Запишем рекурсию для поиска наилучшего локального выравнивания:

$$w_{i,j} = \max \begin{cases} w_{i-1,j} & -del, & i > 0 \\ w_{i,j-1} & -del, & j > 0 \\ w_{i-1,j-1} & +M(x_i, y_j), & i, j > 0, \\ 0 & \end{cases} \quad (3.3)$$

$w(i, j)$ – это наилучший вес из начала в позицию (i, j) . Здесь 0 соответствует переходу из B в текущую позицию. В этом алгоритме также надо запоминать обратные переходы $\pi(i, j)$. Однако этот алгоритм не дает ответа на вопрос из какой позиции лучше всего уйти в конец, т.е. нет выбора ребра $(i, j) \rightarrow E$. Ответ достаточно очевиден. Надо просто найти максимум

значений $w(i, j)$:

$$\pi_E = (i_{max}, j_{max}) = \arg \max w(i, j) \quad (3.4)$$

Теперь можно сделать обратный проход, стартуя с этой точки. Описанный алгоритм (3.3, 3.4) является частью алгоритма локального выравнивания *Смита-Ватермана*.

3.4 Штрафы за делеции

Классический алгоритм Нидльмана-Вунша предполагает, что штраф за делецию пропорционален длине делеции, что означает, что одиночные пропуски появляются независимо. Это приводит к тому, что выравнивания имеют разбросанные делеции, а наблюдение над выравниваниями из золотого стандарта показывает, что, как правило, делеции группируются, и выравнивания имеют блочную структуру. Наверное, штраф за делеции должен иметь зависимость, как на рис.3.6

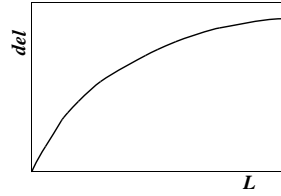


Рис. 3.6: Ожидаемая зависимость величины штрафа от длины делеции

3.4.1 Общие штрафы

Пусть дана некоторая функция штрафов за делеции $\Delta(l)$. Построим алгоритм динамического программирования. В классическом алгоритме Нидльмана-Вунша делеции соответствовал переход на предыдущую позицию $(i - 1, j)$ или $(i, j - 1)$. Если было несколько делеций подряд, то они суммировались, например: $w_{del_x}(i, j) = w_{i-1,j} - del = w_{i-2,j} - 2del = \dots$. При общем штрафе за делецию мы должны просмотреть все возможные предыдущие позиции (рис.3.7):

$$w_{del_x}(i, j) = \max \{ w_{i-1,j} - \Delta(1), \\ w_{i-2,j} - \Delta(2), \\ w_{i-3,j} - \Delta(3), \dots \}$$

аналогично для делеций в последовательности y . Таким образом, рекурсия для локального выравнивания примет вид

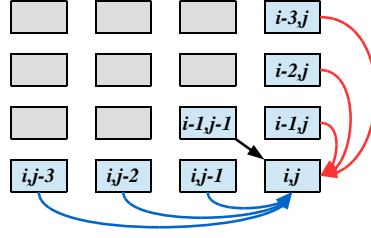


Рис. 3.7: Выравнивание при общих штрафах за делеции

$$w_{i,j} = \max \begin{cases} \max_l \{w_{i-l,j} - \Delta(l)\} \\ \max_l \{w_{i,j-l} - \Delta(l)\} \\ w_{i-1,j-1} + M(x_i, y_j) \\ 0 \end{cases} \quad (3.5)$$

Легко провести анализ этого алгоритма. Мы заполняем матрицу размером $m \times n$ и при заполнении каждой клетки выполняем порядка $m + n$ операций для определения оптимального размера делеции. Итого, время работы алгоритма порядка $T = O(n \cdot m \cdot (m + n))$, т.е. кубическое (если $m = n$). Можно, конечно, ограничить размер делеции, заявив, что непрерывная делеция не может быть длиннее заданного размера d_{max} . Тогда трудоемкость алгоритма будет порядка $t = O(m \cdot n \cdot d_{max})$.

Замечание. Если зависимость штрафа от длины такая, что $f(a + b) > f(a) + f(b)$, то такой штраф лишен смысла, поскольку для блока делеций все равно выгодно их рассматривать как сумму независимых делеций и все делеции будут рассматриваться как одиночные. Поэтому использование такой функции эквивалентно рассмотренным ранее линейным штрафам.

3.4.2 Аффинные штрафы

Приятным исключением является случай, когда функция делеции имеет вид $\Delta(l) = d_{open} + d_{ext} \cdot l$. Такой вид функции делеций называется *аффинными штрафами* за делецию в отличие от *линейных штрафов* $\Delta(l) = del \cdot l$. Аффинные штрафы позволяют построить эффективный алгоритм динамического программирования. Путь выравнивания разбивается на фрагменты — есть делеционный путь, где вес линейно зависит от длины и диагональный путь, где вес суммируется в соответствии с матрицей замен. А между этими путями есть переходы, связанные с открытием и закрытием делеций. Поэтому в каждой ячейке матрицы мыведем три переменные $X(i, j)$, $Y(i, j)$, $w(i, j)$, где $X(i, j)$, $Y(i, j)$ отвечают делеционным путям по последовательностям x , y , а $w(i, j)$ соответствует диагональному пути (рис.3.8А). Из этой схемы легко получить рекурсии для $X(i, j)$, $Y(i, j)$, $w(i, j)$:

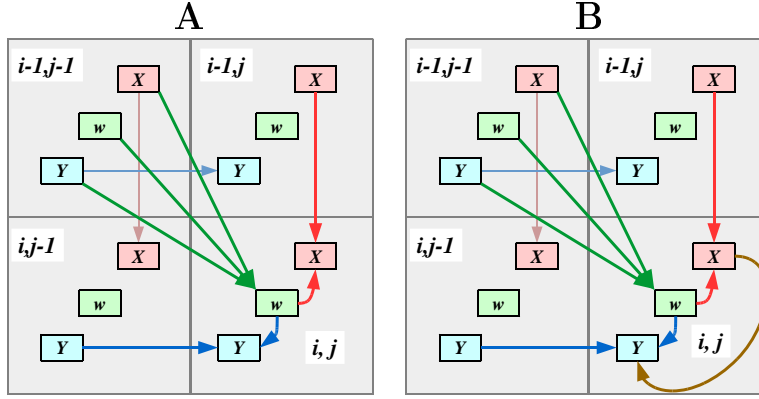


Рис. 3.8: Выравнивание при аффинных штрафах за делеции (А). Выравнивание с учетом не выравниваемых участков (В). Коричневая стрелка – переход из состояния X в состояние Y

$$w(i, j) = M(x_i, y_j) + \max \begin{cases} X(i-1, j-1) \\ Y(i-1, j-1) \\ w(i-1, j-1) \\ 0 \end{cases} \quad (3.6)$$

$$X(i, j) = \max \begin{cases} X(i-1, j) - d_{ext} \\ w(i, j) - d_{open} \end{cases}$$

$$Y(i, j) = \max \begin{cases} Y(i, j-1) - d_{ext} \\ w(i, j) - d_{open} \end{cases}$$

Естественно, надо запомнить обратные переходы. Поскольку здесь три типа переменных, то надо завести три типа обратных переходов π_w , π_X , π_Y и в этих переменных надо запоминать не только предыдущие координаты, но и тип переменной, откуда пришел оптимум. При программной реализации обратные переходы запоминаются проще, поскольку переход π_X – всегда на один шаг вверх, π_Y – на один шаг влево, а π_w всегда по диагонали, поэтому достаточно помнить только из какого состояния мы вошли в то или иное состояние. Алгоритм поиска оптимального локального выравнивания с аффинными штрафами за делеции называется алгоритмом Смита-Ватермана.

3.4.3 Невыравниваемые фрагменты

Часто в выравниваниях появляются участки, которые сильно различаются. Это, как правило петли в пространственной структуре. Они к тому же могут иметь разную длину. Эти фрагменты последовательностей не надо выравнивать и пытаться как-то сравнивать аминокислотные остатки в них. Для того, чтобы учесть не выравниваемые участки, надо просто в граф добавить переход из состояния X в состояние Y без штрафов (рис.3.8В) При таком подходе не выравниваемые участки штрафуются так же как и де-

леции. Более того, делеция является частным случаем не выравниваемого участка.

Упражнение. Модифицируйте рекурсию (3.6) так, чтобы допускались не выравниваемые фрагменты.

3.5 Статистики выравниваний

Допустим, мы построили выравнивание и получили вес выравнивания, скажем, 18. Много это или мало? Может быть выравнивание случайных последовательностей даст тот же результат? А как зависит вес выравнивания от длины последовательностей?

В этом разделе мы попробуем понять, как зависит вес выравнивания от его длины на случайных последовательностях, скажем, бернуллиевских. Для этого анализа вернемся к простейшей схеме весов выравниваний – премия за совпадение *match*, штраф за замену *mism* и штраф за делецию *del*. Заметим, что если мы умножим все параметры на одну и ту же положительную константу, то все веса выравниваний умножатся на эту константу, но сама структура оптимального выравнивания не изменится. Поэтому можно зафиксировать премию за совпадение $match = 1$.

3.5.1 Наибольшая общая подпоследовательность

Рассмотрим предельный случай, когда мы не штрафует за замены и делеции $mism = del = 0$ и будем искать оптимальное выравнивание. Вес оптимального выравнивания в этом случае равен длине *наибольшей общей подпоследовательности* (рис.3.9А). *Подпоследовательностью* называется последовательная выборка из последовательности, например, выбираем элементы с номерами 3,5,6,7,12,37. Важно, чтобы последовательность номеров была строго возрастающей. Покажем, что на случайных последователь-

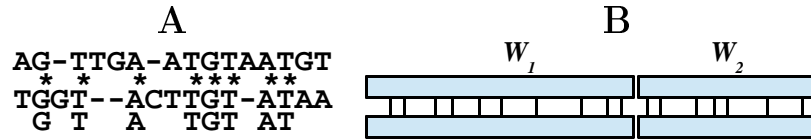


Рис. 3.9: Выравнивание и общая подпоследовательность.

стях вес оптимального выравнивания линейно зависит от длины последовательности. Разобьем обе последовательности на две части и построим отдельно оптимальные выравнивания первых частей последовательностей и вторых частей последовательностей (рис.3.9В). Получим веса выравниваний

$$W_1 = W(x_1, y_1), \quad W_2 = W(x_2, y_2)$$

Ясно, что вес оптимального выравнивания полных последовательностей не может быть меньше суммы весов подвыравниваний (Почему?):

$$W(x, y) \geq W_1 + W_2$$

Отсюда следует, что оптимальный вес выравнивания не меньше, чем некоторая линейная функция от длины последовательности и математическое ожидание веса выравнивания не меньше, чем некоторая линейная функция от длины (линейность следует из аддитивности):

$$E(W) \geq C \cdot L$$

С другой стороны, есть идеальное выравнивание, когда две последовательности изначально были идентичны. Вес такого выравнивания равен длине последовательности. Вес любого выравнивания не превышает веса идеального выравнивания, поэтому

$$E(W) \leq L$$

т.е. длина наибольшей общей подпоследовательности с двух сторон зажата линейными функциями:

$$C \cdot L \leq E(W) \leq L \quad (3.7)$$

и поэтому линейно зависит от длины последовательности. Заметим, что здесь нигде не использовалась модель последовательности. Важно только, что последовательность можно было разбивать на куски. Поэтому уравнение (3.7) имеет весьма широкую область применимости. Коэффициент в указанном линейном поведении зависит от модели последовательности.

Упражнение. Задача наибольшей общей подпоследовательности ставится как задача глобального выравнивания. Однако, если мы будем решать задачу локального выравнивания при параметрах $mism = del = 0$ мы получим точно такое же выравнивание, что и при глобальном подходе. Докажите это.

3.5.2 Наибольшее общее слово

Рассмотрим теперь другой предельный случай $mism = del = \infty$, и будем искать локальное выравнивание. В таком выравнивании не допускаются ни вставки/делеции ни замены, т.е. буквы в выравнивании совпадают и идут без разрыва. Это значит, что в двух последовательностях есть *общее слово*, а задача построения оптимального локального выравнивания есть задача поиска *наибольшего общего слова*. Для анализа веса такого выравнивания наложим одну последовательность на другую, быть может с некоторым сдвигом, и пойдем вдоль пары последовательностей, отмечая совпадения букв как успех (рис.3.10). Из теории вероятностей известно, что математическое ожидание длины максимальной серии успехов r , идущих подряд

```

acggtatatgacagtcgctcgcacacaagggcccaattgcttgtga
cggcgatttgcgatggctcgctgaatttgcggtggtattgcgtatgg
0010011001100000000100100000011000011111000110

```

Рис. 3.10: Серии успехов (отмечены красным)

в последовательности N независимых испытаний логарифмически зависит от длины серии испытаний:

$$E(r) = \log_{1/p} N$$

где p – вероятность успеха. Однако при выравнивании мы накладываем одну последовательность на другую с разными сдвигами. Каждому сдвигу соответствует соответствующая диагональ на матрице сопоставления. Можно сложить (конкатенировать) все диагонали и получить одну длинную серию испытаний Бернулли. Длина серии испытаний будет равна $m \cdot n$. Поэтому максимальная серия успехов на таком длинном испытании будет порядка $\log_{1/p}(m \cdot n)$. Однако надо принять во внимание краевые эффекты – ведь максимальная серия успехов может проходить через точку сшивки диагоналей. Кроме того, испытания при таком подходе не совсем независимы. Поэтому более точная формула имеет вид:

$$\begin{aligned}
E(l) &\approx \log_{1/p}(nm \cdot (1 - p)) + \gamma \cdot \log_{1/p} e - 1/2 = \log_{1/p}(Knm), \\
(m, n &\rightarrow \infty, \gamma \approx 0.577) \\
\sigma(l) &\approx \left(\pi \log_{1/p}(e) \right)^2 + 1/2
\end{aligned}$$

Таким образом, длина наибольшего общего слова зависит от длины логарифмически.

3.5.3 Два типа поведения выравнивания

Можно показать, что все пространство параметров (в простейшем случае параметров *mism*, *del*), даже если использовать матрицу сопоставления и аффинные штрафы, разбивается на две области: в одной – логарифмическая зависимость веса выравнивания от длины, а в другой – линейная. Если параметры пришли из линейной области, то выравнивания – рыхлые с большим количеством делеций и, как правило эти выравнивания мало похожи на биологически осмысленные. С другой стороны, если параметры пришли из логарифмической области, то выравнивания гораздо более приемлемы.

3.6 Матрицы сопоставления аминокислотных остатков

В алгоритмах выравнивания используются матрицы сопоставления аминокислотных остатков. Эти матрицы должны отражать степень сходства

3.6. МАТРИЦЫ СОПОСТАВЛЕНИЯ АМИНОКЛОТНЫХ ОСТАТКОВ 57

остатков, а, точнее, частоту их замен. Наивное представление о заменах основано на представлении о частотах мутаций. Однако это представление категорически не верное. Одиночная замена в кодоне GAT → GGT приводит к замене аспартата на глицин, т.е. к значительному изменению свойств остатка. Более правильное понимание процесса заключается в том, что аминокислотные остатки в белках подвержены эволюционному *отбору*. Поэтому правильно говорить о заменах как о *зафиксированных* мутациях.

В основе построения матриц аминокислотных замен лежат наблюдения. Допустим, у нас есть много выравниваний. Два остатка могут быть сопоставлены, если они имеют общего предка. Пусть вероятность того, что остатки α , β сопоставлены в выравнивании будет $P(\alpha, \beta | M)$. Другая модель предполагает, что остатки были порождены независимо $P(\alpha, \beta | R) = P(\alpha)P(\beta)$. Рассмотрим выравнивание. Вероятность выравнивания (без учета позиций с делециями) при случайной модели независимых последовательностей есть:

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

Если же выравнивание имеет смысл и порождено моделью, в которой остатки произошли от общего предка, то

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

Вычислим логарифм отношения правдоподобия для выравнивания:

$$\begin{aligned} L = \log \frac{P(x, y | M)}{P(x, y | R)} &= \log \prod_i \frac{P(x_i, y_i | M)}{P(x_i)P(y_i)} \\ &= \sum_i \log \frac{P(x_i, y_i | M)}{P(x_i)P(y_i)} \\ &= \sum_i M(x_i, y_i) \end{aligned} \quad (3.8)$$

таким образом, матрицу замен остатков можно представить в виде:

$$M(\alpha, \beta) = \log \frac{P(\alpha, \beta | M)}{P(\alpha)P(\beta)} \quad (3.9)$$

Очевидно, что $P(\alpha)$ — это просто частота встречаемости аминокислоты α в мире. Для оценки вероятностей $P(\alpha, \beta | M)$ нам нужны выравнивания, в которые мы верим. Можно было бы использовать выравнивания, построенные при анализе пространственных структур, однако этих выравниваний мало для того, чтобы получить достаточно хорошие оценки вероятностей замен $P(\alpha, \beta | M)$. Кроме того, по-видимому, эти вероятности должны зависеть от эволюционного расстояния. Поэтому для построения матриц замен используют другие подходы.

3.6.1 Матрицы РАМ

Процесс эволюции можно представить себе как марковский процесс, при котором на каждом интервале времени происходят небольшие изменения (гипотеза молекулярных часов). Тогда, если мы знаем вероятности замен на небольшом интервале времени $P_{\Delta t}(\alpha | \beta)$ мы можем определить вероятности замен на большем интервале:

$$P_{2\Delta t}(\alpha | \beta) = \sum_{\gamma} P_{\Delta t}(\alpha | \gamma) P_{\Delta t}(\gamma | \beta)$$

Здесь надо понимать, что в матрице вероятностей замен частным случаем является отсутствие замен. Заметим, что в этой формуле мы просто умножаем матрицы. Поэтому можно написать вероятности замен на больших интервалах $n\Delta t$:

$$P_{n\Delta t} = P_{\Delta t}^n \quad (3.10)$$

Для возведения матрицы в большую степень можно применить такой прием. При некоторых условиях матрицу A можно представить в виде произведения $A = UDU^{-1}$, где D – диагональная матрица. Тогда любая степень матрицы A может быть легко вычислена;

$$A^n = A \times A \times A \dots = UDU^{-1}UDU^{-1} \dots UDU^{-1} = UD^nU^{-1}$$

Возведение в степень диагональной матрицы не представляет проблем – надо просто все диагональные элементы возвести в эту степень. Остался вопрос как построить матрицу замен для маленьких интервалов времени. Возьмем пары последовательностей, в которых наблюдается 2 замены на 100 остатков. Эти последовательности отошли от общего предка на одну замену на 100 остатков. Величину эволюционного расстояния от общего предка будем называть РАМ (Point Accepted Mutations). Рассматриваемые последовательности находятся на расстоянии 1РАМ.

Построить выравнивание на таком эволюционном расстоянии не представляет проблем – можно использовать алгоритм выравнивания, который не базируется ни на какой матрице, а только подсчитывает число замен. Глядя на эти выравнивания можно построить марковскую матрицу условных вероятностей $P_{РАМ1}(\alpha | \beta)$. Далее мы можем возвести эту матрицу в достаточно большую степень и получить матрицу условных вероятностей $P_{РАМN}(\alpha | \beta)$.

Отметим, что если в исходной матрице не было видно ни одного сопоставления, скажем глицина с триптофаном, после возведения матрицы в высокую степень такие сопоставления становятся возможными, поскольку переход от одной аминокислоты к другой происходит через множество промежуточных событий, которые были замечены в матрице переходных вероятностей $P_{РАМ1}$. Этот подход к построению матриц был в 1972г. предложен Маргарэт Дейхофф. Итак, матрица замен основанная на Марковской модели эволюции имеет вид:

$$M(\alpha, \beta) = \log \frac{P(\alpha, \beta)}{P(\alpha)P(\beta)} = \log \frac{P(\alpha | \beta)P(\beta)}{P(\alpha)P(\beta)} = \log \frac{P(\alpha | \beta)}{P(\alpha)} \quad (3.11)$$

3.6. МАТРИЦЫ СОПОСТАВЛЕНИЯ АМИНОКЛОТНЫХ ОСТАТКОВ 59

Рассмотрим эволюционное расстояние между двумя последовательностями 100 РАМ. На таком эволюционном расстоянии в среднем на 100 позиций произошло 100 событий от общего предка, стало быть при сравнении последовательностей у них должны быть отличия в 200 позициях из 100. Т.е. в каждой позиции произойдет по 2 события и эти последовательности будут совсем не похожи и не будут иметь ни одной совпадающей буквы. Но это совсем не так. Дело в том, что в некоторых позициях произойдет 5 событий, а в некоторых – 0 событий. Кроме того, в некоторых позициях произойдет цепочка событий, приводящих обе последовательности к одинаковым буквам. В результате около 40% символов в дочерних последовательностях совпадают. Кстати, на очень большом расстоянии при равновероятных остатках около 5% символов совпадет по случайным причинам.

3.6.2 Матрицы BLOSUM

Другой подход, предложенный в 1992г. Стивеном Хеникофф & Джорджией Хеникофф, основан совсем на другом принципе. Авторы разработали систему программ, которая в наборе родственных белков находит безделеционные мотивы. С использованием этих мотивов они строили множественное безделеционное выравнивание. Это выравнивание называется блоком. Была создана база данных, содержащая десятки тысяч таких блоков. Затем, основываясь на этой базе данных, были построены матрицы сопоставления аминокислотных остатков.

Для вычисления вероятностей $P(\alpha, \beta)$ мы в каждой колонке подсчитаем число всевозможных пар аминокислотных остатков (рис.3.11), потом просуммируем по всем колонкам и получим значения $n_{\alpha, \beta}$. Наконец, нормируем эти счетчики к 1 и получаем искомые оценки вероятностей:

$$P(\alpha, \beta) = \frac{n_{\alpha, \beta}}{\sum_{\gamma, \delta} n_{\gamma, \delta}}$$

Чтобы учесть разные уровни сходства, блоки прореживались так, чтобы

F	T	R	H	Y	T	E	RR=1
Y	S	R	H	W	T	D	RA=2
F	S	A	R	W	Q	S	RK=2
R	T	K	K	F	S	L	RH=2
W	V	H	L	Y	E	A	AK=1
							AH=1
							KN=1

Рис. 3.11: подсчет числа пар аминокислот в блоке

в них оставались последовательности с уровнем сходства не более заданного порога N%. Кроме того, из выборки удалялись идентичные последовательности. Учет разного уровня сходства приводит к построению серии матриц, называемых BLOSUM_N. Например, матрица BLOSUM80 соответствует последовательностям, имеющим уровень сходства 80% и менее,

матрица BLOSUM40 соответствует последовательностям схожим не более 40%. Наиболее популярной является матрица BLOSUM62, поскольку именно на этом уровне сходства получается наибольшее количество правильных выравниваний, соответствующих золотому стандарту.

Отметим, что в серии матриц BLOSUM чем больше номер, тем больше отношение диагональных элементов к не диагональным. В серии же PAM наоборот — чем больше номер, тем больше вклад недиагональных элементов.

3.7 Быстрые методы поиска сходства последовательностей

Выравнивание последовательностей достаточно тяжелая задача. Для выравнивания двух последовательностей требуется время порядка $T = O(m \cdot n)$, и для массовых задач типа поиска в банке последовательностей такой подход оказывается неприемлемым. В 90-е годы 20-века рассматривался технический подход к этой проблеме. Была попытка создать специализированный процессор, который мог бы быстро строить выравнивания. Однако эта идея не получила должного развития — специальный процессор оказался весьма дорогим. В настоящее время, наверное, можно построить алгоритм для построения выравниваний на графических процессорах. Но это пока не сделано. В большой степени это связано с тем, что эвристические подходы, основанные на идее хеширования, оказались достаточно эффективными.

Здесь мы будем обозначать Query последовательность запроса, а DB — банк данных с последовательностями, в которых будем искать гомологов Query. Основная идея подходов к поиску гомологов заключается в том, чтобы для DB создать индексные таблицы, которые облегчили бы поиск. Для построения индексных таблиц используются так называемые l-граммы — это короткие слова длиной l. Для каждого такого слова в таблицу записываются все адреса в банке последовательностей, где они встретились. Далее сканируется последовательность запроса и для каждого слова в Query по индексным таблицам выбираются вхождения в банк данных.

3.7.1 FASTA

Этот алгоритм впервые был предложен Вилбуром и Липманом в 1985 г. В основе этого алгоритма лежит идея поиска сильных диагоналей. Две пары позиций (i_1, j_1) и (i_2, j_2) принадлежат одной диагонали, если $i_1 - j_1 = i_2 - j_2 = t$, t — номер диагонали. При сравнении двух последовательностей мы сканируем первую последовательность, и с помощью индексной таблицы находим все совпадающие l-граммы, т.е. получаем набор пар позиций — одна позиция в Query, другая в DB (рис.3.12). Мощностью диагонали называется количество общих l-грамм, попавших на нее. Величина l обычно берется для аминокислотных последовательностей от 1 до 3 (чаще всего — 2), а для нуклеотидных последовательностей порядка 15.

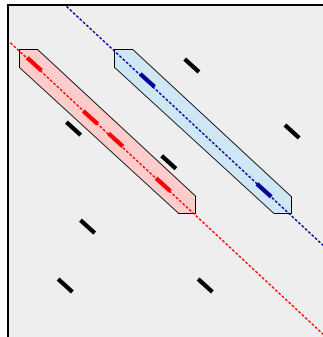


Рис. 3.12: Алгоритм FASTA

Далее, мы отбираем наиболее сильные диагонали и строим вокруг них выравнивание в полосе. Для определения статистической значимости мы вес выравнивания преобразуем в Z-score:

$$Z - score = \frac{W - \mu}{\sigma}$$

где μ , σ – среднее и стандартное отклонение, полученное при большом количестве сравнений случайных последовательностей. Предполагая нормальность распределения Z-score, можно оценить p-value. Программная реализация алгоритма получает на вход последовательности в определенном формате, в котором первая строка начинается с символа '>' и содержит заголовок, а остальные строки содержат собственно последовательность. При этом в строках допускаются пробелы. Файл может содержать несколько последовательностей. Этот формат записи получил широкое распространение и называется FASTA-форматом.

3.7.2 BLAST

Алгоритм BLAST также использует индексную таблицу. Но здесь вместо того, чтобы искать сильные диагонали, он пытается расширить общую l-грамму до тех пор, пока накопленный вес не станет отрицательным (рис.3.13). Такой расширенный сегмент называется HSP (high scoring segment). Для каждого HSP вычисляется его статистическая значимость, и, если она выше порога, то результат репортируется.

Оценка статистической значимости и e-value.

Распределение экстремальных значений Представим себе, что мы N раз сгенерировали значения некоторой случайной величины и получили набор из N чисел η_1, η_2, \dots , затем мы выбрали среди них максимальное значение $\eta^* = \max \eta_i$. Поскольку сделанная выборка – случайное событие,

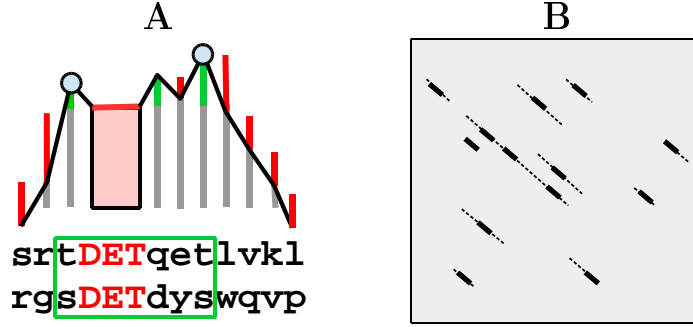


Рис. 3.13: Алгоритм BLAST. А – расширение затравки. В – набор расширенных затравок.

то и η^* – случайная величина. Как она распределена? Для того, чтобы получить это распределение вспомним определение кумулятивной функции распределения – это вероятность того, что значение случайной величины ξ не больше заданного значения x :

$$F(x) = \Pr(\xi \leq x)$$

Вероятность того, что при всех испытаниях значения η_i были не больше заданной величины x есть произведение вероятностей того, что каждая из них не больше x :

$$G(x) = \prod_i \Pr(\eta_i \leq x) = F^N(x)$$

Таким образом, мы получили распределение максимума. Если случайная величина ξ распределена экспоненциально с плотностью $f(x) = \lambda e^{-\lambda x}$, то ее кумулятивная функция распределения есть $F(x) = 1 - e^{-\lambda x}$. Можно показать, что плотность распределения максимума при N испытаниях (N достаточно велико) в этом случае имеет вид:

$$g(x) = \lambda e^{-\lambda x} \exp(-e^{-\lambda x}), \quad N \rightarrow \infty$$

Это распределение называется распределением Гамбела. Оказывается, что не только для экспоненциальных, но и для многих других типов исходных распределений распределение максимума также описывается распределением Гамбела.

Статистическая значимость выравниваний BLAST. Для оценки статистической значимости используется модель бернуллиевой генерации последовательностей. В этом случае можно показать, что при сравнении последовательностей длиной m и n , математическое ожидание количества бездеletionных выравниваний с весом не меньше S равно:

$$E(S) = K m n e^{-\lambda S}$$

3.7. БЫСТРЫЕ МЕТОДЫ ПОИСКА СХОДСТВА ПОСЛЕДОВАТЕЛЬНОСТЕЙ 63

где K, λ – константы, зависящие от матриц сопоставления остатков и частот встречаемости аминокислот. Величина $E(S)$ называется *e-value*. Вероятность того, что максимальный вес бездефекционного выравнивания двух последовательностей длиной m, n не меньше заданной величины S равна

$$\Pr(S \geq x) = 1 - e^{-E(S)}$$

При практическом использовании BLAST обычно смотрят на выравнивания с $e - value < 10^{-6}$. Казалось бы приемлемыми должны быть выравнивания с $e - value$ порядка 10^{-3} , однако они, как правило, не имеют достаточного биологического смысла. Надо понимать, что оценки были получены в двух, вообще говоря, неверных предположениях. Во-первых предполагалось, что все последовательности порождены моделью испытаний Бернулли, а во вторых, предполагалось, что все последовательности независимы. Поэтому оценки $e - value$, как правило, занижены и приходится использовать весьма серьезные пороги.

Развитие программы BLAST

Следующим этапом развития программ BLAST было расширение индексных таблиц. Для l -граммы XXX назовем T -соседями все l -граммы YYY такие, что суммарный вес сопоставления аминокислотных остатков в соответствии с матрицей замен не меньше порога T :

$$W(XXX, YYY) \geq T$$

При построении индексных таблиц теперь против каждой l -граммы будем писать адреса не только таких же l -грамм, но и всех T -соседей. Это приводит к увеличению размеров индексной таблицы, но, с другой стороны, увеличит чувствительность. Такой подход позволяет увеличить величину l , тем самым сократив объем вычислений. Кроме того, здесь применяется идея из алгоритма FASTA и процедура расширения затравок применяется только если затравки принадлежат сильной диагонали. Это расширение алгоритма позволило искать не только последовательности, но и мотивы. Поиск мотивов, в свою очередь, позволил создать программу PSI-BLAST, которая позволяет итеративно искать, например, семейства белков.

3.7.3 Выравнивание цепочек (Chain alignment)

Есть еще один «гибридный» метод выравнивания. Пусть, методом хеширования, мы нашли множество затравок

$$\Omega = \{(b_k^1, b_k^2, e_k^1, e_k^2)\}$$

где b_k^1, b_k^2 – координаты начал затравок на двух последовательностях, а e_k^1, e_k^2 – координаты концов затравок. Для каждой пары затравок $(b_i^1, b_i^2, e_i^1, e_i^2), (b_j^1, b_j^2, e_j^1, e_j^2)$ мы можем определить их совместимость и порядок следования:

$$(b_i^1, b_i^2, e_i^1, e_i^2) < (b_j^1, b_j^2, e_j^1, e_j^2) \Leftrightarrow b_i^1 < e_j^1 \ \& \ b_i^2 < e_j^2 \quad (3.12)$$

Иными словами одна затравка меньше другой, если по обоим последовательностям начало первой затравки меньше конца второй затравки. Отношение частичного порядка позволяет построить ориентированный граф – ребро между вершинами в графе проводится, если между ними есть отношение (3.12) (рис. 3.14). Добавим еще вершины начала и конца. Не на каждой вершине графа написан вес, равный мощности затравки. На ребрах можно написать вес, отражающий насколько диагональ одной затравки смещена относительно другой.

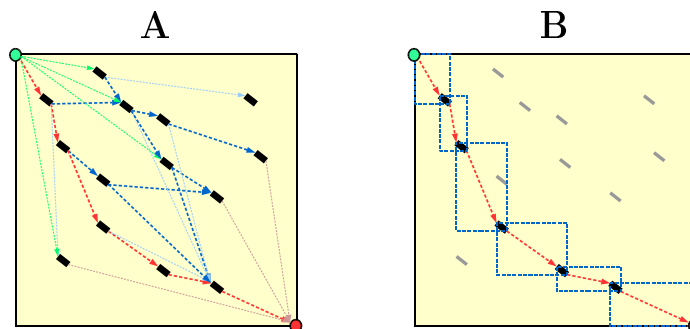


Рис. 3.14: Выравнивание цепочек. А – Граф затравок. Проведены не все ребра. В – Оптимальный путь.

Теперь мы можем поставить задачу – найти путь наибольшего веса в этом графе. Таким образом мы определим «якорные» затравки, через которые можно провести выравнивание. В больших пустых пространствах (если они есть) можно выделить прямоугольники. В них, если они достаточно большие, можно тоже поискать затравки с меньшим порогом и также построить внутренние пути. Потом, в оставшихся прямоугольниках можно сделать выравнивание Нидльмана-Вунша.

Такой подход называется выравниванием цепочек (chain alignment). В качестве затравок часто используют не просто l-граммы, а результаты BLAST. Он широко применяется для выравнивания целых геномов. В частности, известная программа *liftover*, переносящая координаты с одного генома на другой, основана на таком алгоритме.

3.7.4 Выравнивание геномов

При выравнивании геномов возникает несколько более сложная задача. ДНК – это двухцепочечная молекула и ее можно прочесть в разные стороны (прочитать разные цепи). Кроме того, в геномах бывают разные перестройки – кроме знакомых нам вставок-делеций бывают инверсии, когда целый фрагмент ДНК разворачивается (рис.3.15) и транслокации, когда фрагмент генома перемещается в другое место.

При выравнивании геномов можно по фрагментам получить отдельные диагонали, в том числе в противоположной ориентации. Далее, надо эти

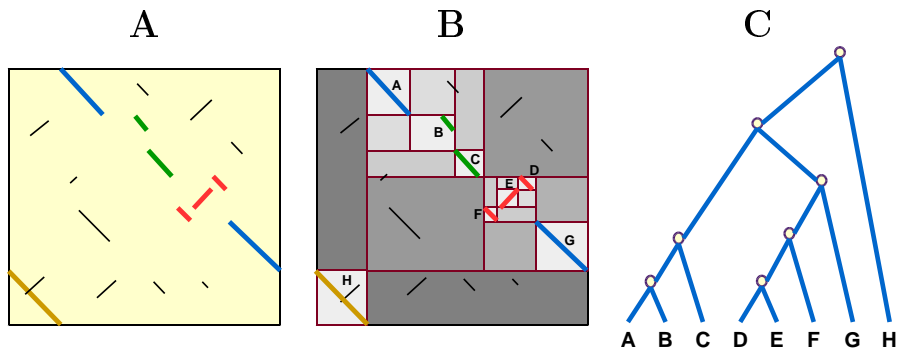


Рис. 3.15: А – Геномные перестройки бактериального кольцевого генома. Коричневая линия соответствует разным началам последовательностей, зеленые – вставки-делеции, красные – инверсии, черные – случайные совпадения. В – разложение выравнивания на прямоугольники. С – дерево разложения

диагонали собрать в общую карту. Если при обычном выравнивании мы всегда идем слева-направо, то здесь возможны перескоки, связанные с инверсиями. Для решения этих задач существуют специальные алгоритмы, обсуждение которых выходит за рамки курса. Одна из идей, лежащих в основе такого рода подходов заключается в том, что выравнивание с инверсиями представляется в виде набора вложенных прямоугольников. Этому набору соответствует некоторое дерево и задача представляется как задача построения оптимального дерева.

Глава 4

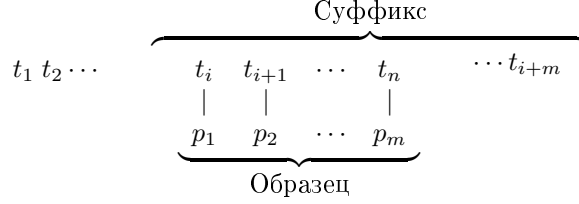
Преобразование Барроуза-Уиллера и картирование на геном

В последнее время с появлением мощных методов секвенирования стала более актуальной задача картирования фрагментов на геном. Для того, чтобы эффективно картировать последовательности на геном, удобно использовать предварительно подготовленный геном, например, геном, представленный в виде суффиксного дерева или суффиксного массива. Но размеры геномов эукариот весьма велики и не позволяют разместить эти конструкции в памяти компьютера. Поэтому необходимы новые структуры. Весьма популярными в настоящее время являются алгоритмы, связанные с преобразованием Барроуза-Уиллера. Эта глава является продолжением главы про алгоритмы на строках.

4.1 Суффиксный массив

Основной задачей при картировании является поиск образца (прочтения) в тексте (геноме). Есть две концепции такого поиска. Первая заключается в том, что сначала приходится подготовка образца. Например, построение конечного автомата или вычисление суффикс-префикс функции для образца, а потом сканирование текста (генома). Этот подход не подходит для нашей задачи, поскольку мы производим сканирование каждого образца. Другой подход – подготовка текста (генома) и достаточно быстрый поиск вхождения образца.

Одним из вариантов концепции – готовим текст и быстро ищем вхождения образца – является суффиксный массив (*SA*, Suffix array). Заметим, что образец P входит в текст тогда и только тогда, когда начало некоторого суффикса текста $T \{t_i, t_{i+1}, \dots, t_{i+m}\}$ совпадает с образцом:



Например образец `ccggtt` входит в текст `gataccggttatt` значит, что начало суффикса `ccggttattga` совпадает с образцом. Для того, чтобы быстро искать просто возьмем все суффиксы текста и отсортируем по алфавиту, например

1	gataccggttat\$	12	\$
2	ataccggttat\$	4	accggttat\$
3	taccggttat\$	10	at\$
4	accggttat\$	2	ataccggttat\$
5	ccggttat\$	5	ccggttat\$
6	cgttat\$	6	cgttat\$
7	gttat\$	1	gataccggttat\$
8	ttat\$	7	gttat\$
9	tat\$	11	t\$
10	at\$	3	taccggttat\$
11	t\$	9	tat\$
12	\$	8	ttat\$

На самом деле нет никакой необходимости держать в памяти компьютера все суффиксы, достаточно помнить сам текст и массив координат в левой колонке. Тогда для поиска образца можно применить бинарный поиск, при котором количество сравнений составляет порядка $\log n$. Но для каждого сравнения надо порядка m операций. Поэтому время работы поиска одного образца составит порядка $m \log n$. Если мы имеем порядка 100 миллионов прочтений по 100 нуклеотидов и нам надо картировать все прочтения, общее число операций будет порядка $n_{read} \cdot l_{read} \cdot \log L_{genome} = 10^8 \cdot 10^2 \cdot 30 = 3 \cdot 10^{11}$. Это довольно много. К тому же надо держать в памяти довольно большой суффиксный массив. Дальнейшее изложение посвящено современному алгоритму поиска всех вхождений образца. При этом мы избавимся от логарифма длины генома, что позволит сократить время работы в 30-60 раз, и сможем сократить требуемый объем памяти.

4.2 Концепция

Рассмотрим все циклические перестановки текста T (рис.4.1А). Упорядочим их лексикографически и выделим последнюю колонку в этой матрице. Слово, содержащееся в этой колонке и есть результат преобразования. Вся процедура очень похожа на концепцию суффиксного массива, только мы здесь работаем не с суффиксами, а с циклическими перестановками. Поскольку мы работаем с циклическими перестановками, то каждая стро-

ка устанавливает связь последнего символа с первым. Например, глядя на первый и последний элемент в строке 5 рис.4.1А, мы понимаем, что где-то в тексте есть пара букв `si`. После сортировки эта связь не теряется.

А		В		
1	<code>mississippi\$</code>	12	<code>\$mississippi</code>	<code>i</code>
2	<code>ississippi\$m</code>	11	<code>i\$mississipp</code>	<code>p</code>
3	<code>ssissippi\$mi</code>	8	<code>ippi\$mississ</code>	<code>s</code>
4	<code>sissippi\$mis</code>	5	<code>issippi\$miss</code>	<code>s</code>
5	<code>issippi\$miss</code>	2	<code>ississippi\$m</code>	<code>m</code>
6	<code>ssippi\$missi</code>	1	<code>mississippi\$</code>	<code>\$</code>
7	<code>sippi\$missis</code>	10	<code>pi\$mississip</code>	<code>p</code>
8	<code>ippi\$mississ</code>	9	<code>ppi\$mississi</code>	<code>i</code>
9	<code>ppi\$mississi</code>	7	<code>sippi\$missis</code>	<code>s</code>
10	<code>pi\$mississip</code>	4	<code>sissippi\$mis</code>	<code>s</code>
11	<code>i\$mississipp</code>	6	<code>ssippi\$missi</code>	<code>i</code>
12	<code>\$mississippi</code>	3	<code>ssissippi\$mi</code>	<code>i</code>

Рис. 4.1: **А** – циклические перестановки текста, **В** – лексикографически упорядоченные циклические перестановки текста. Последняя колонка матрицы – преобразование Барроуза-Уиллера.

Последняя колонка символов в упорядоченном массиве циклических перестановок называется преобразованием **Барроуза-Уиллера**. В приведенном примере результатом преобразования Барроуза-Уиллера строки `mississippi$` будет новая строка `ipssm$piissii`:

`mississippi$` → `ipssm$piissii`

Преобразование Барроуза-Уиллера обратимо, т.е. зная только результат преобразования (строку, например `ipssm$piissii`), мы можем восстановить исходную строку. Для этого есть несколько подходов – разной степени сложности алгоритма. Простейший алгоритм (рис. 4.2) состоит в следующем. В начале мы строим столбец из пустых строк. Затем последовательно повторяем две операции:

1. Добавить слева столбец BW
2. Отсортировать полученные слова

После первого этапа мы в столбце имеем упорядоченный набор всех букв в тексте. После второго этапа мы получаем набор всех пар букв, которые встречаются в тексте и во всех его циклических перестановках. На k шаге мы имеем полный набор k -грамм, которые встречаются в тексте и в его циклических перестановках. На последнем этапе мы получаем набор всех циклически переставленных вариантов текста, причем в первой строке будет исходный текст, который будет идти сразу после спец-символа `$`.

i		\$		i\$		\$m		i\$m		\$mi		i\$mi
p		i		pi		i\$		pi\$		i\$m		pi\$m
s		i		si		ip		sip		ipp		sipp
s		i		si		is		sis		iss		siss
m		i		mi		is		mis		iss		miss
\$	Sort	m	Insert BWT	\$m	Sort	mi	Insert BWT	\$mi	Sort	mis	Insert BWT	\$mis
p	⇒	p	⇒	pp	⇒	pi	⇒	ppi	⇒	pi\$	⇒	ppi\$
i		p		ip		pp		ipp		ppi		ippi
s		s		ss		si		ssi		sip		ssip
s		s		ss		si		ssi		sis		ssis
i		s		is		ss		iss		ssi		issi
i		s		is		ss		iss		ssi		issi

Рис. 4.2: Восстановление исходного текста

4.3 Быстрое восстановление исходного текста

Обозначим:

T – исходный текст (mississippi\$)

S – строка из первых символов в отсортированном массиве циклических суффиксов (\$iiiiimppssss)

BWT – строка из последних символов в отсортированном массиве циклических суффиксов (ipssm\$piissii)

Будем отслеживать номер появления символа в строке $Oc(S_i)$. Например, символ s на четвертой позиции BWT является вторым появлением символа s в строке BWT: $Oc(BWT_4) = 2$.

Лемма i- появление символа в последнем столбце указывает ровно на тот же текстовый символ, что i- появление в первом столбце, например первая буква p в правой колонке (2-я строка) и первая буква p в левой колонке (7-я строка) соответствует одной и той же второй букве p в основном тексте (эта буква подчеркнута на рис.4.1B, рис.4.3).

Отообразим номера появления символа в последней колонке на первую колонку. Для этого введем массив LF (last-first), в котором

$$LF[k] = i; \quad i : Oc(S_i) = Oc(BWT_k)$$

т.е. будем указывать номер строки, в которой номер появления символа в первой колонке равен номеру появления символа в текущей строке и в последней колонке (рис.4.4). Кроме того введем массив o номеров появления символа в строке BWT:

$$o[k] = Oc[BWT_k]$$

i - появление символа X в последнем столбце указывает ровно на тот же текстовый символ, что i -появление символа X в первом столбце	1	12	\$mississippi	i	1
	1	11	i\$mississipp	p	1
	2	8	ippi\$missis s	s	1
	3	5	issippi\$mis s	s	2
	4	2	ississippi\$m	m	1
	1	1	mississippi\$	\$	1
	1	10	pi\$mississip	p	2
	2	9	ppi\$mississi	i	2
	1	7	s ippi\$missi s	s	3
	2	4	s ississippi\$mi s	s	4
	3	6	s sippi\$missi	i	3
	4	3	s sissippi\$mi	i	4

mi s s i s s i p p i

Рис. 4.3: Порядок следования символа в левой колонке тот же, что и в правой. Разными цветами отмечены разные символы s . Видно, что порядок следования цветов одинаков слева и справа – оранжевый-желтый-голубой-фиолетовый. Этот порядок следования не обязан совпадать с порядком следования букв в исходном тексте.

Введем еще один дополнительный массив размеров в длину алфавита. $C[\alpha] = k$, где k – количество строк, предшествующих первому появлению α в первой позиции суффиксов. Иными словами, это номер строки, в которой впервые появился символ в префиксе минус 1. Длина массивов LF и O равна длине исходной строки.

C		LF		O
\$:0	\$BANANA	2	\$BANANA	1
A:1	ABANAN	6	ABANAN	1
	ANABAN	7	ANABAN	2
	ANANAB	5	ANANAB	1
B:4	BANANA	1	BANANA	1
N:5	NABANA	3	NABANA	2
	NANABA	4	NANABA	3

Рис. 4.4: Дополнительные массивы

Нетрудно догадаться, что если $o[i] = k$, то $LF[i] = C(BWT_i) + k$. Поэтому построить массив LF можно просто и эффективно. Отметим, что массив LF определяет предыдущий символ – просто потому, что все построение основано на циклических перестановках исходного текста. Отсюда – алгоритм восстановления:

	BWT	LF		T	r	c
\$BANANA	A	2	1		1	A
A\$BANAN	N	6	2	A	2	N
ANA\$BAN	N	7	3	NA	6	A
ANANA\$B	B	5	4	ANA	3	N
BANANA\$	\$	1	5	NANA	7	A
NAN\$BANA	A	3	6	ANANA	4	B
NANAN\$BA	A	4	7	BANANA	5	\$

Рис. 4.5: Пошаговый процесс восстановления строки. Слева: исходные данные: матрица Барроуза-Уиллера (нарисовано для сведения, на самом деле не используется), Строка BWT и массив LF. Справа: изменение параметров T – текст, r – номер позиции в BWT, c – текущий символ

```

1  T=""; r=1; c=BWT[r];
2  while(c!='$'){
3      T=cT;
4      r=LF[r];
5      c=BWT[r];
6  }
```

Пояснения. Строка 1 – инициация. Вначале встаем на первую строку и определяем последний символ строки, поскольку первая строка в суффиксном массиве отвечает единственному символу конца строки и в первой строке всегда лежит исходная строка с приписанным в начале \$. Потом циклически повторяем следующее: приписываем очередной символ в начало текста (строка 4), находим номер строки с предыдущим символом; определяем предыдущий символ. Если мы прочитали символ конца строки (\$), то цикл заканчивается. Покажем, как это работает (рис.4.5)

4.4 Поиск образца с помощью преобразования BWT

Найти все вхождения образца P в текст T эквивалентно тому, что найти в суффиксном массиве фрагмент, который отвечает суффиксам, начинающимся с P. На рис.4.6. Обозначим:

$F(W)$ – номер первого суффикса в суффиксном массиве, начало которого совпадает с образцом W

$L(W)$ – номер последнего суффикса, начинающегося с W .

Идея алгоритма поиска основана на простом наблюдении.

Введем еще один двумерный массив: $O[k, \alpha]$ – количество букв α выше позиции k в BWT. При этом если в позиции k есть буква α , то она также учитывается.

	#	SA	suff		BWT	\$	A	B	N
	1	7	\$BANANA		A	0	1	0	0
	2	6	A\$BANAN		N	0	1	0	1
$F('AN')=3$ →	3	4	ANA\$BAN		N	0	1	0	2
$L('AN')=4$ →	4	2	ANANA\$B		B	0	1	1	2
	5	1	BANANA\$		\$	1	1	1	2
	6	5	NA\$BANA		A	1	2	1	2
	7	3	NANA\$BA		A	1	3	1	2

Рис. 4.6: Слева: поиск всех вхождений образца эквивалентен поиску фрагмента суффиксного массива; Справа: Массив O – количество букв соответствующего типа выше заданной позиции

Лемма. Если мы знаем границы суффиксного массива для слова W мы можем легко вычислить границы для слова aW :

$$\begin{aligned} F(aW) &= C(a) + O(F(W) - 1, a) + 1 \\ L(aW) &= C(a) + O(L(W), a) \end{aligned} \quad (4.1)$$

Первая строка этого уравнения показывает, что: Во-первых, все слова aW начинаются там, где находятся все слова, начинающиеся с буквы a . Далее, мы знаем, что количество букв, находящееся над словом aW слева равно количеству букв a , предшествующих слову W в столбце BWT столбце, а это записано в массиве O . Поскольку в массиве $O(a)$ соответствующая счетчик увеличивается в момент появления буквы, то надо взять $O(F(W) - 1, a)$ и потом прибавить 1. Поэтому получаем формулу позиции в суффиксном массиве для первого вхождения слова aW : $F(aW) = C(a) + O(F(W) - 1, a) + 1$. Аналогично можно получить формулу для последнего вхождения $L(aW) = C(a) + O(L(W), a)$

Эта лемма позволяет индуктивно вычислять границы суффиксного массива, в пределах которых находятся адреса вхождений. Начинаем с конца слова, которого ищем. Для последнего символа вхождения начинаются с позиции $c(a)$ и кончаются в позиции $c(a + 1) - 1$, где $a + 1$ – следующий по алфавиту символ после a . Далее, добавляя слева по одному символу, пересчитываем начало и конец соответствующего суффикса образца. Пример поиска показан на рис.4.7

Вот псевдокод алгоритма поиска

```

1  a=P[m]; F=C[a]; L=C[a+1]-1 //a+1 - следующий символ в алфавите
2  for(i=m-1; i>=1; i--){
3      a=P[i];
4      F=C[a]+O[F-1,a]+1;
5      L=C[a]+O[L,a];
6  }
7  return (F,L);
```

Исходные данные						
#	SA		BWT	O		
				\$ (0)	A (1)	B (5)
1	11	\$ABABBAVABV	B	0	0	1
2	6	ABABV\$ABABV	B	0	0	2
3	1	ABABBAVABV\$	\$	1	0	2
4	8	ABV\$ABABVAB	B	1	0	3
5	3	ABVABABV\$AB	B	1	0	4
6	10	B\$ABABBAVAB	B	1	0	5
7	5	BAVABV\$ABAB	B	1	0	6
8	7	BAVV\$ABABVA	A	1	1	6
9	2	BAVBAVABV\$A	A	1	2	6
10	9	VB\$ABABBAVA	A	1	3	6
11	4	VBAVABV\$ABA	A	1	4	6

Изменения границ F и L по шагам

1	$F(A) = 2; L(A) = 5$	Изменения границ
2	$F(BA) = C(B) + O(1, B) + 1 = 5 + 1 + 1 = 7$ $L(BA) = C(B) + O(5, B) = 5 + 4 = 9$	
3	$F(ABA) = C(A) + O(6, A) + 1 = 1 + 0 + 1 = 2$ $L(ABA) = C(A) + O(9, B) = 1 + 2 = 3$	

суффиксного массива

1 \$ABABBAVABV	1 \$ABABBAVABV	1 \$ABABBAVABV
2 ABABV\$ABABV	2 ABABV\$ABABV	2 ABABV\$ABABV
3 ABABBAVABV\$	3 ABABBAVABV\$	3 ABABBAVABV\$
4 ABV\$ABABVAB	4 ABV\$ABABVAB	4 ABV\$ABABVAB
5 ABVABABV\$AB	5 ABVABABV\$AB	5 ABVABABV\$AB
6 B\$ABABBAVAB	6 B\$ABABBAVAB	6 B\$ABABBAVAB
7 BAVABV\$ABAB	7 BAVABV\$ABAB	7 BAVABV\$ABAB
8 BAVV\$ABABVA	8 BAVV\$ABABVA	8 BAVV\$ABABVA
9 BAVBAVABV\$A	9 BAVBAVABV\$A	9 BAVBAVABV\$A
10 VB\$ABABBAVA	10 VB\$ABABBAVA	10 VB\$ABABBAVA
11 VBAVABV\$ABA	11 VBAVABV\$ABA	11 VBAVABV\$ABA

Рис. 4.7: Процесс поиска паттерна АВА в тексте АВАВВАВВ. Вверху показаны Суффиксный массив, собственно суффиксы (нужны для справки), массивы BWT и O

Поиск происходит за время, линейное по длине паттерна, причем, в отличие от суффиксного дерева нет необходимости проходить дерево до конца, чтобы перечислить все вхождения — мы сразу получаем список позиций. Если паттерн отсутствует, то верхний индекс будет меньше нижнего.

4.5 Оптимизация памяти

Оценим память, необходимую для всех структур.

- L — для запоминания BWT
- $L \cdot 4$ — для запоминания суффиксного массива SA
- $L \cdot |\Sigma| \cdot 4$ — для запоминания массива O
- $|\Sigma| \cdot 4$ — для запоминания массива C

Таким образом, по размеру памяти использование BWT для поиска образцов ничем не лучше суффиксного дерева. Первое, что можно сделать — это не запоминать весь массив O , а запомнить только его часть. Например, давайте запомним значения этого массива в каждой n -ой позиции, а необходимые значения элементов массива вычислять. Для того, чтобы их вычислить, надо просмотреть часть строки BWT, в худшем случае — $n/2$ позиций. Пусть $n = 32$ и нам необходимо определить значение $O[i = 78, A]$. Тогда вычисляем $j_0 = i/32 = 2$; $j_1 = i \bmod 32 = 14$. Теперь можно взять $o = O[j_0 = 2, A]$ и пройти $j_1 = 14$ шагов вдоль строки BWT, начиная с позиции $i_0 = j_0 * n$, подсчитывая встраивающиеся буквы A. Если значение j_1 окажется больше половины n , то просмотр можно вести сверху, вычитая по 1 каждый раз, когда встретится буква A. Вот псевдокод этого вычисления.

```

1  CalcO(i,a){
2      j0=i/n; j1=i%n;      // здесь целочисленное деление
3      if(j1 <= n/2){
4          Oc=O[j0,a];
5          for(k=j0*n+1; k<=i; k++)
6              if(BWT[k] == a) Oc++;
7          return Oc;
8      }
9      else{
10         Oc=O[j0,a]
11         for(k=(j0+1)-1; k>=i; k--)
12             if(BWT[k] == a) Oc--;
13         return Oc;
14     }
15 }
```

Обычно используют шаг прореживания, равный $n = 128$. Это дает экономии памяти в 128 раз, но за это приходится платить временем — надо сделать до 64 шагов вдоль BWT. Для случая генома человека для запоминания массива O необходима память $L \cdot 4 \cdot 4/128 = 0.4Gb$. Здесь первая четверка — это размер алфавита, а вторая четверка — размер целого числа в байтах. Использование некоторых дополнительных приемов позволяет сократить размер необходимой памяти еще примерно в 2 раза. Если мы не

используем прореживание, то нам нет необходимости помнить саму строку BWT, однако при использовании прореживания мы используем эту строку. Поэтому ее тоже надо помнить.

В случае нуклеотидных последовательностей у нас 4-буквенный алфавит. Это значит, что каждый символ может быть закодирован двумя битами, тем самым сжав размер BWT в 4 раза. Геном человека в этом случае займет $0.75Gb$.

4.5.1 Программистские приемы

Здесь мои собственные представления, как можно эффективно сделать подсчет числа букв. Для того, чтобы вычислить значение элементов массива O в позициях не кратных 128 нет необходимости делать 64 шага, а достаточно сделать определенное небольшое количество побитовых операций, намного меньшее, чем 64. Сначала запишем массив BWT в виде двух битовых массивов – один будет отвечать за то, что нуклеотид в соответствующей позиции является пурином, а второй – за то, что он является «сильным» (рис.4.8). Теперь, используя такое представление, посчитаем количество букв 'а' в

seq		accgttga	gcatgatc	tggtgcag
SW		01110010	11001001	01101101
RY		10010011	10101100	01101011

Рис. 4.8: битовое представление последовательности. Чтобы узнать, какой нуклеотид находится в позиции, скажем, 3, смотрим на значение битов в этих массивах. Имеем: SW=1, RY=0, т.е. нуклеотид является «сильным» и пиримидином, значит, это 'с'

первой группе (в первом байте). Для этого построим битовый массив (байт), в котором 1 стоят в позициях, где буква 'а', т.е. позиции, где стоят пурины и при том "слабые" нуклеотиды. Это сделать просто с помощью побитовых операций, которые выполняются за одну команду. Сначала выделим позиции, где стоят слабые, т.е. позиции, где в первом массиве записаны 0 (w) и отметим позиции, где во втором массиве стоят 1, т.е. пурины (r). Побитовое логическое умножение этих двух битовых массивов (логическое "и") даст искомый результат.

$$\begin{aligned}
 w &= \neg SW[0] = (10001101) && \text{нам нужны позиции, где 0, поэтому отрицание} \\
 r &= RY[0] = (10010011) && \text{нам нужны позиции, где 1} \\
 a &= w \& r = (10000001)
 \end{aligned}$$

Теперь, чтобы посчитать количество букв 'а' до, скажем, 5-й позиции надо убить единичные биты после этой позиции, т.е. логически умножить на (11111000). Получим единицы только там, где нас интересует ((10000000)). Слова-маски – типа (11111000) для отбора нужных позиций можно затабулировать – таких масок будет 64. Осталось подсчитать количество ненулевых битов в слове (10000000). Для слов небольшой разрядности, например,

байтов, можно их просто затабулировать. Слова размерностью, например, 64 битовых разряда можно представить как 8 байтов и по очереди обратиться к соответствующему массиву и просто просуммировать количество ненулевых разрядов. На самом деле, можно хранить количество битов для всех двух-байтовых слов – это займет всего 64к. Тогда подсчет количества ненулевых битов сведется к 4 обращениям к массиву. Итого, для интерполяции значений массива O нам нужно выполнить до двух отрицаний, два логических умножения и до 4 обращений к массиву – т.е. практически мгновенно.

4.6 Сжатие суффиксного массива

Осталось найти способ экономно запомнить сам суффиксный массив. Для этого используется еще одна конструкция – функция последования Ψ . Если у нас есть суффикс $s_k, s_{k+1}, \dots, \$$ и у него есть позиция в суффиксном массиве i , то по определению $\Psi[i]$ – есть позиция следующего суффикса $s_{k+1}, \dots, \$$ (рис.4.9). Поскольку $SA[i]$ – позиция некоторого суффикса, то $SA[i] + 1$ – позиция следующего суффикса. Поэтому по определению:

$$SA(\Psi[i]) = SA(i) + 1$$

Функция Ψ эквивалентна суффиксному массиву – зная функцию Ψ можно

#	SA	Ψ	Ψ^{-1}	Suffix
1	11	3	6	\$ABABBBABBB
2	6	8	7	ABABBB\$ABAB
3	1	9	1	ABABBBABBB\$
4	8	10	8	ABBB\$ABABBB
5	3	11	9	ABBBABBB\$AB
6	10	1	10	B\$ABABBBAB
7	5	2	11	BABABBB\$AB
8	7	4	2	BABBB\$ABABBA
9	2	5	3	BABBBABBB\$A
10	9	6	4	BB\$ABABBBABA
11	4	7	5	BBABABBB\$ABA

Рис. 4.9: Функция последования Ψ

легко восстановить суффиксный массив. Она обладает рядом замечательных свойств. Например, она частично отсортирована. Поскольку все значения массива Ψ – разные, то можно определить обратную функцию Ψ^{-1} :

$$\Psi^{-1}[i] = x \Leftrightarrow i = \Psi[x]$$

Тогда, поскольку $SA[\Psi[i]] = SA(i) + 1$, то, обозначив $i = \Psi^{-1}(q)$ можно записать:

$$SA[q] = SA[\Psi^{-1}(q)] + 1$$

Применив это равенство несколько раз можно по части суффиксного массива восстановить его недостающую часть:

$$SA[k] = SA[(\Psi^{-1})^j(k)] + j \quad (4.2)$$

т.е. если мы знаем SA для некоторого значения, то мы можем вычислить значение SA для другого значения. Проблема в том, что вместо массива SA мы должны помнить массив Ψ^{-1} . Здесь опять приходит на помощь BWT. Мы можем вычислить значение Ψ^{-1} с помощью того, что у нас уже есть:

$$\Psi^{-1}(i) = C(a) + O(a, i); \quad a = BWT[i] \quad (4.3)$$

Теперь мы запомним в суффиксном массиве только каждый 32 элемент (это значит, что для суффиксного массива нам надо $L/8$ байт). Если нам надо получить элемент суффиксного массива для k не кратном 32 мы применяем функцию Ψ^{-1} до тех пор, пока не встретим доступное значение суффиксного массива:

```

1 Psi_1(k){
2     a=BWT[k]
3     return C[a] + CalcO(k,a);
4 }
5 GetSA(k){
6     j=0;
7     while(k%32 !=0){
8         k=Psi_1(k);
9         j++;
10    }
11    return SA(k)+j;
12 }
```

К суффиксному массиву мы обращаемся только в самом конце поиска и то, если мы найдем искомое вхождение. Если, к тому же, мы программе скажем, что неоднозначное картирование нас не устраивает, то мы только однажды для каждого фрагмента будем искать адрес по указанному алгоритму. Применяв эти приемы, мы, заплатив дополнительным временем, смогли сократить объем необходимой памяти. Для генома человека необходимая память составляет примерно $L/4 + L/8 + L/8 = L/2 = 1.5Gb$.

Поиск образца с заменами При поиске образца с заданным числом замен дерево замен (рис.4.10). В каждой позиции, кроме первой и последней, допускается замена, если только счетчик числа замен не исчерпан. При этом размер дерева может быть весьма большим, но, поскольку поиск происходит одновременно с построением дерева, есть большой шанс, что на некотором этапе очередная ветка засохнет, т.е. станет ясно, что значения границ в суффиксном массиве противоречивы: $L > U$. Многие современные программные реализации борются с проблемой замен разными способами.

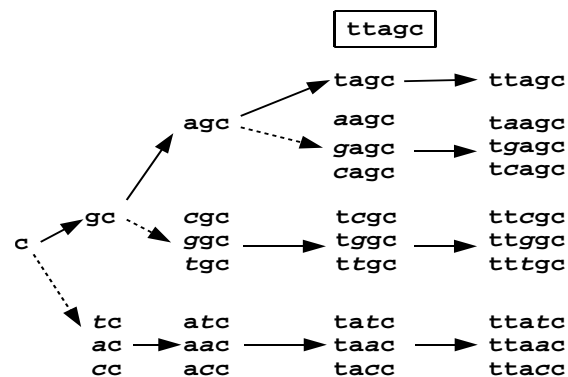


Рис. 4.10: дерево замен

Кроме проблемы замен есть еще проблема интронов. Ведь если мы картируем результаты РНК секвенирования, то в наших прочтениях как правило уже сплайсированные РНК с вырезанными интронами.

4.7 Итоги

Итак, общая схема алгоритма картирования прочтений на геном состоит из двух этапов.

Препроцессинг На первом этапе мы делаем подготовку генома. Она делается один раз и до смены релиза генома полученные массивы не меняем. Поэтому особенно упираться в скорость работы нет особой необходимости. В результате получаем:

- компрессированный массив BWT
- массив $C[\alpha]$
- прореженный массив $O[i, \alpha]$
- прореженный суффиксный массив SA

Эти подготовленные данные можно потом использовать многократно.

Поиск Поиск образца P начинаем с конца и с помощью уравнений (4.1) вычисляем границы в суффиксном (настоящем) массиве, где лежат соответствующие суффиксы. Продвигаясь по образцу к началу, границы перестраиваются и в конце концов находим интересующий нас участок суффиксного массива. Если количество образцов больше заданного порога, то репортируем множественное совпадение. Иначе с помощью уравнений (4.2,

4.3) вычисляем значения суффиксного массива. Ниже приведен псевдокод процедуры поиска.

```

1  a=P[m]; L=C[a]; U=C[a+1]    // a+1 -- следующий символ в алфавите
2  for(i=m-1; i>1; i--){
3      a=P[i];
4      L=C[a]+Calc_0(L-1,a)+1;
5      U=C[a]+Calc_0[U,a];
6      if(L > U) return 'not found';
7  }
8  if(U-L > N_Max) return 'too many matches'
9
10 for(i=L; i<=U; i++)
11     print GetSA(i);

```

Здесь мы обращаемся к описанным ранее функциям интерполяции массивов $\text{Calc}_0(i, a)$ и $\text{GetSA}(i)$. Время поиска можно оценить как

$$T(P) = O(|P| \times (T(\text{Calc}_0(i, a))) + T(v \cdot \text{GetSA}(i)))$$

где v — число вхождений образца. Вычисление значений $O(i, a)$ выполняется за время, не зависящее от длины образца и от длины текста $T(\text{Calc}_0(i, a)) = O(1)$. Интерполяция суффиксного массива $\text{GetSA}(i)$ не предсказуема, но в среднем составляет порядка $O(1)$. В итоге

$$T(P) = O(|P|) + O(v)$$

То, что время зависит от количества вхождений приводит к простому выводу — поиск с существенным ограничением на максимальное количество вхождений требует значительно меньшего времени.

Глава 5

Сборка геномов

5.1 Постановка задачи

В настоящее время для секвенирования геномов применяют метод дробовика (shotgun), суть которого сводится к тому, что весь геном фрагментируется с помощью ультразвука, после чего читаются фрагменты. Поскольку в эксперименте участвуют миллионы копий молекул, а каждая молекула рвется в случайных точках, то получается, что каждый локус на геноме перекрыт несколькими фрагментами. Используя перекрытия фрагментов можно попытаться восстановить полный геном.

Надо понимать, что большинство методов секвенирования не читают фрагмент полностью, а лишь его часть. Метод дробовика не привязан к технологии секвенирования, однако разные технологии дают во-первых прочтения разной длины, во-вторых, прочтения имеют разный уровень ошибок, и, наконец, в-третьих, дает разное количество прочтений. При секвенировании методом дробовика используют параметр – величина среднего покрытия – это суммарная длина всех прочтений, деленная на длину генома.

При секвенировании по Сенгеру характерная длина прочтения составляет до 1000 нуклеотидов, уровень ошибок $< 10^{-4}$. Характерное покрытие составляет порядка 5-10X. В настоящее время секвенирование по Сенгеру считается золотым стандартом и если необходимо подтвердить, например, какую-нибудь мутацию, применяют именно его. Однако этот метод дает слишком мало информации и достаточно дорогостоящий.

Методы второго поколения дают фрагменты порядка 100 нуклеотидов. Уровень ошибок составляет порядка $10^{-2} - 10^{-3}$. Характерный уровень покрытия составляет порядка 20-200X. Сейчас активно развиваются методы секвенирования третьего поколения. Для них характерны очень большие фрагменты, однако уровень ошибок все еще довольно высок.

Кроме геномов также есть задача секвенирования транскриптомов. Для случая, когда геном организма неизвестен, можно поставить задачу се-

квенирования транскриптома – и такая задача часто встречается. Обычно здесь используют обратную транскрипцию со случайных праймеров, предварительно отобрав поли-А фракцию. Здесь тоже получают большое количество фрагментов, которые намного короче, чем целевые последовательности.

Постановка задачи. Есть множество фрагментов последовательности, которые надо по перекрытиям собрать в единое целое – либо один геном, либо множество транскриптов. При этом ясно, что перекрытия должны быть достаточно большими, чтобы понизить вероятность объединения случайных фрагментов. С другой стороны, минимальный размер перекрытия не должен быть слишком большим, поскольку тогда мы не сможем объединить многие фрагменты.

Математическая постановка задачи. Идеальная постановка задачи звучит так. Дано множество слов. Построить последовательность минимальной длины такую, чтобы она содержала все подслова – задача о минимальной над-последовательности. Это довольно трудная задача. Для реальности применяют разные эвристические подходы, которые решают эту задачу приближенно. Кроме того, эта задача не всегда отражает реальность. Допустим есть три слова `accgattca`, `agctgatgcg`, `cgtttaaaa`. Эти слова зацепляются только одним-двумя крайними буквами и минимальная над-последовательность не имеет особого биологического смысла.

5.1.1 Дополнительные трудности

ДНК – двунитевая!

ДНК является двунитевой молекулой но при секвенировании читается одна нить, и мы не знаем какая. Поэтому в массиве прочтений есть фрагменты, прочитанные в разных направлениях. При описании подходов к сборке геномов мы будем предполагать, что читается только одна нить. Для учета того, что в прочтениях есть две нити можно каждое прочтение продублировать своим комплементом и попытаться собрать независимо обе нити. Однако есть и более умные подходы, которые мы здесь не рассматриваем.

Неравномерность покрытия

Под покрытием секвенирования обычно понимают среднее покрытие. Фрагменты, которые получаются при методе дробовика образуются и секвенируются являются случайными и по случайным причинам на некоторые участки генома попадает много фрагментов, в то время как другие участки могут быть не покрыты совсем. Поэтому часто черновая сборка генома может содержать значительное количество дырок.

Ошибки секвенирования

Прочтения, получаемые в результате секвенирования, являются экспериментальными результатами и могут содержать случайные ошибки. При построении алгоритмов сборки геномов необходимо принимать во внимание возможность ошибок в прочтениях.

Повторы

Если у нас прочтения достаточно короткие, то возникает проблема повторов. Она заключается в следующем. Если в геноме есть достаточно длинный повтор, длина которого больше длины прочтения, то принципиально не возможно однозначно восстановить полную последовательность. Все участки между повторяющимися элементами можно расположить в любой последовательности и при этом все наши наблюдения будут полностью объяснены. Действительно, зеленый фрагмент на рис.5.1 имеет перекрытия

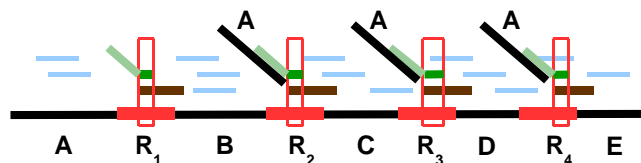


Рис. 5.1: Проблема повторов

со всеми коричневыми фрагментами, поскольку эти перекрытия принадлежат повтору, и, следовательно их последовательности идентичны. Поэтому перекрытия, выделенные красными прямоугольниками являются вполне допустимыми. Таким образом, после фрагмента *A* могут идти фрагменты *RB*, *RC*, *RD*, *RE*. Следовательно, возможны сборки *ARBRCRDRE* и, например, *ARDRBRCRE*. Надо отметить, что большинство эукариотических геномов содержат большое количество повторов. В частности, в геноме человека есть около миллиона копий *Alu*-повторов что составляет около 10.7% от всего генома. Кроме таких повторов есть еще повторы *LINE*, псевдогены и много других.

Полиморфизмы

Другой проблемой, с которой приходится сталкиваться — это полиморфизм. Многие геномы в нормальном состоянии диплоидные, т.е. в каждой клетке есть отцовская и материнская хромосомы, и они не идентичны. Количество простых замен (однонуклеотидных полиморфизмов) в геноме человека составляет около 0.1%, что говорит о молодости вида. Кроме того, есть еще вставки, делеции и геномные перестройки, которые особенно часто встречаются в раковых клетках. В более древних видах уровень полиморфности может достигать 1% (дрозофила, мышь) и даже 10% (оболочечник сиона).

В таких условиях сборка генома по перекрытиям сталкивается с проблемой неточного перекрытия. Можно рассматривать задачу сборки отцовского и материнского генома и определения гаплотипов. Однако при небольшом уровне полиморфности задача восстановления гаплотипов становится достаточно трудной. При секвенировании бактериальных геномов проблемы с полиморфностью генома не возникает.

Полиплоидия

Многие, в основном растительные, геномы обладают еще одной особенностью. У них есть дубликации целых хромосом. Например, геном культурной пшеницы является гексаплоидом. Полиплоидия сродни повторам, только размер повторов составляет целые хромосомы. В полиплоидных геномах дублицированные хромосомы не совсем идентичны, поэтому сборка полиплоидных геномов представляет большую трудность. Заметим, что геном человека также в своей истории прошел через, как минимум, две дубликации – первая в начале эволюции позвоночных, а вторая – при переходе от бесчелюстных к челюстноротым. На это указывает анализ геномного состава геномов. Однако эти события произошли давно, накопилось большое количество мутаций, и такая полиплоидия не представляет проблемы для сборки геномов. В культурных растениях полиплоидия была зафиксирована искусственной селекцией и произошла совсем недавно, так что количество замен не велико.

5.1.2 Основные термины

Прочтения. Это последовательности, полученные из секвенатора и прошедшие предварительную обработку, включающую в себя удаление праймеров и адаптеров, а также очищенные по качеству. Дело в том, что нуклеотиды, прочитанные ближе к концу прочитаны менее надежно. Секвенатор выдает качество прочтения нуклеотидов, например, зашифрованное в файлах FEASTQ.

Контиг. Контигом называют последовательность перекрывающихся прочтений. В настоящее время обычно контиг и последовательность контига используются как синонимы.

Скаффолд. Это последовательность контигов. При этом между контигами могут быть пропуски – не расшифрованные участки, часто неизвестной длины. Контиги можно упорядочить с использованием дополнительных экспериментальных процедур.

Качество сборки. По разным причинам сборка бывает не полной. Это может быть связано как с большим количеством повторов, высокой полиморфностью, так и с недостаточным покрытием. Есть несколько схожих

характеристик качества сборки. Определим L – суммарная длина всех контигов, G – предполагаемая длина генома, C_i – длины контигов, упорядоченные по убыванию. Основные характеристики качества сборки представлены в таблице 5.1.2.

Таблица 5.1: Основные характеристики качества сборки генома

N50	<p>Длина самого короткого контига такого, что суммарная длина более длинных контигов не меньше половины длины сборки L. Можно воспринимать N50 как центр масс распределения длин контигов.</p> $N50 = C_k, k = \arg \max_s \sum_{i \leq s} C_i \leq 0.5 \cdot L$
L50	<p>Число контигов контига такое, что суммарная длина более длинных контигов не меньше, чем суммарная длина более коротких контигов.</p> $L50 = \arg \max_s \sum_{i \leq s} C_i \leq 0.5 \cdot L$
N90	<p>Длина самого короткого контига такого, что суммарная длина более длинных контигов не меньше, чем 90% от суммарной длины сборки L:</p> $L50 = \arg \max_s \sum_{i \leq s} C_i \leq 0.9 \cdot L$
NG50	<p>Длина самого короткого контига такого, что суммарная длина более длинных контигов не меньше, чем половина размера генома.</p> $L50 = \arg \max_s \sum_{i \leq s} C_i \leq 0.5 \cdot G$

5.2 Графовая постановка задачи. Гамильтонов путь

Пусть у нас есть некоторое количество прочтений. Мы можем каждое прочтение сравнить с каждым и найти перекрытия. Если у нас, скажем, 10^6

прочтений (что очень немного), то необходимо проверить на наличие перекрытий $10^6 \times 10^6 = 10^{12}$ пар прочтений, а это уже совсем не простая задача. Для этого есть целый ряд эвристических алгоритмов. Далее, можно построить граф: вершины графа соответствуют прочтениям, а ребра проводятся, если между прочтениями есть перекрытие заданного размера, причем если суффикс вершины A совпадает с префиксом вершины B , то ребро ориентировано $A \rightarrow B$. Рассмотрим игрушечный пример. Пусть у нас есть следующие прочтения:

аacc, accg, ccga, ccgt, cgaa, cgat, cgta, gaac, gtac, tacc

И пусть мы поставим порог 2 совпадающие буквы на перекрытии. Тогда граф перекрытий будет иметь вид (рис.5.2) На этом графе есть две вер-

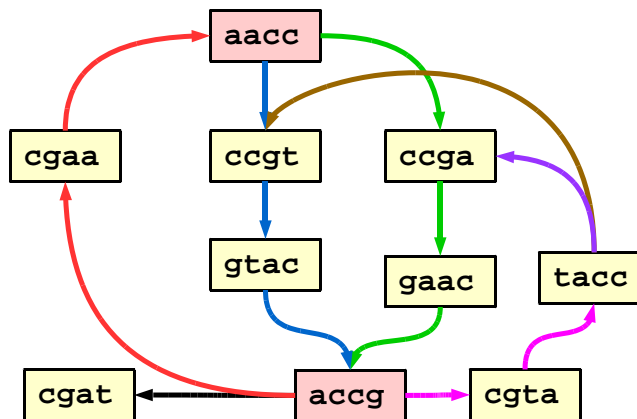


Рис. 5.2: Граф перекрытий

шины, которые имеют неоднозначное продолжение (выделены красным). Однозначное прочтение последовательности не возможно. Эта неоднозначность говорит о том, что в последовательности есть повтор. Расшифровка последовательности в этом случае заключается в том, чтобы найти однозначные фрагменты. В нашем случае это будет 4 фрагмента:

accg aacc	красный
accg at	черный
aacc gt accg	голубой
accg taccga accg	пурпурный+фиолетовый+зеленый
accg taccgt accg	пурпурный+зеленый
aacc ga accg	зеленый

Объединяя фрагменты по повторам получим разные варианты прочтений. Поскольку при восстановлении последовательности нам надо обойти все вершины, то этот подход по аналогии называют Гамильтоновым, хотя это не совсем адекватно, поскольку восстановление однозначных путей в этом графе не является задачей Гамильтона и решается простым обходом графа.

5.3 Графовая постановка задачи.

Граф де-Брюина

Альтернативный подход основан тоже на построении графов, только ребра соответствуют не перекрытиям, а прочтениям, а вершины соответствуют перекрытиям. Если дан словарь, то граф, в котором ребра соответствуют словам, а вершины – суффиксам-префиксам слов называется графом де-Брюина. Алгоритм заключается в двух этапах. Если задан минимальный размер перекрытия k , то строится словарь из всех слов размером $k + 1$, которые встречаются во всех прочтениях. Характерный размер $k = 30-50$ букв. По этому словарю строится граф де-Брюина. Заметим, что для построения такого графа нет необходимости искать перекрытия. С другой стороны, при построении такого графа мы теряем часть информации, поскольку все прочтения рассыпаны на слова, которые, как правило, значительно короче прочтений. По заданному набору прочтений строим граф де-Брюина (рис.5.3А). Далее, на графе де-Брюина цепочки вершин без ветвле-

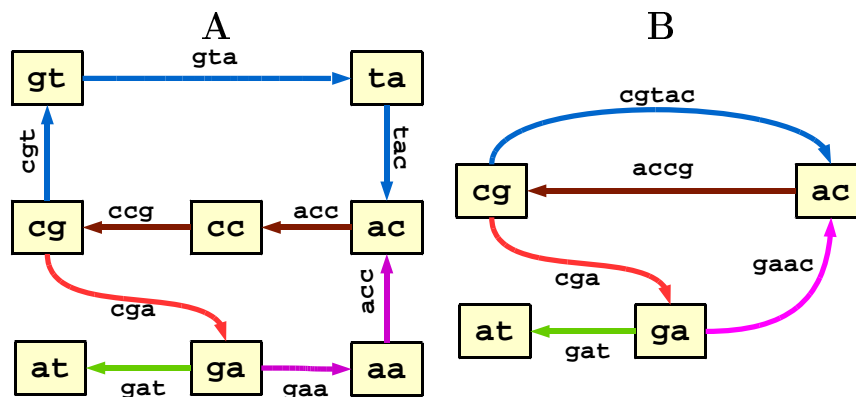


Рис. 5.3: Граф де-Брюина. А – полный граф де-Брюина; В – упрощенный граф.

ний объединяются в одно ребро. Получается упрощенный граф де-Брюина (рис.5.3В). Каждая такая цепочка представляет собой контиг. На этом игрушечном примере все контиги, кроме одного (cgtac), тривиальны и даже не превышают размер прочтений. Однако в реальном мире размер графа намного больше и размер перекрытия, и, стало быть, размер слова намного больше. Поэтому в результате упрощения графа получаются достаточно большие контиги. После упрощения графа де-Брюина идет прочистка ребер, а именно анализируются тупиковые ребра, и другие особенности упрощенного графа, например, анализируются "пузыри" выпячивания и др. Словарный подход к сборке генома, основанный на графе де-Брюина имеет еще преимущества. Во-первых, нет необходимости в качестве отдельной процедуры искать перекрытия, поскольку перекрытия уже определены

словарем. Второе преимущество – прочистка от ошибок прочтения. Современные методы секвенирования дают очень большое покрытие. Поэтому правильные слова в словаре должны встречаться много раз. Слова же содержащие ошибки будут уникальными и их можно просто отбросить.

5.4 Дополнительная информация и построение скаффолдов

Для того, чтобы упорядочить контиги, привлекают дополнительные экспериментальные данные, связанные с тем, что фрагменты ДНК можно читать с двух концов, при этом можно сохранить информацию о том, какие прочтения принадлежат одной паре. Эта технология называется парно-концевым чтением. Одним из первых шагов при секвенировании методом дробовика является фрагментирование ДНК. Это обычно делается с помощью ультразвука, который физически разрывает ДНК в случайных местах. Обработка ультразвуком разрывает ДНК в произвольных местах, в результате образуется множество фрагментов. Размеры фрагментов распределены примерно нормально. При разных параметрах обработки получаются фрагменты разной характерной длины. Обычно средний размер фрагментов порядка 200-2000 нуклеотидов.

Чтобы получать пары прочтений, разнесенные на большее расстояние, применяют технологию mate pair (рис.5.4). Суть метода заключается в том, что сначала ДНК фрагментируется на большие сегменты, например, с помощью редко-щепящих и супер-редко-щепящих рестриктаз (рис.5.4А). В результате получаются фрагменты в килобазы или в десятки килобаз. Далее фрагменты закольцовывают через биотенилированную вставку (рис.5.4В). Далее кольца разрывают с помощью мелкощепящих рестриктаз (рис.5.4С), отбирают биотенилированные фрагменты и секвенируют фрагменты с двух концов (рис.5.4D). В результате получают пары прочтений, разнесенные на достаточно большое но не известное точно расстояние. С помощью данных,

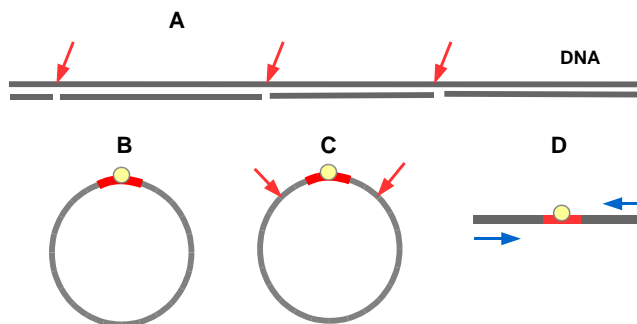


Рис. 5.4: Технология mate pair

5.4. ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ И ПОСТРОЕНИЕ СКАФФОЛДОВ89

полученных с помощью парно-концевого прочтения и с помощью метода mate pair можно попытаться восстановить примерное взаимное расположение контигов. Для этого тоже строят граф, вершины которого соответству-

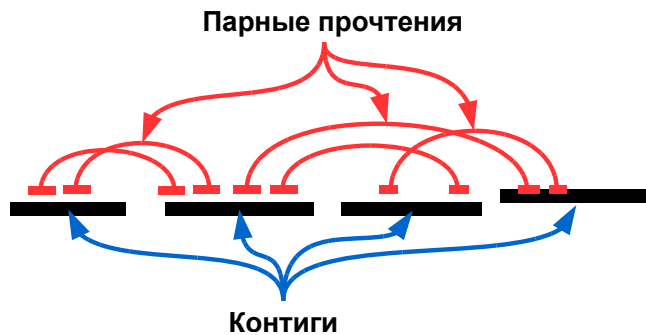


Рис. 5.5: Построение скаффолдов

ют контигам, а ребра — парам прочтений. Вес ребра оценивается как количество парных прочтений, соединяющих контиги. Основная идея заключается в том, чтобы в этом графе найти путь максимального веса. При этом путь не должен проходить через одну вершину дважды. Последнее ограничение делает задачу NP-полной. Поэтому для ее решения применяют различные приближенные эвристики, в частности жадные стратегии.

Глава 6

Скрытые марковские модели

6.1 Вводные замечания

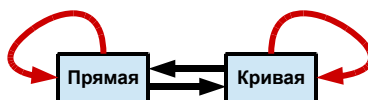
Мы ранее рассматривали модели последовательностей, например, бернуллиевская или марковская. В таком рассмотрении есть один существенный недостаток — последовательности предполагались однородными. Однако, очевидно, что в биологии это не так. Например в геноме есть, по крайней мере, кодирующие и некодирующие области, и их вероятностные свойства сильно различаются. Поэтому надо рассматривать неоднородные модели, в которых предполагается наличие разных областей с разными вероятностными характеристиками. Одним из способов построения таких моделей являются скрытые марковские модели (Hidden Markov Model, HMM).

6.1.1 Пример

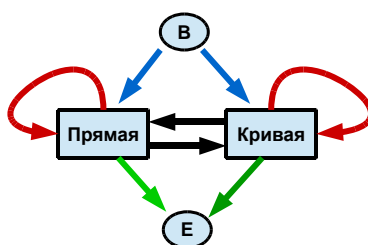
Для того, чтобы разобраться с этим, вернемся к основному объекту теории вероятностей — монете. Пусть у кого-то есть две монеты разной кривизны — одна правильная, а другая — кривая. Этот некто бросает монету много раз, время от времени заменяя монету. Нам предъявляется серия результатов. Наша задача — определить, когда использовалась правильная монета, а когда — перекошенная в сторону, скажем орла. Вот результаты испытаний:

```
0011010001010011110101111111011111101101001001000100100110
100101101100010100010001011110101101011110101111011111010111
```

Для того, чтобы полностью определить задачу нам необходимо задать вероятностные характеристики, а именно, вероятности выпадения орла или решки для каждой монеты и вероятности замены монеты. Более формально, мы можем описать процесс в виде конечного автомата, в котором состояния отвечают типу монеты.



На этой схеме блоки (будем называть состоянием) отвечают типам монеты, стрелки отвечают переходам между состояниями. Черные стрелки – монеты заменяются, а красные соответствуют случаю, когда на очередном шаге не происходит подмены. Однако, последовательность состояний с чего-то начинается и чем-то кончается. Поэтому более полная схема должна содержать состояния начала (В) и конца (Е):



На этой схеме синие стрелки означают, что последовательность наблюдений с чего-то начинается (состояние В). Зеленые стрелки – конец последовательности испытаний. Для полного определения модели нам необходимо определить свойства монет (вероятности исхода одного броска) и вероятности переходов. Таким образом схема становится такой: В биоинформатике мы

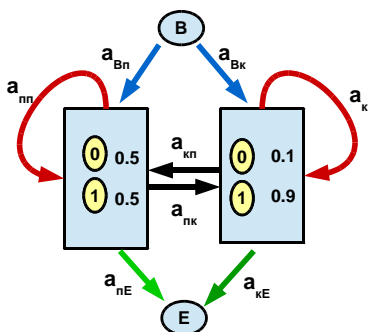


Рис. 6.1: Простейшая модель

имеем дело с символьными последовательностями. Поэтому картина выглядит следующим образом: Нечто (Природа) порождает последовательность символ за символом. В какой-то момент начинается последовательность другого типа, например, была глобулярная часть белка, потом начинается трансмембранная. Вероятности порождения символов в разных частях – разная.

6.2 Скрытые Марковские Модели

Определение. Скрытой Марковской Моделью называется:

$$\{A, Q, B \in Q, E \in Q, a_{kl}, e_k(\alpha)\},$$

где

A – алфавит;

Q – набор состояний;

B – стартовое состояние;

E – финальное состояние;

a_{kl} – набор *переходных* вероятностей из состояния k в состояние l ;

$e_k(\alpha), \alpha \in A$ – набор *эмиссионных* вероятностей, т.е. вероятностей порождения символа α в состоянии k

Поскольку из любого состояния есть выход, то есть ограничения на переходные вероятности.

$$\sum_l a_{kl} = 1 \quad (6.1)$$

Не забываем, что случай “остаться в состоянии k ” является частным случаем перехода из состояния k в состояние k . На эмиссионные вероятности также есть ограничение – в любом состоянии (кроме B, E) происходит обязательно порождение символа. Поэтому

$$\sum_{\alpha} e_k(\alpha) = 1 \quad (6.2)$$

6.2.1 НММ – Генеративная модель

Скрытые марковские модели, равно как и множество других моделей (бернуллиевские, однородные марковские) – это модели порождения (генерации) последовательностей. Действительно, мы можем с использованием датчика случайных чисел порождать последовательности, отвечающие модели. Порождение последовательности происходит следующим образом. Сначала мы находимся в состоянии B . Из этого состояния выходит несколько стрелок, на которых написаны вероятности a_{Bk} .

Мы знаем, что сумма вероятностей равна 1. Поэтому интервал $[0,1]$ можно представить как сумму отрезков, каждый из которых имеет длину a_{Bk} :

Мы разыгрываем случайную величину, равномерно распределенную на отрезке $[0,1]$ и смотрим, в какой интервал попали. Осуществляем переход в то состояние, которое было выбрано таким образом. Далее, точно также, но с использованием эмиссионных вероятностей выбираем какую букву мы генерируем. Теперь мы находимся в новом состоянии. Точно также выбираем переход в новое состояние и генерируем очередную букву. Так до тех пор, пока не попадем в состояние E .

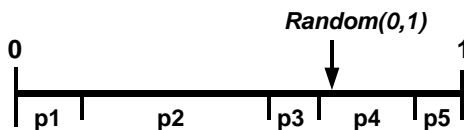


Рис. 6.2: выбор события

```

Select(p[]){ // Выбираем событие; pi -- массив вероятностей
    r=random(0,1);
    ll=0;
    for(i=0; i<length(p); i++){
        ll=ll+p[i]
        if(r < ll) return i
    }
}

Ggenetator{ // Генерация последовательности
    curState=Beg // начинаем со стартового состояния
    while(){
        curState=Select(a[curStae]) // Переходим в новое состояние
        if(curState == End) break; // дошли до конца
        c = Select(e[curState]) // выбираем символ
        print(a[c]) // печатаем символ
    }
}

```

Задача генерации последовательностей имеет ограниченный интерес. Единственное, что можно сделать – это породить любое количество последовательностей, отвечающих представленной модели. Однако скрытые марковские модели можно использовать и для разметки последовательностей – т.е. можно поставить такую задачу. Дана НММ и последовательность наблюдений (например, последовательность генома). Разметить последовательность, т.е. сказать, какие участки последовательности порождены какими состояниями (например, где находятся гены). Такие задачи называются задачами декодирования.

6.2.2 Граф состояний

Рассмотрим задачу о бросках двух монет рис.6.1. Пусть дана последовательность наблюдений: 00101110111101011001 Тогда мы получаем граф, любой путь в котором отвечает возможному декодированию (рис.6.3А). Например, последовательности состояний В-П-П-П-К-К-К-...-К-П-Е соответствует выделенный путь (рис.6.3В) Таким образом, во-первых, по последовательности мы можем построить граф: вершины соответствуют паре позиция-состояние: позиция слева направо, состояние сверху-вниз. Во-вторых, каждому пути на этом графе соответствует последовательность со-

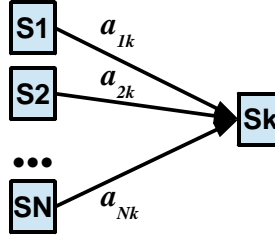
где $X = \{x_i\}$ последовательность наблюдений. Для состояния E мы считаем эмиссионную вероятность равной 1. Тогда задачу декодирования можно сформулировать так: найти путь, который максимизирует правдоподобие:

$$\Pr(\Pi) \rightarrow \max_{\Pi}$$

Для решения этой задачи применим динамическое программирование. В каждой вершине графа определим число $v_k(i)$ – максимальное правдоподобие пути от начала B до этой вершины. Ясно, что для вершины B оно равно 1:

$$v_B(0) = 1$$

Чтобы найти максимальное правдоподобие $v_k(i)$ в вершине $V_k(i)$ посмотрим как в нее можно попасть:



чтобы максимизировать правдоподобие просто переберем все предыдущие вершины:

$$v_k(i) = \max_l v_l(i-1) a_{lk} e_k(x_i)$$

таким образом, начиная от вершины B и заканчивая вершиной E получим значения максимального правдоподобия на каждой вершине. Чтобы определить оптимальный путь мы, как обычно, будем запоминать оптимальные переходы:

$$\pi_k(i) = \arg \max_l v_l(i-1) a_{lk} e_k(x_i)$$

Здесь $\pi_k(i)$ – оптимальные предшественник для состояния k в позиции i . Таким образом получаем окончательно рекурсию

$$\begin{aligned}
 v_B(0) &= 1 \\
 v_k(i) &= \max_l v_l(i-1) a_{lk} e_k(x_i) \\
 v_E(L+1) &= \max_l v_l(i-1) a_{lE} \\
 \pi_k(i) &= \arg \max_l v_l(i-1) a_{lk} e_k(x_i) \\
 \pi(E) &= \arg \max_l v_l(n) a_{lE}
 \end{aligned} \tag{6.4}$$

В результате работы алгоритма мы получим набор оптимальных переходов $\pi(i)$ (рис.6.4, толстые стрелки). В каждую вершину входит ровно одна (оптимальная) стрелка. Таким образом, мы получаем дерево оптимальных

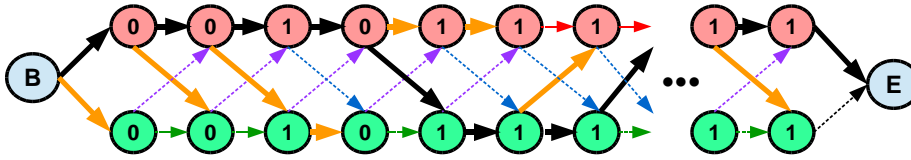


Рис. 6.4: Восстановление оптимального пути. Толстые стрелки – переходы, отмеченные в $\pi_k(i)$. Черный – оптимальный путь.

переходов. Для того, чтобы восстановить оптимальный путь, мы, начинаем с конечного состояния. Смотрим, из какого состояния оптимальный путь приводит в конец. Переходим в это состояние и запоминаем его. Далее, смотрим как мы попали в это состояние, запоминаем его и переходим в предыдущее состояние. И так до тех пор, пока не придем в начало. Псевдокод обратного прохода:

```
// Определяем состояние для последней позиции
st=B; max=0;
for l in states{
    if(v[l]*a[l,e] > max) st=l
}
// рекурсия для восстановления состояний
for(i=n-1; i>0; i=i-1){
    states[i]=st;    //запоминаем текущее состояние
    st=pi[st][i];    //определяем новое состояние
}
```

Весьма распространенной ошибкой среди студентов является игнорирование построения обратного прохода и просто выбор пути по максимальному значению переменной $v_k(i)$: $\pi^*(i) = \arg \max v_k(i)$. **Это категорически неверно!** Предлагаю придумать контрпример, демонстрирующий, что так делать нельзя.

Рекурсия (6.4) называется алгоритмом **Витерби**. Суть его (как и любого динамического программирования) заключается в проходе вперед с вычислением оптимальных значение и проходе назад с восстановлением пути.

6.2.4 Апостериорное декодирование. Алгоритм Вперед-назад (FB)

В предыдущей секции мы определили наилучший путь. Но при этом мы не знаем насколько он достоверен. Например, оптимальный путь проходит в позиции i через состояние S_k . Но может быть путь, проходящий в этой позиции через другое состояние не намного хуже? Поэтому можно поставить другую задачу – вычислить вероятность состояния S_k в позиции i

$(S_k(i))$. Вероятность состояния можно определить, как сумму по всем путям правдоподобий пути при заданной последовательности (рис.6.5): Сумма по

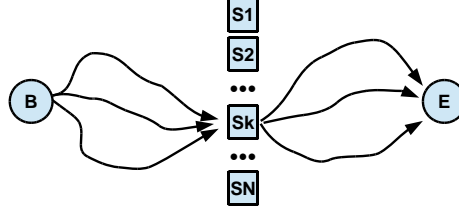


Рис. 6.5: Сумма по всем путям, проходящим через вершину $S_k(i)$

путям:

$$\Pr(S_k(i)|x) = \sum_{path \in \text{all } \Pi: \pi(i)=S_k} \Pr(path|x) \quad (6.5)$$

По формуле Байеса пишем:

$$\Pr(S_k(i)|x) = \frac{\Pr(S_k(i), x)}{\Pr(x)} \quad (6.6)$$

Вероятность $\Pr(S_k(i), x)$ можно представить как произведение вероятности прийти из начала B в $S_k(i)$ и породить последовательность $x_1 \dots x_i$ и вероятности уйти из $S_k(i)$ в конец E :

$$\begin{aligned} \Pr(S_k(i), x) &= \Pr(x_1 \dots x_i, \pi(i) = S_k) \cdot \Pr(x_{i+1} \dots x_L | \pi(i) = S_k) \\ &= f_k(i) \cdot b_k(i) \end{aligned} \quad (6.7)$$

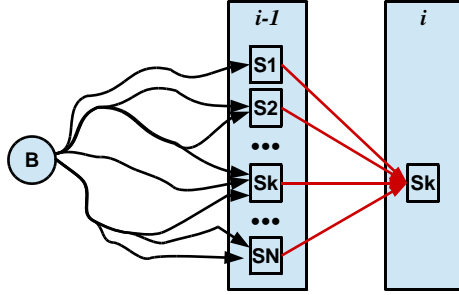
Первая вероятность есть сумма по всем путям от начала B до $S_k(i)$. Её значение в начале равна 1, поскольку сумма вероятностей по всем путям от начала в начало равна 1, и там есть единственный тривиальный путь (ни одного ребра): $f_b(0) = 1$. Если мы знаем суммы по всем путям для всех состояний в предыдущей позиции последовательности, то сумму по всем путям для $S_k(i)$ можно вычислить рекурсивно (рис.6.6):

$$f_k(i) = \sum_l f_l(i-1) \cdot a_{lk} \cdot e_k(x_i)$$

Наконец, полная вероятность последовательности это вообще сумма по всем путям от начала до конца и может быть вычислена как значение переменной f в конце:

$$\Pr(x) = \sum_{\text{all paths}} \Pr(path) = f_E(L)$$

Теперь разберемся как считать вероятность $\Pr(x_{i+1} \dots x_L | \pi(i) = S_k) = b_k(i)$ (рис.6.7). Если нам известны суммы по путям для всех состояний в следующей позиции $i+1$, то можно вычислить эти суммы для позиции i . Для

Рис. 6.6: Сумма по всем путям, от B до $S_k(i)$

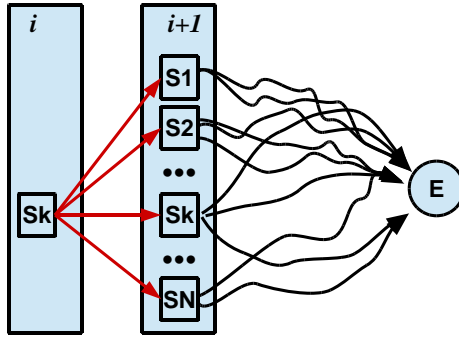
состояния E в конце последовательности наблюдений сумма по всем путям из конца в конец равна 1:

$$b_E(L) = 1$$

Сумма в середине последовательности вычисляется рекурсивно (рис.6.7):

$$b_k(i) = \sum_l a_{kl} \cdot e_l(x_{i+1}) \cdot b_l(i+1)$$

Когда мы придем в начало, переменная $b_B(0)$ равна полной сумме по всем

Рис. 6.7: Сумма по всем путям, от $S_k(i)$ до E

путям, т.е. равна вероятности последовательности:

$$b_B(0) = \Pr(x) = \sum_{all\ paths} \Pr(path) = f_E(L)$$

Обратим внимание, что здесь рекурсия идет в обратном направлении. Поэтому переменная $b_k(i)$ называется backward, в то время, как переменная

$f_k(i)$ называется forward. Подводя итог, можно сформулировать рекурсии:

$$\begin{aligned}
 f_b(0) &= 1 \\
 f_k(i) &= \sum_l f_l(i-1) \cdot a_{lk} \cdot e_k(x_i) \\
 f_E(L) &= \Pr(x) \\
 b_E(L) &= 1 \\
 b_k(i) &= \sum_l a_{kl} \cdot e_l(x_{i+1}) \cdot b_l(i+1) \\
 b_B(0) &= \Pr(x)
 \end{aligned} \tag{6.8}$$

Искомая вероятность состояния в позиции i определяется выражением:

$$\Pr(\pi(i) = S_k) = \frac{f_k(i) \cdot b_k(i)}{f_E(L)} \tag{6.9}$$

6.2.5 Итеративное предсказание

Обратим внимание, что в алгоритмах Витерби и FB встречается только та эмиссионная вероятность, которая соответствует символу в последовательности: $e_k(x_i)$ и только она. В позиции i не встречается эмиссионная вероятность других символов, отличных от x_i . После того, как мы определили вероятности состояний в каждой позиции, мы можем вместо вероятности $e_k(x_i)$ поставить вероятность $P_k(i)$, которая была определена алгоритмом FB. После этого мы опять можем применить алгоритм FB. А можем применить алгоритм Витерби с новыми, позиционно-зависимыми, эмиссионными вероятностями. Этот прием, когда по наблюдениям определяют вероятность, а потом перестраивают модель называется методом максимизации ожидания (Expectation Maximization, EM).

6.3 Биологические примеры

6.3.1 Сайты рестрикции

Есть ферменты – рестриктазы, которые распознают определенную последовательность. Попробуем построить НММ для поиска сайтов рестрикции, например CATG. Очевидно, что это достаточно бессмысленный пример, однако на нем мы увидим некоторые особенности того, как это работает. При построении мы предполагаем, что есть фоновая последовательность, не содержащая сайты связывания и, собственно, сайт связывания. Сайт связывания будем описывать в виде серии состояний – по одному состоянию на позицию. Ясно, что переходные вероятности между позициями сайта равны 1: действительно, после первой позиции обязательно должна идти вторая позиция, а после второй – обязательно третья и т.д. По окончании сайта мы

должны вернуться в фоновую модель. Из фоновой модели мы можем опять попасть на сайт рестрикции, после чего вернуться на фоновую модель. Все это показано на рис.6.8

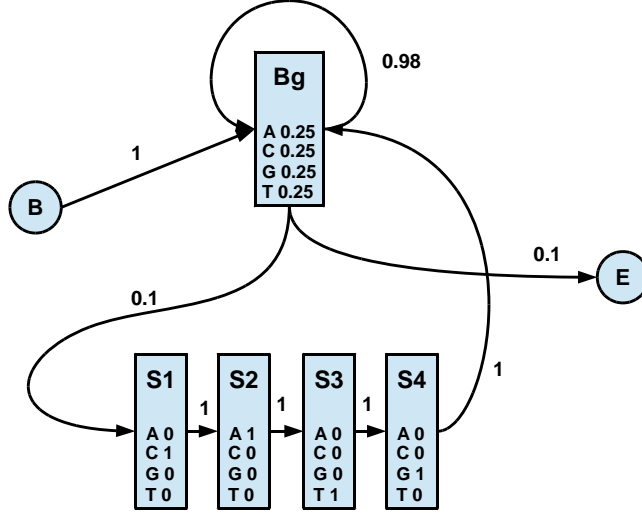


Рис. 6.8: Первый вариант модели поиска сайта CATG

Какими свойствами обладает эта модель? Во-первых, в этой модели сайт *никогда* не появится (не будет распознан) в начале и в конце последовательности (угадайте, почему?). Чтобы ее поправить надо допустить вход в сайт из начала и выход из конца сайта в конец последовательности рис.6.9.

Возникает естественный вопрос – если с эмиссионными вероятностями все понятно, то на что влияют переходные вероятности? Рассмотрим последовательность *с с а t c A T G а t а g а*. Весь текст может быть порожден как фоновой моделью, так и с переключением на модель сайта. Вычислим правдоподобие для обоих случаев.

$$\begin{aligned} \Pr(x|Bg) &= a_{B-Bg} \cdot a_{Bg-Bg}^3 \cdot e_{Bg}^4 a_{Bg-Bg}^5 e_{Bg}^5 \cdot a_{Bg-E} \\ \Pr(x|Site) &= a_{B-Bg} \cdot a_{Bg-Bg}^3 \cdot e_{Bg}^4 a_{Bg-site} \cdot 1^4 1^5 e_{Bg}^5 \cdot a_{Bg-E} \end{aligned}$$

В первом случае мы переходим из начала в фон, генерируем 4 буквы фона, потом опять переходим 5 раз в фон и генерируем еще 4 буквы фона, потом опять переходим в фон и генерируем последние 4 буквы фона. Во втором случае мы после перехода из начала в фон генерируем 4 буквы из фона, потом переходим на сайт. Там переходные и эмиссионные вероятности вероятности равны 1, потом переходим в фон. Посчитаем отношение правдоподобия.

$$L = \frac{\Pr(x|Site)}{\Pr(x|Bg)} = \frac{a_{B-Bg} \cdot a_{Bg-Bg}^3 \cdot e_{Bg}^4 a_{Bg-site} \cdot 1^4 1^5 e_{Bg}^5 \cdot a_{Bg-E}}{a_{B-Bg} \cdot a_{Bg-Bg}^3 \cdot e_{Bg}^4 a_{Bg-Bg}^5 e_{Bg}^5 \cdot a_{Bg-E}}$$

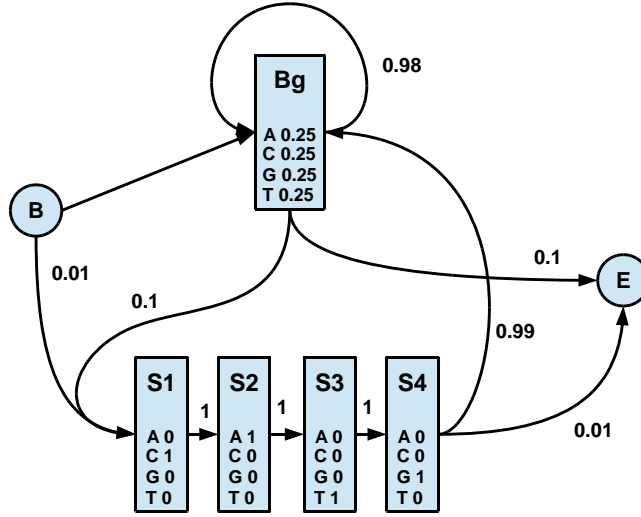


Рис. 6.9: Второй вариант модели поиска сайта CAGT

после сокращения получим:

$$L = \frac{a_{Bg-site}}{a_{Bg-Bg}^5 e_{Bg}^4}$$

Отсюда делаем вывод, что если $a_{Bg-site} < a_{Bg-Bg}^5 e_{Bg}^4$, то сайт никогда не будет найден, в противном случае сайт будет найден гарантировано. Аналогичные ограничения будут наложены на переходные вероятности из начала в сайт и из сайта в конец. Однако для апостериорного декодирования ситуация несколько иная. Если построить графики распределения вероятностей состояний $S1 \dots SL$ вдоль последовательности, то мы увидим характерные пики в местах сайтов.

6.3.2 Сайты связывания транскрипционных факторов

Предыдущая модель легко обобщается на поиск сайтов связывания транскрипционных факторов. Просто надо заменить эмиссионные вероятности. В этом случае эмиссионные вероятности на сайте уже будут достаточно разнообразными. В результате мы получим отношение правдоподобия в виде

$$L = \frac{a_{Bg-site}}{a_{Bg-Bg}^{l+1}} \cdot \frac{e_{si}(x_i)}{e_{Bg}(x_i)}$$

где l – длина сайта. Прологарифмируем:

$$\log L = A + \sum \log \left(\frac{e_{si}(x_i)}{e_{Bg}(x_i)} \right), \quad A = -\log \left(\frac{a_{Bg-Bg}^{l+1}}{a_{Bg-site}} \right)$$

Если мы для всех букв вычислим заранее величину

$$M_i(\alpha) = \log \left(\frac{e_{si}(\alpha)}{e_{Bg}(x_i)} \right),$$

то получим матрицу размером (алфавит \times длина сайта). Эта матрица называется позиционной весовой матрицей (PWM). Значения весов в этой матрице может быть как положительными, так и отрицательными. При наложении матрицы на последовательность получается вес, который сравниваем с порогом. Если вес больше порога (A), то считаем, что в данной позиции есть сайт. Ясно, что величина порога отвечает за ожидаемую частоту встречаемости сайта. Рассмотрим пример. Пусть есть весовая матрица:

a	1.58	1.22	0.00	-0.58
c	-1.58	-0.58	0.74	-1.58
g	-1.58	-0.58	-1.58	1.42
t	-1.58	-1.58	0.00	-1.58

вычислим ее вес на последовательности **ACGG**. Суммируем $1.58 + (-0.58) + (-1.58) + 1.42 = 0.83$. Далее сравниваем ее с порогом A . Положим, что сайт встречается примерно каждые 10 букв. Тогда $a_{Bg-Bg} = 0.9$, $a_{Bg-site} = 0.1$.

Пороговое значение равно $A = \log \left(\frac{a_{Bg-Bg}^{t+1}}{a_{Bg-site}} \right) = 2.56$. Полученное значение PWM меньше порога 2.56. Считаем, что последовательность **ACGG** не является сайтом связывания, в то время как **atac** имеет вес 3.15 и она соответствует нашей модели сайта, хотя в некоторых позициях матрица отрицательна. Эта модель очень проста и никто явно не использует алгоритм Витерби. Для решения таких задач достаточно использовать позиционную весовую матрицу, хотя НММ дает вероятностную интерпретацию результата.

6.3.3 Трансмембранные сегменты в белках

Следующее применение НММ – это поиск трансмембранных сегментов в последовательностях белков. Мы понимаем, что участки белков, пронизывающие мембрану, должны быть гидрофобными. Более того, в них должно быть много алифатических (разветвленных) остатков типа лейцина. Последовательность имеет два типа сегментов – глобулярные и трансмембранные. Поэтому, наверное, схема должна быть такой (рис.6.10). Однако, такая схема будет неправильной. Во всяком случае при генерации последовательностей распределение длин трансмембранных сегментов будет геометрическим (рис.6.11А). Действительно, вероятность сегмента длины L определяется как повторить L раз переход из TM в TM , а потом выйти из состояния TM : $P(L) = a_{TM-TM}^L \cdot a_{TM-GI}$. Но реальные трансмембранные сегменты – это альфа-спирали, пронизывающие мембрану (рис.6.11В) и они не могут быть короткими. С другой стороны, они не могут быть слишком длинными. Ожидаемое распределение длин должно быть как на рис.6.11С. Чтобы исправить ситуацию придется усложнить модель. Вспомним, как моделировались сайты связывания. У них длина была фиксированная. Чтобы смоделировать сайт фиксированной длины мы использовали столько внутренних

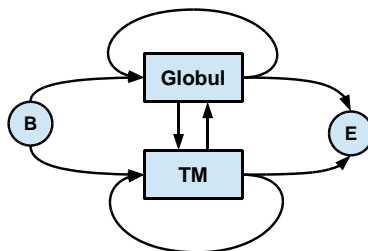


Рис. 6.10: Простейшая модель трансмембранных сегментов

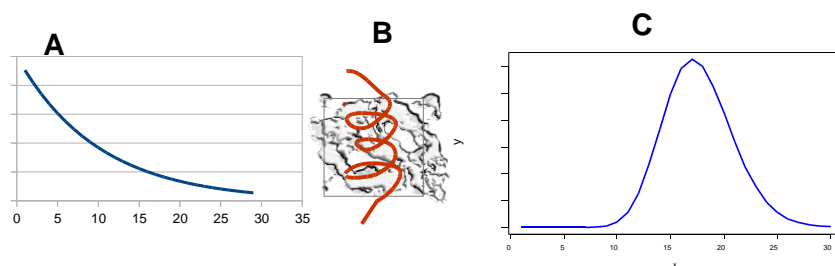


Рис. 6.11: А – распределение длин трансмембранных сегментов при генерации с помощью модели рис.6.10; В – белок, пронизывающий мембрану; С – распределение, которое должно быть

состояний, какова была длина сайта. При этом переходные вероятности были равны 1. Таким образом можно построить модель для ТМ длины ровно 20. Чтобы допустить более короткие сегменты, нам надо позволить покидать ТМ раньше, а чтобы позволить иметь более длинные сегменты, позволим последнему состоянию возвращаться в себя. В результате получаем модель (рис.6.12)

6.3.4 НММ – Лего

Можно строить маленькие модельки и их объединять в более сложные. Например, мы рассматривали сайты связывания. На самом деле у нас было две под-модели – модель сайта и модель фона (рис.6.13А), каждая из которых имеет свой *B* и свой *E*. Мы можем начало одной модели соединить с концом другой и породить новое “немое” состояние – состояние, которое не продуцирует символы, а имеет только переходные вероятности. В результате такого объединения получаем красивую модель (рис.6.13В). Каждый элемент этой модели – сам по себе является моделью. С помощью такого приема, например, можно сконструировать точку инициации трансляции (RBS). Он состоит из мотива Шайна-Дальгарно (SD), спейсера перемен-

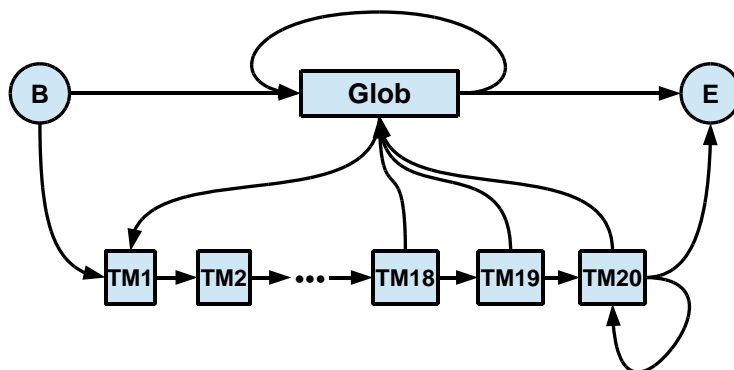


Рис. 6.12: Уточненная модель трансмембранных сегментов

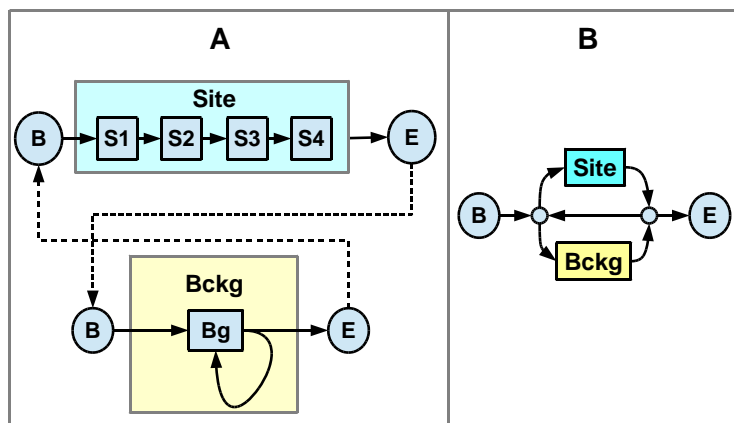


Рис. 6.13: НММ – конструктор. А – модель сайта и модель фона; В – объединенная модель

ной длины и старт-кодона (рис.6.14А). Каждый из этих элементов также можно смоделировать. Мотив Шайна-Дальгарно моделируется также как сайт связывания транскрипционных факторов – там в каждой позиции свои распределения эмиссионных вероятностей (рис.6.14В). Спейсер имеет переменную длину и его можно смоделировать также, как трансмембранные сегменты (только эмиссионные вероятности берутся из фона)(рис.6.14С), старт-кодон моделируется как несколько кодонов, причем переходные вероятности из старта отражают частоту использования кодонов.

Биологическое замечание. Мотив Шайна-Дальгарно есть у *E.coli* и других протеобактерий, а также у *B.subtilis* и других фирмикут. Но никакого мотива нет у очень многих бактерий, например у цианобактерий и у

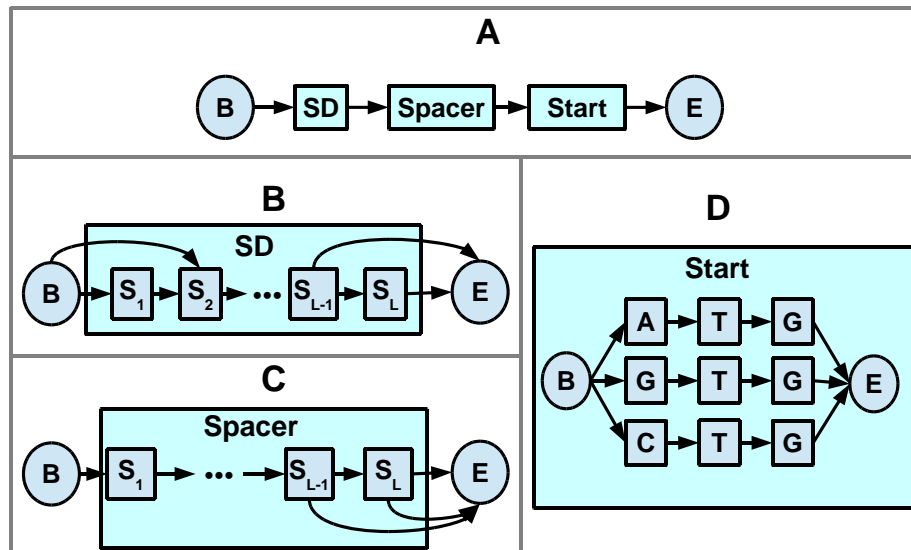


Рис. 6.14: Модель инициации трансляции. A – композитная модель; B – модель мотива Шайна-Дальгарно; C – модель спейсера; D – модель старт-кодона

микобактерий, а также у микоплазм. Оценим количество параметров модели. Модель мотива Шайна-Дальгарно содержит 6 позиций, для каждой из которых есть 4 эмиссионные вероятности (3 независимые). Модель спейсера имеет 6 переходных вероятностей обеспечивающих распределение длин (5 независимых). Модель старт-кодона имеет три переходные вероятности, отражающие частоту встречаемости старт-кодонов (2 независимых). Итого 25 независимых параметров.

6.3.5 Предсказание генов в прокариотах

Чтобы построить модель для поиска генов в геноме, построим сначала модель гена (рис.6.15A). Белок-кодирующий ген состоит из старта, описанного ранее, последовательности кодонов и стоп-кодона. Кодон описывается как множество из 61 композитных состояний, аналогичных построенной ранее модели сайта рестрикции (рис.6.15B).

Переходные вероятности $a_{B-codon}$ отражают частоту встречаемости кодонов. Стоп-кодон описывается аналогично обычным кодонам (рис.6.15C). При этом переходные вероятности также неравномерны, поскольку разные стоп-кодоны по-разному используются. В частности, в *E.coli* кодон TAG используется в 7 раз реже, чем кодон ТАА. Построив модели для кодонов можно их объединить в модель гена (рис.6.15A), которую затем скомбинировать с моделью фона, в результате получим модель (рис.6.15D)

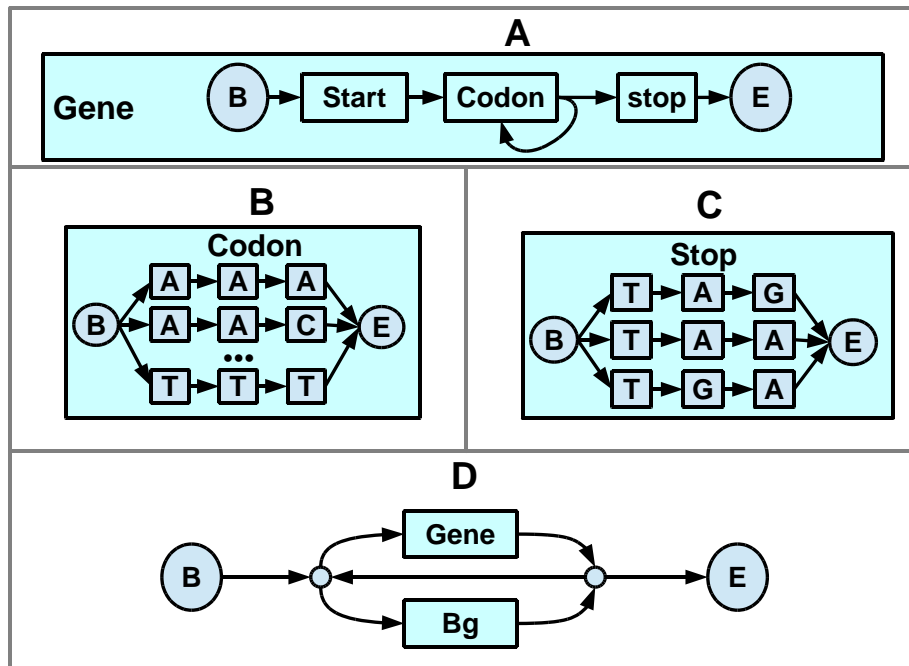


Рис. 6.15: Модель гена в геноме. A – композитная модель гена; B – модель кодона; C – модель стоп-кодона; D – композитная модель генома с генами

Подсчитаем число параметров модели генов в геноме. Модель старта имеет 25 независимых параметров. Модель кодона имеет 61 параметр – переходные вероятности, отражающие частоту использования кодонов – из них 60 независимых. Модель стоп-кодона содержит три переходные вероятности (2 независимые). Есть еще в модели гена вероятность остаться в состоянии кодона. Эта вероятность отвечает за характерную длину гена. Итого 88 параметров. В модели фона есть 4 эмиссионные вероятности (3 независимые) и вероятность остаться в состоянии фона – итого 4 вероятности. Еще есть вероятности выйти на конец и вероятности начать с гена или с фона. Итого имеем 94 независимых параметра.

Общее число состояний модели – 21 состояние в старте, 60 состояния в кодонах и 9 состояний в стопах, 1 состояние в фоне, 2 пересадочных состояния и состояния B, E. Итого 217 состояний.

При этом модель обладает существенным недостатком – таким же, что и наивная модель трансмембранных сегментов – распределение длин генов в этой модели предполагается экспоненциальным. Чтобы это исправить придется усложнить модель гена, добавив в нее примерно 300 композитных состояний кодонов, чтобы приблизить распределение длин белков к наблюдаемому.

6.4 Оценка параметров модели

Модель содержит два элемента: конечный автомат, порождающий последовательность и наборы вероятностей – переходных и эмиссионных. Топологию конечного автомата зафиксируем и зададимся вопросом как оценить наборы вероятностей. Эта процедура называется обучение модели. Здесь есть два подхода. Во-первых, обучение с учителем. В этом случае нам известен некоторый набор последовательностей, для которых известна разметка. Например, если мы строим модель для поиска альфа-спиралей в белках, у нас есть достаточно большой набор белков с известной пространственной структурой, откуда можно извлечь разметку и по этой разметке найти вероятностные параметры.

Однако есть достаточно большой класс задач, где у нас нет достаточно большого количества данных, для того, чтобы оценить параметры. Например, та же проблема трансмембранных сегментов. Это связано с тем, что получение кристаллов мембранных белков – сложная экспериментальная задача и количество структур таких белков весьма ограничено. Другой вариант – поиск сайтов связывания транскрипционных факторов. Здесь нам точно не известно, что узнает тот или иной белок, а известен лишь набор последовательностей, где есть связывание, но позиции нам не известны. В этом случае применяют процедуру обучения без учителя.

6.4.1 Обучение с учителем

Максимизация правдоподобия

При обучении с учителем нам дан набор последовательностей $X = \{x^s\}$, для которых известна разметка в соответствии с представленной схемой. Для того, чтобы оценить параметры $\theta = \{a_{kl}, e_k(\alpha)\}$, вычислим правдоподобие

$$L(X|\theta) = \prod_s L(x^s|\theta)$$

Для каждой последовательности правдоподобие есть произведение наблюдаемых переходных и эмиссионных вероятностей. Их можно сгруппировать, тогда получим:

$$L(x^s|\theta) = \prod a_{kl}^{A_{kl}^s} \cdot \prod e_k^{E_{k,\alpha}^s}(\alpha)$$

где A_{kl}^s – количество наблюдений переходов $k \rightarrow l$ в представленной размеченной последовательности x^s , а $E_{k,\alpha}^s$ – сколько раз мы видели букву α в состоянии k . Перемножая правдоподобия для всех последовательностей, получим

$$L(X|\theta) = \prod a_{kl}^{A_{kl}} \cdot \prod e_k^{E_{k,\alpha}}(\alpha)$$

где A_{kl} – полное количество наблюдений переходов $k \rightarrow l$ в представленных размеченных последовательностях, а $E_{k,\alpha}$ – сколько раз мы видели букву α в состоянии k во всех последовательностях. Если при декодировании Витерби последовательностей мы искали путь с наибольшим правдоподобием,

то здесь нам путь известен. мы будем искать параметры $\theta = \{a_{kl}, e_k(\alpha)\}$, которые максимизируют правдоподобие при заданном пути. Но при этом надо учитывать ограничения на параметры $\sum_l a_{kl} = 1$, $\sum_\alpha e_k(\alpha) = 1$

$$L(X|\theta) \rightarrow \max_{\theta} \quad | \quad \sum_l a_{kl} = 1, \quad \sum_\alpha e_k(\alpha) = 1$$

Применяя метод неопределенных множителей Лагранжа, строим функционал

$$\begin{aligned} \Phi &= L(\theta) - \lambda_k \left(\sum_l a_{kl} - 1 \right) - \mu_k \left(\sum_\alpha e_k(\alpha) - 1 \right) \\ &= \prod_s a_{kl}^{A_{kl}} \cdot e_k^{E_{k,\alpha}}(\alpha) - \lambda_k \left(\sum_l a_{kl} - 1 \right) - \mu_k \left(\sum_\alpha e_k(\alpha) - 1 \right) \end{aligned} \quad (6.10)$$

Прежде, чем считать производные, заметим, что если $f(x) = x^n$, то $\frac{df}{dx} = nx^{n-1} = nf(x)/x$. Чтобы найти оптимум дифференцируем (6.10) по параметрам $\theta = \{a_{kl}, e_k(\alpha)\}$ с учетом сделанного наблюдения:

$$\frac{\partial \Phi}{\partial a_{nm}} = A_{nm} \frac{L(X|\theta)}{a_{nm}} - \lambda_m = 0; \quad \frac{\partial \Phi}{\partial e_m(\beta)} = E_m(\beta) \frac{L(X|\theta)}{e_m(\beta)} - \mu_m = 0$$

Из этих уравнений можно получить

$$a_{nm} = A_{nm} \frac{L(X|\theta)}{\lambda_m}; \quad e_k(\beta) = E_k(\beta) \frac{L}{\mu_k}$$

Для того, чтобы определить множители λ_m , μ_m , воспользуемся ограничениями на вероятности:

$$\sum_m a_{nm} = \sum_m A_{nm} \frac{L(X|\theta)}{\lambda_n} = 1; \quad \sum_\alpha E_k(\alpha) \frac{L(X|\theta)}{\mu_k} = 1$$

Отсюда получаем:

$$\frac{\lambda_n}{L(X|\theta)} = \sum_m A_{nm}; \quad \frac{\mu_k}{L(X|\theta)} = \sum_\alpha E_k(\alpha) = 1$$

Окончательно получаем тривиальную оценку:

$$a_{nm} = \frac{A_{nm}}{\sum_m A_{nm}}; \quad e_k(\beta) = \frac{E_k(\beta)}{\sum_\alpha E_k(\alpha)} \quad (6.11)$$

Т.е. оценка вероятности равна числу наблюдений, деленное на число испытаний.

Байесова оценка

Оценка вероятности с помощью максимизации правдоподобия является хорошей при достаточно большом числе наблюдений – каждое событие должно быть наблюденно не менее 1000 раз, что не всегда доступно. Поэтому можно прибегнуть к байесовой оценке. Вероятность того, что какой-то параметр равен θ может быть записана так:

$$\Pr(\theta|X) = \frac{\Pr(X|\theta) \Pr(\theta)}{\Pr(X)}$$

Использование в качестве априорного распределения распределение Дирихле $f = Z^{-1} \prod \theta_i^{\psi_i}$ приводит к апостериорному распределению:

$$\Pr(\theta|X) = \frac{\Pr(X|\theta) Z^{-1} \prod \theta_i^{\psi_i}}{\Pr(X)}$$

Вспоминая, что $\theta = \{a_{kl}, e_k(\alpha)\}$, получаем:

$$\begin{aligned} \Pr(\theta|X) \sim & \prod a_{kl}^{A_{kl}} \cdot \prod e_k^{E_{k,\alpha}}(\alpha) \cdot \prod a_{kl}^{\psi_{a_{kl}}} \cdot \prod e_k(\alpha)^{\psi_{e_k(\alpha)}} = \\ & \prod a_{kl}^{A_{kl} + \psi_{a_{kl}}} \cdot \prod e_k^{E_{k,\alpha} + \psi_{e_k(\alpha)}}(\alpha) \end{aligned}$$

Сделаем MAP оценку. Для этого найдем максимум апостериорной вероятности при тех же ограничениях на вероятности a_{kl} ; $e_k(\alpha)$. Применив ту же технику, получим:

$$a_{kl} = \frac{A_{kl} + \psi_{kl}}{\sum_m A_{ml} + \psi_{a_{ml}}}; \quad e_k(\alpha) = \frac{E_k(\alpha) + \psi_{k\alpha}}{\sum_{\beta} E_k(\beta) + \psi_{e_{k\beta}}} \quad (6.12)$$

т.е. мы к реальным счетчикам A_{kl} , $E_{k,\alpha}$ добавляем значения $\psi_{a_{ml}}$; $\psi_{e_{k\beta}}$, которые, с одной стороны являются параметрами распределения Дирихле, а с другой – как бы счетчики. Эти величина называются псевдосчетчиками.

6.4.2 Обучение без учителя

Если у нас нет, или очень мало, наблюдений, то хорошего учителя нет. Поэтому возникает задача обучения с нуля. Алгоритм такого обучения весьма прост, и основан на итеративной процедуре максимизации ожидания

Витерби обучение

Простейшим подходом является обучение, основанное на Витерби предсказании (рис.6.16). Сначала иницилируем параметры – либо из представления о том, какие они должны быть, либо просто случайно из некоторых распределений (например, из равномерных). Затем с помощью алгоритма Витерби размечаем все последовательности по состояниям. По новым разметкам последовательностей с помощью максимизации правдоподобия или

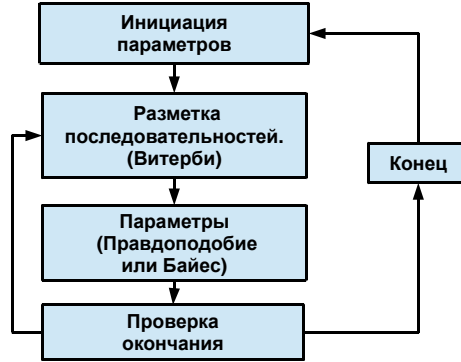


Рис. 6.16: Обучение Витерби

максимизации апостериорных вероятностей определяем новые значения параметров и возвращаемся к алгоритму Витерби. Процесс заканчивается, если параметры перестали меняться, либо если сделано достаточно итераций (обычно хватает 20 шагов). Поскольку алгоритм находит только локальный оптимум, надо его запустить несколько раз с разными исходными параметрами – надо еще раз сгенерировать новые стартовые параметры и повторить процесс. Если мы использовали несколько исходных наборов, то выбираем набор, у которого максимальное правдоподобие.

Алгоритм Баума-Вельча

Оценка вероятности по максимальному правдоподобию определяется отношением $p = n/N$, где n – количество успехов, а N – число испытаний. Количество чего-либо есть просто сумма 1: $n = \sum_n 1$. Если мы заменим 1 на некоторый вес ω_i , то мы можем обобщить понятие количества. Веса ω_i отражают нашу уверенность в наблюдении успеха или неуспеха. Теперь вернемся к нашим моделям. Алгоритм “вперед-назад” дает нам как раз такую замену – вместо утверждения, что в позиции i было состояние k мы можем использовать вероятность $P_k(i)$ того, что в соответствующей позиции было состояние k . Это позволяет нам выбрать в качестве веса ω_i использовать вероятность $P_k(i)$. Таким образом счетчики числа букв в состояниях можно заменить:

$$E_k(\alpha) = \sum_{\substack{i: x_i = \alpha \\ \pi(i) = k}} 1 \Rightarrow E_k(\alpha) = \sum_{i: x_i = \alpha} P_k(i)$$

Для оценки эффективного количества переходов используем следующие соображения (рис.6.17). Вероятность перехода из состояния k в позиции i в

состояние l в позиции $i + 1$ равна:

$$\begin{aligned} \Pr(\pi(i) = k, \pi(i + 1) = l, \mid x_1, \dots, x_L) &= \\ &= \frac{\Pr(\pi(i) = k, \pi(i + 1) = l; x)}{\Pr(x)} = \\ &= \frac{\Pr(x_1, \dots, x_i, \pi(i) = k) \cdot \Pr(x_{i+1} \dots x_L \mid \pi(i + 1) = l)}{\Pr(x)} \end{aligned}$$

Вероятности в числителе – это переменные $f_k(i)$ и $b_l(i + 1)$. Поэтому можно записать:

$$A_{kl} = \sum_i \frac{f_k(i) \cdot b_l(i + 1)}{P(x)}$$

Теперь строим алгоритм максимизации ожидания, основанный на вероят-

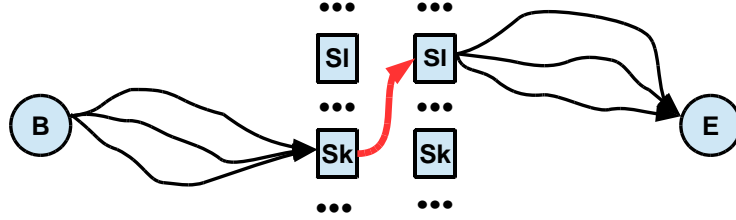


Рис. 6.17: Обучение по Бауму-Велчу: оценка вероятности перехода

ностях состояний (рис.6.18). Выбираем случайный набор параметров. Применяем алгоритм forward-backward для всех последовательностей. Оцениваем эмиссионные и переходные вероятности по формулам:

$$\begin{aligned} E_k(\alpha) &= \sum_s \sum_{i: x_i^s = \alpha} P_k^s(i); \quad e_k(\alpha) = \frac{E_k(\alpha)}{\sum_\beta E_k(\beta)} \\ A_{kl} &= \sum_s \sum_i \frac{f_k^s(i) \cdot b_l^s(i + 1)}{P^s(x)}; \quad a_{kl} = \frac{A_{kl}}{\sum_l A_{kl}} \end{aligned}$$

Снова применяем алгоритм forward-backward и переоцениваем параметры. Так продолжаем до тех пор, пока параметры не перестанут меняться, или по достижении заданного количества итераций. Это алгоритм сходится медленнее (нужно около 100 итераций), чем Витерби обучение, но при этом он дает лучшее качество – алгоритм реже посещает локальные минимумы. Поскольку такой подход находит лишь локально-оптимальный набор параметров, надо его запустить несколько раз.

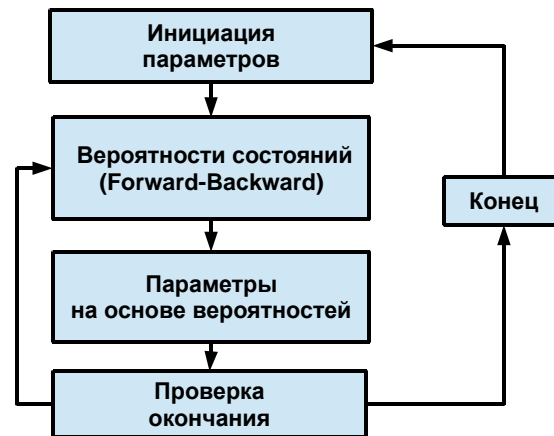


Рис. 6.18: Обучение по Бауму-Велчу

6.5 Пример применения НММ к эпигенетике

6.5.1 Вводные замечания

В эукариотических клетках ДНК упакована в хроматин. Важнейшим элементом хроматина являются нуклеосомы, которые состоят из высоко консервативных и сильно положительно заряженных белков – гистонов H2A, H2B, H3, H4, которые образуют октамер, на который намотана ДНК (1.7 витка) – с одной нуклеосомой связано примерно 140 нуклеотидов.

Хроматин имеет разные типы упаковки. Наиболее плотно упакованные области называются гетерохроматином. Активный хроматин называется эухроматином. Гистоны часто несут пост-трансляционные модификации – фосфорлирование, ацетилирование, метилирование и много чего еще. По современным представлениям наиболее важными для плотности упаковки, и, следовательно, для регуляции экспрессии генов являются метилирование и ацетилирование разных остатков лизина на гистоне H3. При этом, метилирование разных остатков приводит к прямо противоположным эффектам, например, метилирование 9-го лизина (H3K9me3) характерно для гетерохроматина, в то время как метилирование 4-го лизина (H3K4me3) характерно для активных промоторов. Однако, эти модификации, как правило, не работают по-отдельности. Поэтому для определения состояния хроматина надо смотреть на комбинации модификаций.

Также важную роль в структуре хроматина играют и некоторые другие белки, например белок CTCF. Этот белок является инсуляторным белком, точки его связывания ограничивают действие энхансеров. Есть данные, что он ответственен за образование хроматиновых петель. Для экспериментального определения локализации разных гистоновых модификаций применяют методы иммунопреципитации хроматина ChIP-seq. Все модифи-

кации хроматина называются хроматиновыми метками.

Итак, есть предположение, что комбинации модификаций и связывание некоторых белков отвечают состояниям хроматина, и есть экспериментальные данные. Разделение на эухроматин и гетерохроматин представляется слишком грубым.

Задача заключается в определении множества состояний хроматина и разметки генома по этим состояниям. Для состояний хроматина принят термин – цвет хроматина. Особенность этой задачи заключается в том, что мы не знаем, что ищем – во всяком случае нам заранее не известно множество состояний хроматина.

6.5.2 Алгоритм Chrom-HMM

Для начала мы дискретизуем наши данные – найдем статистически достоверные пики в данных ChIP-seq по каждой хроматиновой метке. Пики могут быть достаточно протяженными (многие килобазы) и короткими (сотни нуклеотидов). В результате по каждой метке получаем на геноме значения 0 или 1, в зависимости от того принадлежит ли данная позиция пику той или иной метки. Сделав это для всех меток из данного набора, мы получим в каждой вектор из 0 и 1. В работе (Ernst J., Kheradpour P., Mikkelsen T.S., Shores N. et al. (2011). Mapping and analysis of chromatin state dynamics in nine human cell types. *Nature* 473, 43-49) использовался следующий набор меток:

CTCF	Инсулятор
H3K27me3	Гетерохроматин, поликомб
H3K36me3	Активная транскрипция
H4K20me1	Активная транскрипция
H3K4me1	Активный энхансер
H3K4me2	Активный промотор/энхансер
H3K4me3	Активный промотор
H3K27ac	Активный промотор/энхансер
H3K9ac	Активный промотор/энхансер

Модель предполагает возможность переходов из всех состояний во все состояния. Задавшись числом состояний и применив алгоритм Баума-Вельча можно обучит модель и получить разметку генома по цветам хроматина. В результате получены следующие состояния и эмиссионные вероятности (рис.6.19А,В).

Применение этой модели к геному дает такие результаты (рис.6.19). Этот подход сейчас является стандартным при разметке цветов хроматина. В частности, для дрозофилы была применена та же схема, но с меньшим числом состояний. К сожалению, при этой разметке не принималась во внимание метка H3K9me3, которая отвечает за плотный гетерохроматин. Другой проблемой этого подхода является то, что около половины генома соответствует состояниям 13 и 15, где либо нет меток совсем, либо наоборот – представлены все метки. Это обусловлено двумя обстоятельствами. Во-первых, это экспериментальный протокол. Он включает в себя обработку

Задача: Предсказать состояния в остальных позициях. Придумайте подход, который позволит решить эту задачу.

Расширенная задача: В предыдущем случае мы знали в позициях точные состояния. Однако, в отдельных позициях мы знаем не точно состояния, а только класс состояний. Например, мы знаем, что некоторая позиция белка – трансмембранная, но в модели несколько состояний отвечают трансмембранному сегменту. Это значит, что позиция принадлежит классу состояний. Придумайте алгоритм для декодирования, в случае когда для *некоторых* позиций мы знаем класс состояний.

Глава 7

НММ выравнивание

7.1 Постановка задачи

До сих пор рассматривались задачи порождения отдельных последовательностей. При этом как порождение последовательностей, так и расшифровка предполагали, что все последовательности порождаются независимо. Однако, последовательности могут порождаться и парами и порождать выравненные пары последовательностей. Типичный пример выравнивания:

```
TGFIFGAQERI--LIM
|*|    |**||   |||
SGY---GQEKILVILM
```

В этом примере первые три символа порождены совместно. Затем в первой последовательности произошло порождение трех символов, никак не связанных со второй последовательностью. Потом опять произошло порождение 5 символов в обеих последовательностях, далее два были порождены два символа во второй последовательности, и они были независимы от первой последовательности. Наконец, было порождено три пары символов. Этот процесс порождения можно представить в виде простой схемы (рис.7.1) Это является простейшей схемой скрытой марковской модели, порождающей выравнивания. Здесь состояния:

В – начало

М – порождение пары символов

Х – порождение символа в первой последовательности

У – порождение символа во второй последовательности

Е – конец

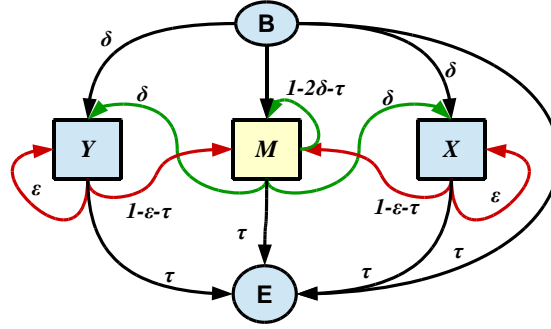


Рис. 7.1: Стандартная схема выравнивания

7.2 Алгоритм Витерби для выравниваний

Схема порождения выравнивания по сути является скрытой марковской моделью. Осталось только определить соответствующие вероятности. Поскольку в состоянии вставки символы порождаются из модели независимого порождения, определим эмиссионные вероятности для состояний X , Y как вероятности порождения символов в мире, т.е. используем частоты встречаемости аминокислот $q(x_i)$. Для состояния M эмиссионные вероятности — это вероятности появления сопоставленных пар $p(x_i, y_i)$ (см. матрицы сопоставления аминокислотных остатков). Переходные вероятности имеют следующий смысл:

- a_{BM} — вероятность начать выравнивание с пары символов
- a_{BX} — вероятность начать со вставки в X
- a_{BY} — вероятность начать со вставки в Y
- a_{BE} — вероятность того, что обе последовательности пустые
- a_{MM} — вероятность остаться в состоянии M — ответственна за длину блоков выравнивания
- a_{MX} — вероятность начать вставку в X
- a_{MY} — вероятность начать вставку в Y
- a_{ME} — вероятность закончить на сопоставленной паре
- a_{XM} — вероятность вернуться к порождению сопоставленных пар
- a_{XX} — вероятность остаться в состоянии X — ответственна за длину вставок в X
- a_{XE} — вероятность закончить на вставке в X
- a_{YM} — вероятность вернуться к порождению сопоставленных пар
- a_{YY} — вероятность остаться в состоянии Y — ответственна за длину вставок в Y
- a_{YE} — вероятность закончить на вставке в Y

Итого мы имеем 14 переходных вероятностей при 4 ограничениях:

$$\begin{aligned} a_{BM} + a_{BX} + a_{BY} + a_{BE} &= 1 \\ a_{MM} + a_{MX} + a_{MY} + a_{ME} &= 1 \\ a_{XX} + a_{XM} + a_{XE} &= 1 \\ a_{YY} + a_{YM} + a_{YE} &= 1 \end{aligned}$$

Кроме того, есть естественные ограничения симметрии:

$$a_{MX} = a_{MY}; \quad a_{XM} = a_{YM}; \quad a_{BX} = a_{BY}; \quad a_{XX} = a_{YY}$$

Еще, наверное, переходные вероятности в начале должны быть такими же, как и в середине, т.е.

$$a_{BM} = a_{MM}; \quad a_{BX} = a_{BM}; \quad a_{BY} = a_{BM}; \quad a_{BE} = a_{ME}$$

Еще можно потребовать, чтобы конец не выделялся

$$a_{XE} = a_{YE} = a_{ME}$$

Не все ограничения являются независимыми. Итого у нас остается $14 - 4 - 4 - 3 = 3$ независимых переходных вероятностей. Обозначим:

$$\begin{aligned} a_{BE} &= \tau; & a_{BX} &= \delta; & a_{BY} &= \delta; & a_{BM} &= 1 - 2\delta - \tau; \\ a_{ME} &= \tau; & a_{MX} &= \delta; & a_{MY} &= \delta; & a_{MM} &= 1 - 2\delta - \tau; \\ a_{XE} &= \tau; & a_{XX} &= \varepsilon; & a_{XM} &= 1 - \varepsilon - \tau; \\ a_{YE} &= \tau; & a_{YY} &= \varepsilon; & a_{YM} &= 1 - \varepsilon - \tau; \end{aligned}$$

Если нам дана пара последовательностей, то, аналогично тому, как делается в обычной скрытой марковской модели, мы можем развернуть конечный автомат (рис.7.1) в граф состояний (рис.7.2), теперь уже двумерный. Поскольку мы получили скрытую марковскую модель, то можно построить алгоритм Витерби для декодирования скрытой марковской модели. Переменные Витерби будут означать наилучшее правдоподобие порождения пары префиксов последовательностей нашей моделью, т.е. $v_k[i, j]$ — наилучшее правдоподобие порождения фрагментов $x_1 \dots x_i$ и $y_1 \dots y_j$ с приходом в состояние k . Понятно, что наилучшее правдоподобие порождения пустых последовательностей в состоянии B равно $v_B[0, 0] = 1$.

Построение алгоритма оптимизации очень похоже на построение алгоритма выравнивания с аффинными штрафами. Позже мы увидим, что это совпадение носит более глубокий характер. Итак, следуя процедуре алгоритма Витерби получаем: Для первых позиций $i = 1, j = 1$ (рис.7.2В)

$$\begin{aligned} v^M(1, 1) &= p(x_1, y_1) & v^B(0, 0) &= (1 - 2\delta - \tau) \\ v^X(1, 0) &= q(x_1) & v^B(0, 0) &= \delta, \\ v^Y(0, 1) &= q(y_1) & v^B(0, 0) &= \delta \end{aligned} \quad (7.1)$$

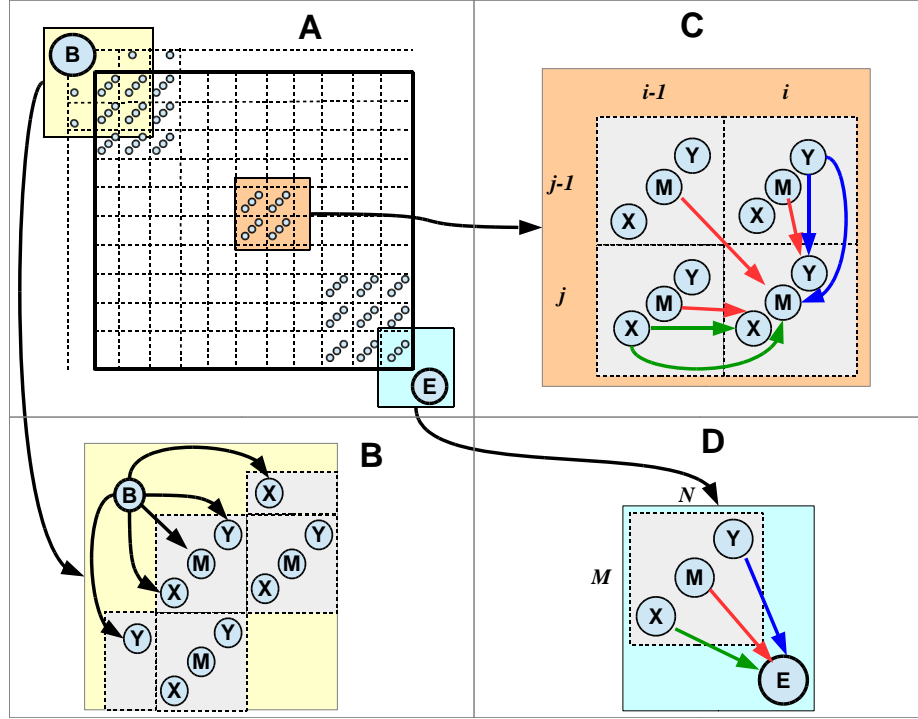


Рис. 7.2: Граф состояний. Слева-вверху (A) – общий вид графа; слева-внизу (B) – подробный граф состояний, связанный с состоянием B ; справа-вверху (C) – подробный граф состояний в середине матрицы. Показаны только ребра, входящие в позицию (i, j) ; справа-внизу (D) – граф состояний в состоянии E

Для внутренних позиций графа (рис.7.2C)

$$\begin{aligned}
 v^M(i, j) &= p(x_i, y_j) \max \begin{cases} v^M(i-1, j-1)(1-2\delta-\tau), & i > 1, j > 1 \\ v^X(i-1, j-1)(1-\varepsilon-\tau), & i > 1, j > 1 \\ v^Y(i-1, j-1)(1-\varepsilon-\tau), & i > 1, j > 1 \end{cases} \\
 v^X(i, j) &= q(x_i) \max \begin{cases} v^M(i-1, j)\delta, & i > 1 \\ v^X(i-1, j)\varepsilon, & i > 1 \end{cases} \\
 v^Y(i, j) &= q(y_i) \max \begin{cases} v^M(i, j-1)\delta, & j > 1 \\ v^Y(i, j-1)\varepsilon, & j > 1 \end{cases}
 \end{aligned} \tag{7.2}$$

Для состояния E (рис.7.2D)

$$v^E(N, M) = \max \begin{cases} v^M(N, M)\tau \\ v^X(N, M)\tau \\ v^Y(N, M)\tau \end{cases} \quad (7.3)$$

Разумеется необходимо запомнить обратные переходы:

$$\pi_k(i, j) = \arg \max v_l(i_{prev}, j_{prev}) a_{lk} e_l(i, j) \quad (7.4)$$

Это уравнение надо понимать так: во всех уравнениях (7.1, 7.2, 7.3) заменяем слева v на π , а \max на $\arg \max$

7.3 Максимизация правдоподобия

Рассмотрим модель выравнивания A и альтернативную модель независимой генерации последовательностей R (рис.7.3A). Модель R устроена так: Мы генерируем последовательность x (или ее игнорируем), после чего генерируем последовательность y (или ее игнорируем), после чего переходим в конец. Вероятность перехода в конец обозначим η . Рассмотрим два пути (рис.7.3A) – один путь по графу состояний модели выравнивания (не обязательно оптимальный), а другой – путь по графу независимой генерации (черная и красная линии на рисунке). Оценим отношение правдоподобия:

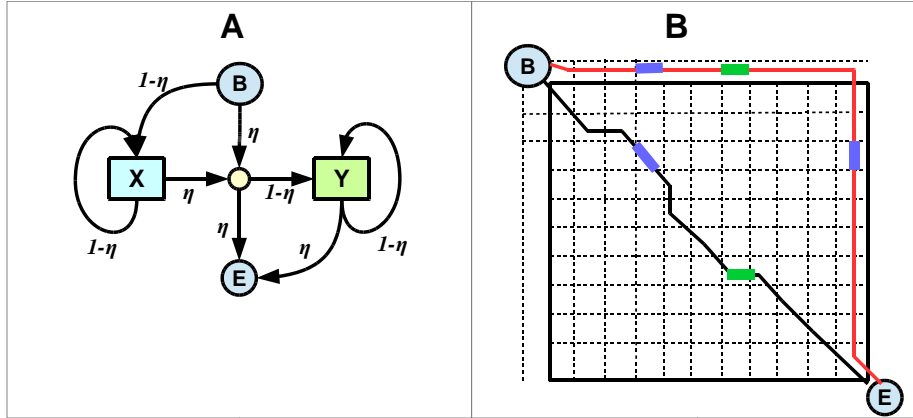


Рис. 7.3: К оценке правдоподобия. А – альтернативная модель. Есть состояния генерации символов в первой последовательности X и символов во второй последовательности Y . В – путь по графу состояний: черный – путь в соответствии с моделью выравнивания, красный – путь независимой генерации

$$L = \frac{Pr(x, y|M, \pi)}{Pr(x, y|R)}$$

Правдоподобие порождения пары последовательностей моделью R можно записать как

$$\begin{aligned} Pr(x, y|R) &= (1 - \eta)^n \eta \prod_i^n q(x_i) \cdot (1 - \eta)^m \eta \prod_i^m q(y_i) \\ &= \eta^2 \prod_i^n (q(x_i)(1 - \eta)) \cdot \prod_j^m (q(y_j)(1 - \eta)) \end{aligned}$$

Действительно, мы вошли в последовательность x (вероятность равна $1 - \eta$), $n - 1$ раз покрутились в состоянии X (вероятность равна $(1 - \eta)^{n-1}$), после чего вышли из этого состояния (вероятность равна η). При этом модель породила символы из последовательности x , вероятность порождения символов равна $\prod_i^n q(y_i)$. Потом перешли в состояние Y (вероятность $1 - \eta$) и аналогично породили последовательность y . Теперь попробуем расписать правдоподобие $Pr(x, y|M, \pi)$. Ясно, что эмиссии в состояниях M, X, Y дадут вклад:

$$eM = \prod_{i \circ j} p(x_i, y_j); \quad eX = \prod_{i \circ} q(x_i); \quad eY = \prod_{j \circ} q(y_j)$$

здесь $i \circ j$ означает, что x_i сопоставлено с y_j , а $i \circ$ ($j \circ$) означает, что x_i (y_j) ни с чем не сопоставлено (вставка). Теперь посмотрим на вклады переходных вероятностей:

$$\begin{aligned} M \rightarrow M &= \prod_{i-1 \circ j-1, i \circ j} (1 - 2\delta - \tau) \\ X \rightarrow M &= \prod_{i-1 \circ, i \circ j} (1 - \varepsilon - \tau) \\ Y \rightarrow M &= \prod_{j-1 \circ, i \circ j} (1 - \varepsilon - \tau) \\ X \rightarrow X &= \prod_{i-1 \circ, i \circ} (\varepsilon) \\ M \rightarrow X &= \prod_{i-1 \circ i-1, i \circ} (\delta) \\ Y \rightarrow Y &= \prod_{j-1 \circ, j \circ} (\varepsilon) \\ M \rightarrow Y &= \prod_{j-1 \circ j-1, j \circ} (\delta) \end{aligned}$$

Теперь разделим правдоподобие модели выравнивания A на правдоподобие модели независимого порождения R . Эмиссионные вероятности дадут вклад:

$$LeM = \frac{\prod_{i \circ j} p(x_i, y_j)}{\prod_{i \circ j} q(x_i)q(y_j)}; \quad LeX = \frac{\prod_{i \circ} q(x_i)}{\prod_{i \circ} q(x_i)} = 1; \quad LeY = \frac{\prod_{j \circ} q(y_j)}{\prod_{j \circ} q(y_j)} = 1$$

На рассматриваемом пути любой символ в последовательности x либо сопоставлен с символом в y , либо не сопоставлен. Поэтому в этом равенстве мы учли все символы в обеих последовательностях, как порожденные моделью A , так и порожденные моделью R . Теперь посмотрим на переходные

вероятности.

$$\begin{aligned}
L_{M \rightarrow M} &= \prod_{\substack{i-1 \circ j-1, \\ i \circ j}} (1 - 2\delta - \tau) / (1 - \eta)^2 \\
L_{X \rightarrow M} &= \prod_{\substack{i-1 \circ, \\ i \circ j}} (1 - \varepsilon - \tau) / (1 - \eta)^2 \\
L_{Y \rightarrow M} &= \prod_{\substack{j-1 \circ, \\ i \circ j}} (1 - \varepsilon - \tau) / (1 - \eta)^2 \\
L_{X \rightarrow X} &= \prod_{\substack{i-1 \circ, \\ i \circ}} \varepsilon / (1 - \eta) \\
L_{M \rightarrow X} &= \prod_{\substack{i-1 \circ j-1, \\ i \circ}} \delta / (1 - \eta) \\
L_{Y \rightarrow Y} &= \prod_{\substack{i-1 \circ j-1, \\ i \circ}} \varepsilon / (1 - \eta) \\
L_{M \rightarrow Y} &= \prod_{\substack{j-1 \circ, \\ j \circ}} \delta / (1 - \eta)
\end{aligned}$$

Здесь тоже учтены все переходы. Действительно, на каждый переход в состояние M мы сделали два перехода в модели R по последовательностям — по последовательности x и по последовательности y , поэтому вклад равен $(1 - \eta)^2$, а каждому переходу в состояния X или Y в модели A соответствует переход в модели R — либо $X \rightarrow X$, либо $Y \rightarrow Y$. Не упираясь в преобразования, можно достаточно легко (хоть и занудно) сгруппировать и разделить элементы: эмиссию в состояниях M , переходные вероятности в состояния X, Y из состояния M , а также переходные вероятности внутри состояний и получить замечательное утверждение:

$$\begin{aligned}
\log L &= \sum_{i \circ j} \log \left(\frac{p(x_i, y_j)}{q(x_i)q(y_j)} \right) + \log \left(\frac{1 - 2\delta - \tau}{(1 - \eta)^2} \right) + \\
&\quad \sum_{\substack{i-1 \circ j-1, \text{ и} \\ i \circ, \text{ или } j \circ}} -\log \left(\frac{\delta(1 - \varepsilon - \tau)}{(1 - \eta)(1 - 2\delta - \tau)} \right) + \\
&\quad \sum_{i \circ \text{ или } j \circ} -\log \left(\frac{\varepsilon}{\tau} \right) = \\
&\quad \sum_{i \circ j} S(p(x_i, y_j) - d \cdot N_d - e \cdot N_e)
\end{aligned}$$

где N_d — число переходов типа $M \rightarrow X$ или $M \rightarrow Y$, а N_e — число переходов типа $X \rightarrow X$ или $Y \rightarrow Y$. Иными словами, мы получаем картину, похожую

на выравнивание с аффинными штрафами за делеции, где d – штраф за инициацию делеции, а e – штраф за продолжение делеции. При этом веса приобретают вероятностную интерпретацию:

$$\begin{aligned} S(\alpha, \beta) &= \log \frac{p(x_i, y_j)}{q(x_i)q(y_j)} + \log \left(\frac{1 - 2\delta - \tau}{(1 - \eta)^2} \right) \\ d &= \log \left(\frac{\delta(1 - \varepsilon - \tau)}{(1 - \eta)(1 - 2\delta - \tau)} \right) \\ e &= \log \left(\frac{\varepsilon}{\tau} \right) \end{aligned}$$

Здесь $S(\alpha, \beta)$ – соответствует матрице сопоставления остатков, d – соответствует штрафу за инициацию вставки, e – соответствует штрафу за продолжение делеции. Рекурсия для максимизации правдоподобия превращается в знакомую рекурсию динамического программирования для аффинных штрафов за делецию: Начало:

$$\begin{aligned} w^M(0, 0) &= 2 \log \eta; \quad w^X(0, 0) = w^Y(0, 0) = -\infty \\ w^X(i, -1) &= w^Y(-1) = w^X(-1, j) = w^Y(-1, j) = -\infty \end{aligned}$$

Рекурсия:

$$\begin{aligned} w^M(i, j) &= s(x_i, y_j) + \max \begin{cases} w^M(i-1, j-1) \\ w^X(i-1, j-1) \\ w^Y(i-1, j-1) \end{cases} \\ w^X(i, j) &= \max \begin{cases} w^M(i-1, j) - d \\ w^X(i-1, j) - e \end{cases} \\ w^Y(i, j) &= \max \begin{cases} w^M(i, j-1) - d \\ w^Y(i, j-1) - e \end{cases} \end{aligned}$$

Завершение:

$$w^E = \max \begin{cases} w^M(n, m) - d \\ w^X(n, m) - e \\ w^Y(n, m) - e \end{cases}$$

7.4 Апостериорное декодирование

Казалось бы – написали столько формул, а получили то, что и так знали – алгоритм построения выравнивания с аффинными штрафами. К тому же со слегка измененной матрицей сопоставления остатков. То, что была получена вероятностная интерпретация выравнивания с аффинными штрафами

– достаточно слабое утешение. Однако, есть весьма важное и полезное расширение. Дело в том, что скрытые марковские модели позволяют не только найти оптимальное (в смысле правдоподобия) декодирование, но также есть алгоритм апостериорного декодирования, который может оценить степень доверия к тем или иным состояниям в разных позициях. В частности, можно оценить насколько достоверно были сопоставлены буквы. Так же, как было при одномерном декодировании, мы можем записать вероятность того, что в позиции (i, j) модель была в состоянии k :

$$\begin{aligned} \Pr(\pi(i, j) = k | x, y) &= \frac{\Pr(x, y, \pi(i, j) = k)}{\Pr(x, y)} \\ &= \frac{\Pr(x_1 \cdots x_i, y_1 \cdots y_j, \pi(i, j) = k) \cdot \Pr(x_{i+1} \cdots x_n, y_{j+1} \cdots y_m | \pi(i, j) = k)}{\Pr(x, y)} \\ &= \frac{f_k(i, j) b_k(i, j)}{\Pr(k)} \end{aligned}$$

7.4.1 Алгоритм Вперед-Назад

Просмотр Вперед

Также, как и в одномерном случае, будем суммировать пути (рис.7.4). Глядя на схему, получаем рекурсию:

Начало:

$$\begin{aligned} f^M(1, 1) &= (1 - 2\delta - \tau) \cdot p(x_i, y_j); \\ f^X(1, 1) &= \delta \cdot q(x_i); \\ f^Y(1, 1) &= \delta \cdot q(y_j) \end{aligned}$$

Рекурсия:

$$\begin{aligned} f^M(i, j) &= p(x_i, y_j) ((1 - 2\delta - \tau) \cdot f^M(i - 1, j - 1) \\ &\quad + (1 - \varepsilon - \tau) \cdot f^X(i - 1, j - 1) \\ &\quad + (1 - \varepsilon - \tau) \cdot f^Y(i - 1, j - 1)), \quad i > 1, j > 1 \\ f^X(i, j) &= q(x_i) (\delta \cdot f^M(i - 1, j) \\ &\quad + \varepsilon \cdot f^X(i - 1, j)), \quad i > 1 \\ f^Y(i, j) &= q(y_j) (\delta \cdot f^M(i, j - 1) \\ &\quad + \varepsilon \cdot f^Y(i, j - 1)), \quad j > 1 \end{aligned}$$

Завершение:

$$f^E = \tau(f^M(n, m) + f^X(n, m) + f^Y(n, m)) = \Pr(x, y)$$

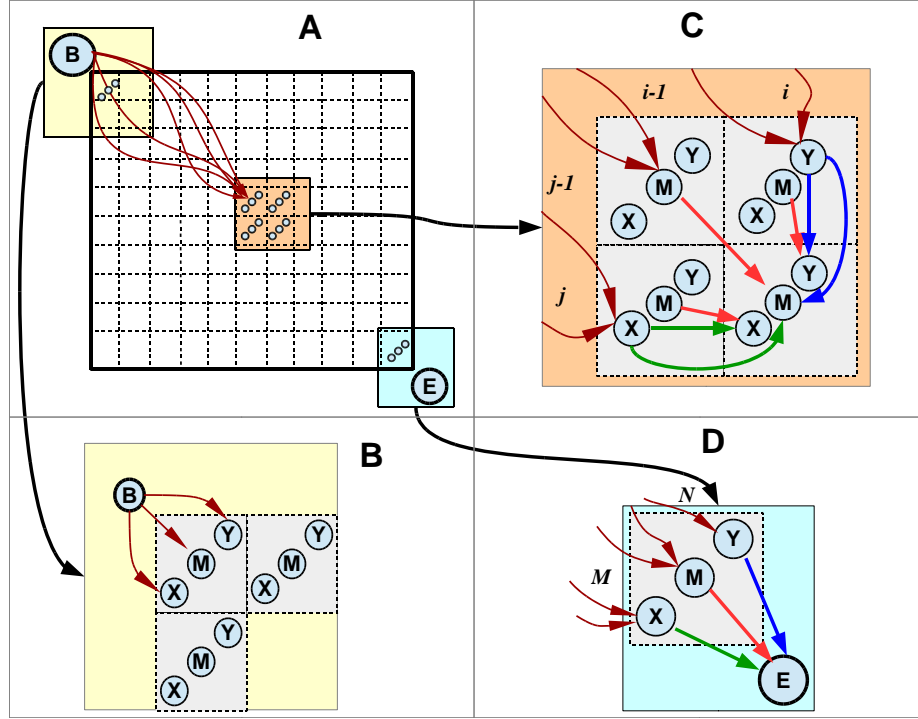


Рис. 7.4: Алгоритм Вперед. А – граф состояний; В – начало; С – рекурсия; D – конец. Коричневые стрелки – это суммы по всем путям.

Просмотр Назад

При просмотре назад используем соображения, аналогичные тем, что были при рассмотрении одномерных моделей. Из каждой вершины графа состояний можно многими путями пройти в конечное состояние E . Суммируя по путям мы можем получить вероятность $b_k(i, j) = \Pr(x_{i+1} \cdots x_n, y_j + 1 \cdots y_m | \pi(i, j) = k)$ (рис.7.5)

Инициация:

$$b^M(m, n) = \tau; \quad b^X(m, n) = \tau; \quad b^Y(m, n) = \tau$$

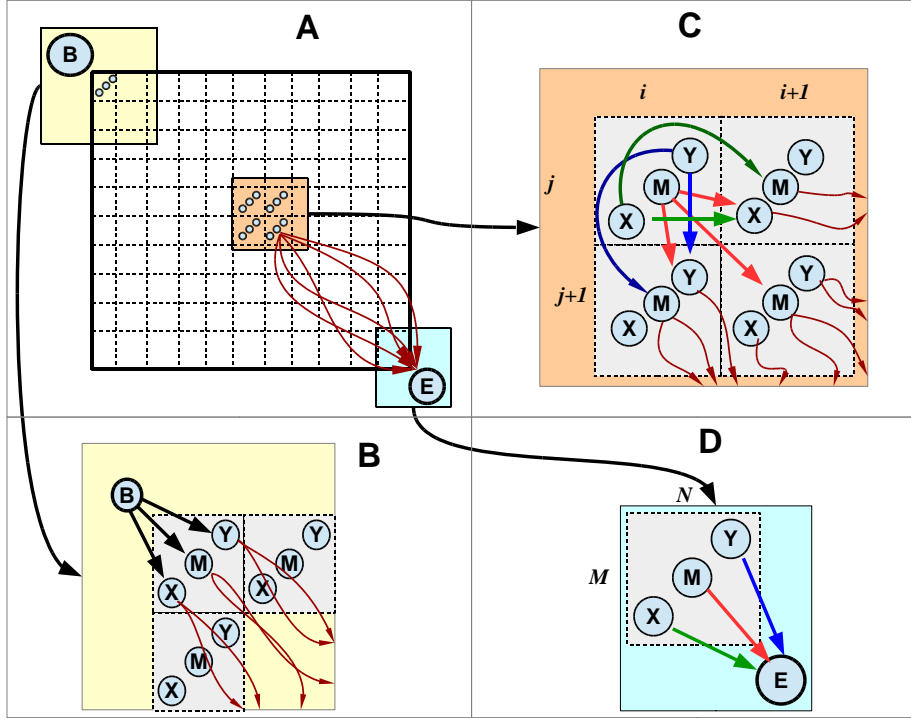


Рис. 7.5: Алгоритм Назад. А – граф состояний; В – начало ; С – рекурсия; D – конец. Коричневые стрелки – это суммы по всем путям.

Рекурсия:

$$\begin{aligned}
 b^M(i, j) = & (1 - 2\delta - \tau) \cdot b^M(i + 1, j + 1) \cdot p(x_{i+1}, y_{j+1}) \\
 & + \delta \cdot b^X(i + 1, j) \cdot q(x_{i+1}) \\
 & + \delta \cdot b^Y(i, j + 1) \cdot q(y_{j+1}) \quad i < n, \quad j < m
 \end{aligned}$$

$$\begin{aligned}
 b^X(i, j) = & (1 - \varepsilon - \tau) \cdot b^M(i + 1, j + 1) \cdot p(x_{i+1}, y_{j+1}) \\
 & + \varepsilon \cdot b^X(i + 1, j) \cdot q(x_{i+1})
 \end{aligned}$$

$$\begin{aligned}
 b^Y(i, j) = & (1 - \varepsilon - \tau) \cdot b^M(i + 1, j + 1) \cdot p(x_{i+1}, y_{j+1}) \\
 & + \varepsilon \cdot b^Y(i, j + 1) \cdot q(y_{j+1})
 \end{aligned}$$

Окончание:

$$b^B(0, 0) = \delta \cdot b^X(1, 1) \cdot q(x_n) + \delta \cdot b^Y(1, 1) \cdot q(y_m) + (1 - 2\delta - \tau) \cdot b^M(1, 1) \cdot p(x_n, y_m)$$

Как это работает

Две последовательности:

S1 gataggcattcataagtttgaaatagctgtccttattctggaacttgat
ttatggctcttgataggcattcataagtttgattatagctgtcctt

S2 gaaaggattatcattcataaggaggagagcaagggccttattctttgat
ttatttctcttaatagg-attcatgagtttgaaatag

имеют выравнивание:

```

gatagg-----cattcataagtttgaaatagctgtccttattctggaact
|| |||      ||||| ||| | | ||||| |||
gaaaggattatcattcataaggaggagagcaagggccttattct-----t

tggatttatggctcttgataggcattcataagtttgattatagctgtcctt
||||| ||| ||| ||| ||| ||| |||
tggatttatttctcttaatagg-attcatgagtttgaaatag

```

Применим для них алгоритм Вперед-Назад и построим матрицу, отражающую величину вероятности сопоставления (рис.7.6) Хорошо видны участки с надежным выравниванием и переходные области около делеций и больших замен.

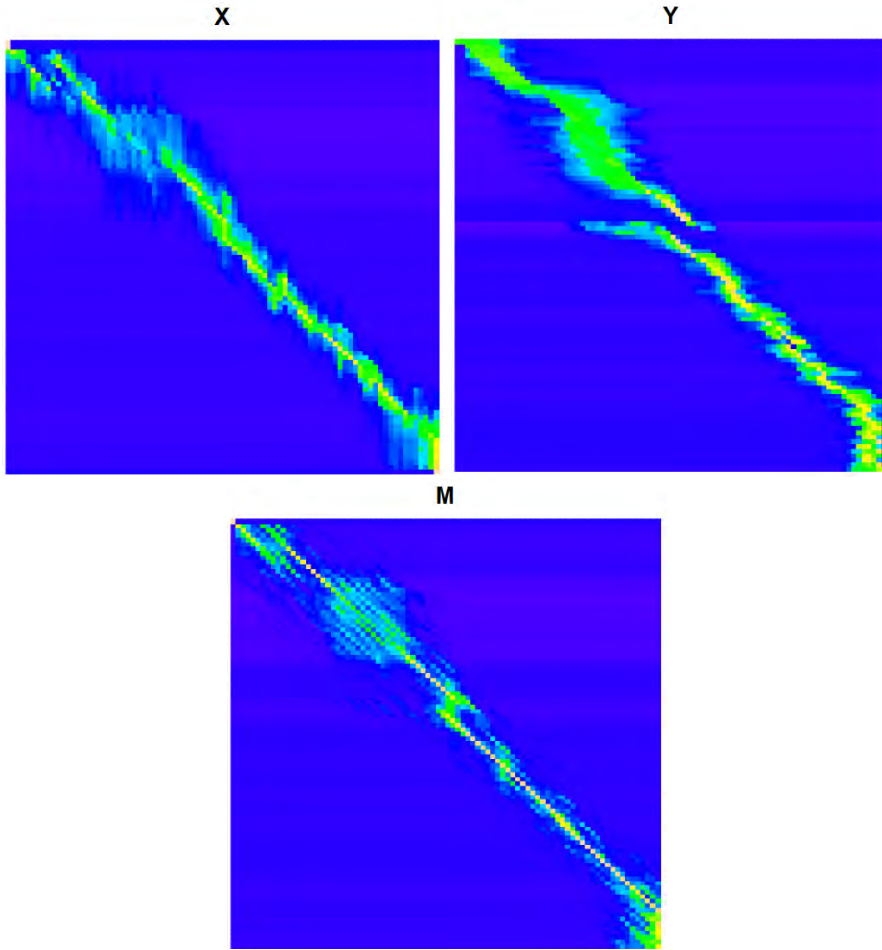
7.4.2 Субоптимальное выравнивание

Алгоритм Витерби, равно, как алгоритм Нидльмана-Вунша, дают единственное решение, которое часто зависит от параметров. Поэтому интересно было бы научиться генерировать несколько почти оптимальных решений. Здесь можно использовать результаты алгоритма Вперед. В стандартном динамическом программировании при восстановлении выравнивания на обратном проходе мы детерминировано выбираем переходы, которые были запомнены при прохождении вперед. Теперь же мы попробуем строить обратный проход стохастически, т.е. будем выбирать каждый шаг обратного хода методом Монте-Карло. Итак, пусть на каком-то шаге алгоритма мы находимся в вершине графа (рис.7.7), например, в вершине M . Вероятность того, что путь дойдет до этой вершины равна сумме вероятностей по всем путям, приводящие в предыдущие вершины, умноженных на соответствующие переходные вероятности и эмиссии:

$$\Pr(x_1 \cdots x_i, y_1 \cdots y_j, \pi(i) = M) = f^M(i, j) = p(x_i, y_j) \sum_l a_{lM} f_l(i-1, j-1);$$

$$\Pr(x_1 \cdots x_i, y_1 \cdots y_j, \pi(i) = X) = f^X(i, j) = q(x_i) \sum_l a_{lX} f_l(i-1, j);$$

$$\Pr(x_1 \cdots x_i, y_1 \cdots y_j, \pi(i) = Y) = f^Y(i, j) = q(y_j) \sum_l a_{lY} f_l(i, j-1);$$

Рис. 7.6: матрица вероятностей для состояний X, Y, M

Подставляя переходные вероятности, получаем вероятности способов входа в состояние M :

$$\begin{aligned}
 \Pr(M \rightarrow M)(i, j) &= p(x_i, y_j) \frac{(1 - 2\delta - \tau)f^M(i - 1, j - 1)}{f^M(i, j)}; \\
 \Pr(X \rightarrow M)(i, j) &= p(x_i, y_j) \frac{(1 - \varepsilon - \tau)f^X(i - 1, j - 1)}{f^M(i, j)}; \\
 \Pr(Y \rightarrow M)(i, j) &= p(x_i, y_j) \frac{(1 - \varepsilon - \tau)f^Y(i - 1, j - 1)}{f^M(i, j)};
 \end{aligned} \tag{7.5}$$

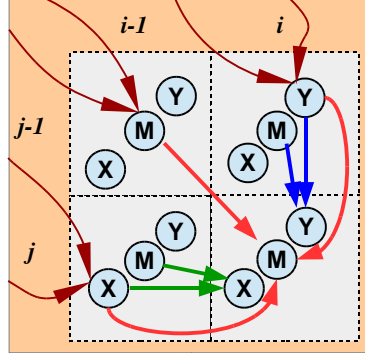


Рис. 7.7: Вероятностные переходы. Красные стрелки – пути, приводящие в состояние M , зеленые – в состояние X , синие – в состояние Y

Аналогично для входа в состояния X, Y :

$$\begin{aligned}
 \Pr(M \rightarrow X)(i, j) &= q(x_i) \frac{\delta f^M(i-1, j)}{f^X(i, j)}; \\
 \Pr(X \rightarrow X)(i, j) &= q(x_i) \frac{\varepsilon f^X(i-1, j)}{f^X(i, j)}; \\
 \Pr(M \rightarrow Y)(i, j) &= q(y_j) \frac{\delta f^M(i, j-1)}{f^Y(i, j)}; \\
 \Pr(Y \rightarrow Y)(i, j) &= q(y_j) \frac{\varepsilon f^Y(i, j-1)}{f^Y(i, j)};
 \end{aligned} \tag{7.6}$$

Наконец, для состояния E :

$$\begin{aligned}
 \Pr(M \rightarrow E)(i, j) &= \frac{(1 - 2\delta - \tau) f^M(i-1, j-1)}{f^M(i, j)}; \\
 \Pr(X \rightarrow E)(i, j) &= \frac{(1 - \varepsilon - \tau) f^X(i-1, j-1)}{f^M(i, j)}; \\
 \Pr(Y \rightarrow E)(i, j) &= \frac{(1 - \varepsilon - \tau) f^Y(i-1, j-1)}{f^M(i, j)};
 \end{aligned} \tag{7.7}$$

Алгоритм заключается в следующем. В начале находимся в состоянии E . Разыгрываем случайную величину, равномерно распределенную на интервале $[0, 1]$ и с использованием вероятностей (7.7) выбираем переход – также как это делалось при генерации последовательности в линейной НММ. Осуществляем переход согласно сделанному выбору и оказываемся в новом состоянии. Далее, если на некотором шаге алгоритма мы оказываемся в состоянии M , то для выбора следующего шага используем вероятности (7.5). Если оказались в состоянии X или Y , то используем формулы (7.6).

```

1  state = E
2  i=n; j=m;
3  do{
4      next_State = random_select(formulas (state))
5      store(i,j)
7      if(next_State == M) {i=i-1; j=j-1;}
8      if(next_State == X) {i=i-1;}
9      if(next_State == Y) {j=j-1;}
10     state=next_State;
11 }while(next_State != B)

```

Пояснения к коду.

строки 1, 2 – инициализация. Мы встаем в конец обеих последовательностей и устанавливаем текущее состояние **state=E**.

строка 4 – определяем следующее состояние. При этом используем соответствующие формулы для вероятностей (**random_select(formulas (state))**).

Строка 5 – запоминаем текущие позиции, либо в выравнивание добавляем соответствующие символы. Далее, в зависимости от того, в какое следующее состояние попали, изменяем индексы i, j .

строка 11 – цикл продолжается до того, пока не встретится состояние B

7.5 Локальное выравнивание

Предыдущая модель предполагала глобальное выравнивание – модель имеет состояния M, X, Y , которые не предполагают независимой генерации последовательностей в начале и в конце. Эту ситуацию можно легко поправить, воспользовавшись конструктором НММ и построить полную модель из под-моделей. Действительно, при локальном выравнивании мы генерируем две независимые последовательности, потом переходим в генерацию выравнивания, затем возвращаемся к генерации независимых последовательностей (рис.7.8А). Для модели независимой генерации используем модель (рис.7.3А). Для генерации выравнивания используем модель (рис.7.1). В результате получим модель (рис.7.8В). Модель (рис.7.8В) содержит параметр η , который отвечает за ожидаемую длину невыровненных участков, и в комбинации с параметром τ определяет соотношение длин выровненных и не выровненных участков.

Часто белки внутри себя содержат участки, которые не следует выравнивать и они соответствуют модели независимого порождения последовательностей. Такими участками являются, например, междоменные линкеры и некоторые петли, которые не выравниваются при сравнении структур. Тогда можно сконструировать модель из того, что есть, воспользовавшись

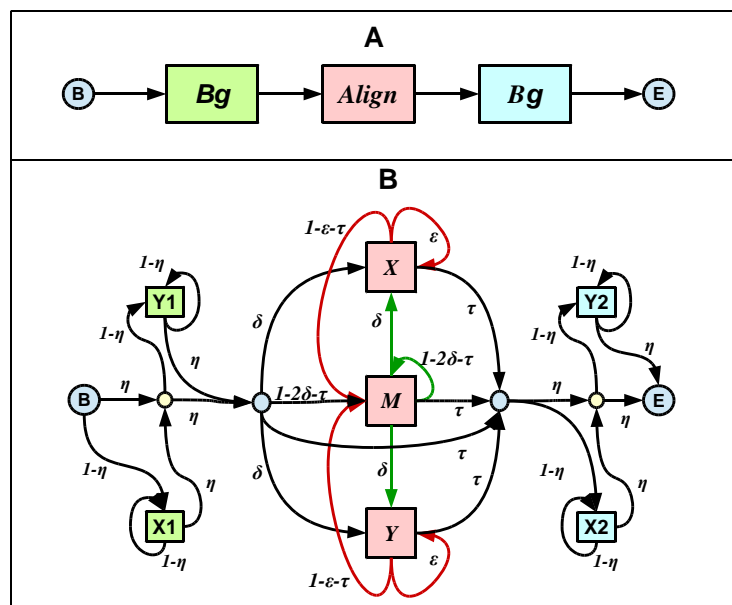


Рис. 7.8: Модель локального выравнивания. А – составление модели из конструктора; В – детальная модель

конструктором: мы поочередно генерируем выровненные и не выровненные участки (рис.7.9А). Другая модификация для тех же целей предполагает переход от генерации только последовательности x , в независимую генерацию последовательности y (рис.7.9В).

Упражнение Напишите рекурсии Вперед-Назад для моделей локального выравнивания (рис.7.8А,В)

Упражнение Напишите рекурсии Витерби для модели (рис.7.9) для максимизации правдоподобия и для максимизации логарифма отношения правдоподобия в сравнении со моделью независимого порождения.

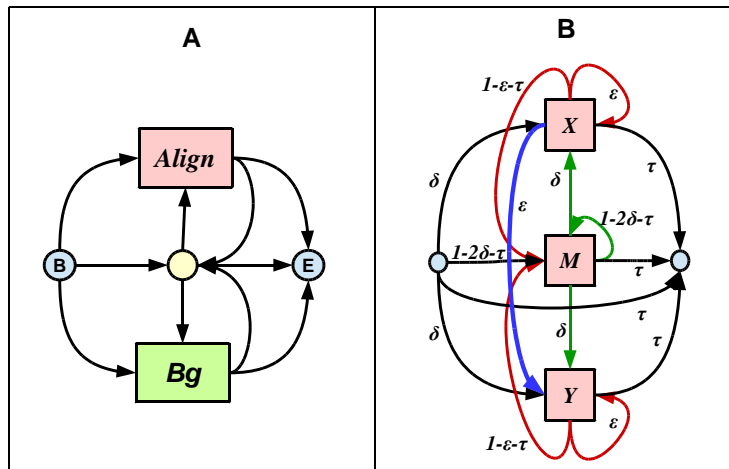


Рис. 7.9: Модель локального выравнивания с возможностью порождения невыравниваемых участков. А – Создание модели с помощью конструктора; В – Модификация модели выравнивания. Здесь добавлен один возможный переход из состояния *X* в состояние *Y* (синяя стрелка)

Глава 8

Профили

8.1 Вероятность, энтропия, информация

8.1.1 Комбинаторная энтропия

Немного термодинамики. Пусть есть несколько ящиков (L), в которых мы наблюдаем некоторое количество частиц (рис.8.1). Например, разобьем пространство, например, комнату на ячейки размером $1\text{см} \times 1\text{см}$. Подсчитаем количество молекул в каждом ящике, получим числа заполнения n_i . Полное количество молекул равно $N = \sum n_i$. Набор чисел $\{n_i\}$ заполнения будем называть макросостоянием, а детальное распределение – микросостоянием системы. Посчитаем количество способов получить данное макросостояние, т.е. определим число микросостояний, отвечающих данному макросостоянию. Вспоминая комбинаторику, получаем:

$$N(\text{micro} | \text{macro}) = \frac{N!}{n_1! \cdot \dots \cdot n_L!}$$

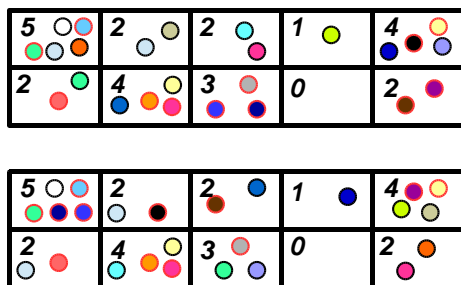


Рис. 8.1: Распределение частиц по ящикам. Показаны два микросостояния, соответствующие одному макросостоянию. Числа заполнения на верхней и нижней картинке получены разными наборами частиц.

По определению логарифм этой величины называется энтропией макросостояния системы:

$$S = \ln \frac{N!}{n_1! \cdot \dots \cdot n_L!} = \ln N! - \sum \ln n_i!$$

Для того, чтобы эту формулу упростить вспомним формулу Стирлинга:

$$n! \simeq \left(\frac{n}{e}\right)^n$$

Тогда, заменяя факториалы по формуле Стирлинга, и принимая во внимание, что $N = \sum n_i$ получаем:

$$\begin{aligned} S &= N \ln N - \cancel{N} - \sum n_i \ln n_i + \cancel{\sum n_i} = N \ln N - \sum n_i \ln n_i \\ &= (n_1 + \dots + n_L) \ln N - n_1 \ln n_1 - \dots - n_L \ln n_L \\ &= n_1 \ln(\ln N - \ln n_1) + \dots + n_L (\ln N - \ln n_L) \\ &= - \sum n_i \ln \frac{n_i}{N} = -N \sum \frac{n_i}{N} \ln \frac{n_i}{N} = -N \sum f_i \ln f_i \end{aligned}$$

где $f_i = n_i/N$ – частота заполнения. Итак, величина

$$S = -N \sum f_i \ln f_i \quad (8.1)$$

называется энтропией макросостояния. Заметим, что если для какого-то состояния количество частиц, а, следовательно и частота равна 0, то вклад в сумму будет равен 0, поскольку $\lim_{x \rightarrow 0} x \ln x = 0$. Так определенная энтропия обладает рядом важных свойств. Например, она аддитивна, т.е. если систему разбить на две подсистемы, то энтропия всей системы будет равна сумме энтропий подсистем.

Отступление про физику. Все мы знаем (или слышали про) второе начало термодинамики – в *замкнутой* системе энтропия (8.1) стремится стать максимальной (теорема Больцмана). Однако, все не так просто и здесь есть несколько противоречий. Одно из них такое. В замкнутой системе (для простоты, дискретной) траектория в фазовом пространстве обязательно вернется в исходную точку. Чтобы это объяснить, возьмем, для примера, кубик Рубика. Для любого состояния кубика можно ввести энтропию, используя степень разнообразия граней (аналог ящиков). Для каждой грани s мы знаем число цветов на этой грани n_c^s . Для каждого состояния можно ввести энтропию

$$S = - \sum_{s,c} \frac{n_c^s}{9} \ln \frac{n_c^s}{9}$$

Здесь мы делим на 9, поскольку на каждой грани 9 клеточек. Впрочем это деление не принципиально. Пусть в начальный момент кубик собран. Его энтропия равна 0. Теперь начнем его случайно вращать. Понятно, что

энтропия будет возрастать, а потом будет колебаться около какого-то значения (около 4). Это вполне соответствует нашей интуиции – при случайных вращениях энтропия (беспорядок) растет. Однако количество состояний кубика – конечно. Поэтому при случайных вращениях мы, рано или поздно, *случайно* окажемся в состоянии собранного или почти собранного кубика с энтропией около 0. Итак, энтропия возрастает, но может оказаться равной исходной. Но надо понимать, что в больших системах вероятность того, что энтропия станет равной исходной весьма мала.

Другое противоречие. Все физические уравнения классической физики, описывающие микросостояния обратимы во времени. Например, уравнения движения Ньютона не изменятся при замене $t \rightarrow -t$. Поэтому энтропия должна расти как при движении в будущее, так и при движении в прошлое. Т.е. классическая физика не определяет стрелы времени, а энтропия – определяет. Это верно и для уравнений теории относительности, в том числе и Общей, и для квантовой механики.

Поэтому энтропия это весьма нетривиальное понятие.

8.1.2 Энтропия и информация

В энтропии используется натуральный логарифм, а при определении информации используют логарифм по основанию 2. Переход от одного основания к другому означает только изменение масштаба. Если есть источник информации, передающий данные, то его энтропия, по определению, равна:

$$H_{source} = - \sum_{\alpha} P(\alpha) \log_2 P(\alpha)$$

где $P(\alpha)$ – вероятность генерации символа.

- Энтропия источника сообщения – степень неопределенности при генерации символов
- Энтропия аддитивна: энтропия неопределенной последовательности X равна сумме энтропий позиций: $H(X) = \sum_i H_i = NH$
- Энтропия максимальна, если все символы равновероятны

Итак, мы получили сообщение X . Каково его информационное содержание (не путать с информативностью!). Информационное содержание – это насколько понизилась неопределенность, т.е. это потеря энтропии:

$$I(X) = H_{before} - H_{after} = - \sum_i P(x_i) \log_2 P(x_i)$$

Второй член в этом равенстве равен 0, поскольку после получения сообщения у нас пропала неопределенность. Например, информационное содержание нуклеотидной последовательности `aagaattac` при равновероятном появлении букв равна $H(X) = -9 \cdot 4 \cdot 0.25 \cdot \log_2 0.25 = -9 \cdot -2 = 18$. Здесь 9

— это длина сообщения, 4 — размер алфавита. Поскольку все буквы равновероятны, то сумма эквивалентна умножению на 4.

Вообще, при равновероятном появлении букв энтропия равна $-\log_2 p_\alpha$ и в нашем случае она равна 2 бита на символ. Теперь, допустим, мы получили не полное сообщение, а на 5-й позиции прочитали неточно — а или g. Чему равно информационное содержание в таком сообщении? Энтропия источника H_{before} не изменилась, однако энтропия полученного сообщения H_{after} не равна 0, так как осталась неопределенность в 5-й букве. Поскольку вероятность появления каждой буквы (а или g) в 5-й позиции равна 0.5, имеем $H_{after} = 0.5 \log_2 0.5 + 0.5 \log_2 0.5 = 1$. Поэтому информационное содержание этого сообщения будет равно $18 - 1 = 17$.

8.1.3 Взаимная информация

В предыдущем анализе мы считали все ящики равновероятными. Пусть теперь мы знаем распределение вероятностей попасть в тот или иной ящик $\{p\}$. Тогда вероятность макросостояния будет иметь полиномиальное распределение:

$$P(n_1 \cdots n_N | p_1 \cdots p_L) = \frac{N!}{n_1! \cdots n_L!} p_1^{n_1} \cdots p_L^{n_L}$$

Логарифм этого распределения дает нам, по определению, взаимную энтропию.

$$\begin{aligned} S(\{n_i\} | \{p_i\}) &= \ln P(\{n_i\} | \{p_i\}) = \ln \left(\frac{N!}{\prod n_i!} p_1^{n_1} \right) \\ &= \ln \left(\frac{N!}{\prod n_i!} \right) + \sum n_i \ln p_i \\ &= \ln \left(\frac{N!}{\prod n_i!} \right) + N \sum f_i \ln p_i \end{aligned}$$

Первый логарифм мы уже вычисляли. Он равен $-N \sum f_i \ln f_i$. Поэтому получаем:

$$S(\{n_i\} | \{p_i\}) = -N \sum f_i \cdot \ln \frac{f_i}{p_i}$$

При определении информации обычно используют двоичный логарифм. Переход от натуральных логарифмов к двоичным приводит к изменению масштаба. Кроме того, при определении информации рассматривают удельную энтропию

$$Inf(\{f_i\} | \{p_i\}) = -S/(N \ln 2) = \sum f_i \cdot \log_2 \frac{f_i}{p_i}$$

Эта величина весьма широко применяется в биоинформатике и характеризует, насколько наблюдаемые частоты соответствуют ожидаемым. Поскольку значения $\{f_i\}$ можно интерпретировать как распределение, то эта величина показывает, насколько распределение $\{f_i\}$ отличается от распределения $\{p_i\}$. Очевидно, что при $\{f_i\} = \{p_i\}$ информационное содержание

$Inf(\{n_i\} | \{p_i\})$ равно 0. Нетрудно показать, что эта величина неотрицательна. Для этого можно попробовать найти ее экстремум $Inf(\{n_i\} | \{p_i\}) \rightarrow \max_f | \sum f_i = 1$. Применим метод неопределенных множителей Лагранжа:

$$\begin{aligned}\Phi &= \sum f_i \cdot \ln \frac{f_i}{p_i} - \lambda(\sum f_i - 1) \\ \frac{\partial \Phi}{\partial f_k} &= \ln f_k + f_k \frac{1}{f_k} - \ln p_k - \lambda = 0 \\ \ln f_k - \ln p_k &= \lambda - 1; \quad \frac{f_k}{p_k} = \exp(\lambda - 1); \quad f_k = p_k \cdot \exp(\lambda - 1)\end{aligned}$$

Применяя ограничение $\sum f_i = 1$, получим окончательно $f_k = p_k$, т.е. экстремум достигается только, если распределения совпадают. Можно легко показать, что это минимум. Таким образом, мы показали, что взаимная информация равна:

- $Inf(\{n_i\} | \{p_i\}) \geq 0$
- $Inf(\{n_i\} | \{p_i\}) = 0 \iff f_i = p_i$

Расстояние Кульбака-Лейблера

Поскольку взаимная информация неотрицательна и равна 0 тогда и только тогда, когда распределения совпадают, взаимную информацию часто называю *расстоянием Кульбака-Лейблера*. На самом деле это не совсем расстояние, поскольку эта величина не симметрична: $d_{KL}(f, p) \neq d_{KL}(p, f)$. Эту величину можно симметризовать, что тоже часто делают:

$$d_{KL}^{sym}(f, p) = \frac{1}{2}(d_{KL}(f, p) + d_{KL}(p, f))$$

Это уже настоящее расстояние. Расстояние Кульбака-Лейблера широко используется в анализе данных. Так, например, весьма популярный метод понижения размерности и визуализации данных t-SNE имеет в своей основе именно это расстояние.

8.2 Профили

Пусть нам дано некоторое множественное выравнивание, отражающее некую общую функцию. Например, это может быть множество сайтов связывания транскрипционных факторов или последовательностей белков, принадлежащих тому или иному семейству. Нам надо выделить особенности в этом наборе последовательностей, чтобы иметь возможность потом их использовать для предсказания свойств других последовательностей. Такой набор признаков называется *профилем* группы последовательностей. В построении профилей могут быть использованы разнообразные подходы. Пусть, например, нам дан фрагмент множественного выравнивания (рис.8.2). Есть несколько способов описать этот набор.

```

GAAAACGTTTGCCTTT
GTAAAGGCAAACGTTT
GCAAACGTTTTCCTTT
GGAAACGTTTTCCTTT
GAAAACGATTGCTTTT
GGAAACGGATTCTTTT
GCAATCGTTCTCTTTT
GAAAACCTTTGTTTTT
GAAAAGGCTTGTCTTT
GCAAACCTTTGCTTTT

```

Рис. 8.2: Пример множественного выравнивания нуклеотидных последовательностей

8.2.1 Консенсус

Простейший, и широко используемый способ описать этот набор – консенсус. Консенсус – это последовательность из наиболее консервативных букв в колонках. Для набора (рис.8.2) консенсус будет такой:

G.AAacg.tt...TTT

где заглавные буквы – абсолютно консервативны, строчные с консервативностью >75%. Этот способ описания очень простой и понятный. Часто в литературе можно встретить выражение типа «сайт описывается консенсусом ...» или «консенсусный символ в позиции ...». Однако такой подход теряет много информации. Вместо него иногда (все реже) используется регулярное выражение.

8.2.2 Регулярное выражение

Этот способ описания был весьма популярен для описания аминокислотных последовательностей. Для нашего случая сайт описывается выражением

G [ACGT] A A [AT] [CG] [GT] [ACGT] [AT] [ACT] [AGT] [CT] [CGT] TTT

Согласно синтаксису регулярного выражения здесь указывается либо буква, которая заведомо встречается в данной позиции, либо в квадратных скобках перечисляются допустимые буквы в соответствующей позиции.

8.2.3 Частотный профиль и правдоподобие

Более детальная информация может быть получена, если мы для каждой колонки укажем частоту появления в ней того или иного символа:

A	0.0	0.4	1.0	1.0	0.9	0.0	0.0	0.1	0.2	0.1	0.1	0.0	...
C	0.0	0.3	0.0	0.0	0.0	0.8	0.0	0.4	0.0	0.1	0.0	0.7	...
G	1.0	0.2	0.0	0.0	0.0	0.2	0.8	0.1	0.0	0.0	0.5	0.0	...
T	0.0	0.1	0.0	0.0	0.1	0.0	0.2	0.4	0.8	0.8	0.4	0.3	...

Можно рассматривать эти частоты в качестве вероятностей для модели выравнивания M . При этом есть некоторая фоновая модель генерации символов R , в которой генерация символов не зависит от позиции $q(\alpha)$. Тогда для любой последовательности можно написать

$$\Pr(x|M) = \prod_{i \in \text{columns}} f^i(x_i)$$

$$\Pr(x|R) = \prod_{i \in \text{columns}} q(x_i)$$

логарифм отношения правдоподобия есть

$$\log \frac{\Pr(x|M)}{\Pr(x|R)} = \sum_{i \in \text{columns}} \log \frac{f^i(x_i)}{q(x_i)}$$

Матрица

$$M(i, \alpha) = \log \frac{f^i(\alpha)}{q(\alpha)}$$

называется *позиционной весовой матрицей* (PWM – Position Weight Matrix), которая является наиболее широко используемой в качестве описания группы последовательностей. Чтобы избежать бесконечностей к частотам добавляют небольшие константы, что соответствуют псевдо-счетчикам. Для нашего случая позиционная весовая матрица будет:

A	-1.8	0.5	1.7	1.7	1.5	-1.8	-1.8	-0.8	-0.2	-0.8	-0.8	-1.8	...
C	-1.8	0.2	-1.8	-1.8	-1.8	1.4	1.4	0.5	-1.8	-0.8	-1.8	1.2	...
G	1.7	-0.2	-1.8	-1.8	-1.8	-0.2	-1.8	-0.8	-1.8	-1.8	0.8	-1.8	...
T	-1.8	-0.8	-1.8	-1.8	-0.8	-1.8	-0.2	0.5	1.4	1.4	0.5	0.2	...

На самом деле позиционную весовую матрицу мы ранее получали из скрытой марковской модели сайтов связывания.

8.2.4 Информационное содержание и Лого

Информационное содержание колонки выравнивания c – взаимная информация наблюдаемых частот встречаемости букв относительно фонового распределения:

$$Inf_c = \sum_{\alpha} f^c(\alpha) \log_2 \frac{f^c(\alpha)}{q(\alpha)}$$

а полное информационное содержание выравнивания является суммой по всем колонкам:

$$Inf_{msa} = \sum_{i \in \text{columns}} \sum_{\alpha} f^i(\alpha) \log_2 \frac{f^i(\alpha)}{q(\alpha)}$$

Полное информационное содержание показывает, насколько распределение наблюдаемых букв отличается от случайного. Однако, если выравнивание тонкое и количество последовательностей невелико, то надо использовать Байесову оценку и добавлять псевдо-счетчики.

Лого Мы можем построить такой график – в каждой позиции и для каждой буквы откладываем по вертикали величину вклада этой буквы в информационное содержание: $f^i(\alpha) \log_2 f^i(\alpha)/q(\alpha)$. Такое графическое представление информационного содержания называется sequence LOGO (рис.8.3)



Рис. 8.3: Лого для выравнивания на рис.8.2. Высота каждой буквы соответствует вкладу соответствующего символа в информационное содержание.

8.3 НММ профиль

В главе, посвященной скрытым марковским моделям мы строили модель для сайтов связывания транскрипционных факторов. Эту модель можно использовать для любых (нуклеотидных и аминокислотных) последовательностей при условии, что нет вставок/делеций. Однако мы хорошо знаем, что выравнивания часто имеют и вставки и делеции (рис.8.4).

```

PDASFDEIKHSYRKALQYHFDKNINDP---EANEKFQKINEAYQVLSDENRRKMYDE-CGMKAT
ANSKLEIEKEKYEVASKYHFEKNIGND---KAFKKFELINSAYQILSNEELRRKYNSDCRSKMN
PTSELSIEIKSNYYNLALKYNFESNLGNA---EALTKFRDINEAYQILSLDQRREAYDRTCKFSAP
RDCTTNLKKAYRKLAAMWHPDKHNDEKSKKEAEKFKNIAEAYDVLADEEKRKIYDTYCEEGLK
RDCTNEDIKKAYKKLAMKWHPDKHLNAASKKEADNMFKSISEAYEVLSDDEEKRDYDKYCEEGLD
VNADMNEITERYFKLAENIYFYQRRSGS---TVFHNFRKVMNEYQVLCIDKRWYNKY-CYDGIK
VDADGDTIKKSYRRILAILYHFDKNRENTP---EAAEFQKLAEAYQVLSDPKLRKYDKLCKVGAU
PTATESIEIKKAYYIKARQVHFDKNPNPDP---QAAHNFQVLGEAYQVLSDSGQRQAFDACCKSGIS
PSASEEIEIRKAYYIKARQVHFDKNQGDTP---LAA-EKQVLGEAYQVLSDPVHREAYDRTCKFSAP
KNCTTDEVKKAYRKLAIIHHPDKG-GDP-----EKFKIEISRAYEVLSDDEEKKLYDEYCEEGL
AYASKTDIQQSFRKMSRIYHFDKNKEP-----DSLDRFNKIREAYEVLSDNKKKYTYDRFCDFGDS

```

Рис. 8.4: Пример множественного выравнивания со вставками и с делециями.

Для такого случая можно построить модель, которая допускает вставки в последовательность (добавлять символы, которые не описаны в профиле) и делеции в последовательности – пропускать некоторые позиции в профиле. Поэтому в модель надо добавить состояния, соответствующие вставкам и делециям (рис.8.5). В этой модели есть состояния M_i – генерация символа в последовательности, в соответствии с профилем; I_i , которые означают вставку в последовательность – генерацию неопределенного числа символов, не входящих в профиль, а также состояния D , которые не генерируют символы, а позволяют пропустить несколько символов в профиле.

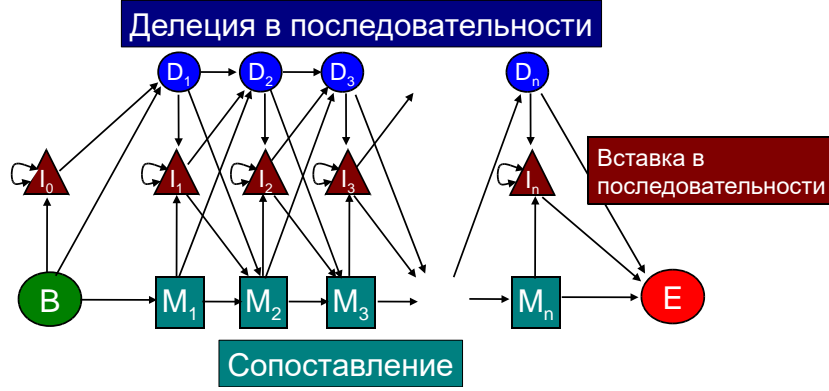


Рис. 8.5: Скрытая марковская модель профиля со вставками и с делециями.

Если у нас определен такой профиль, то можно «натянуть последовательность» x на профиль. Поскольку у нас есть делеции и вставки, то это будет, фактически, выравниванием последовательности с профилем и для построения выравнивания можно использовать динамическое программирование. Обратим внимание, что в отличие от парного выравнивания, здесь все переходные и эмиссионные вероятности зависят от позиции профиля.

Итерация Витерби для логарифма отношения правдоподобия в этом случае выглядит так:

$$V_j^M(i) = \log \frac{e_j^M(x_i)}{q_{x_i}} + \max \begin{cases} V_{j-1}^M(i-1) + \log a_{j-1}^{MM} \\ V_{j-1}^I(i-1) + \log a_{j-1}^{IM} \\ V_{j-1}^D(i-1) + \log a_{j-1}^{DM} \end{cases} \quad (8.2)$$

$$V_j^I(i) = \log(q_{x_i}/q_{x_i}) + \max \begin{cases} V_j^M(i-1) + \log a_j^{MI} \\ V_j^I(i-1) + \log a_j^{II} \\ V_j^D(i-1) + \log a_j^{DI} \end{cases} \quad (8.3)$$

$$V_j^D(i) = \max \begin{cases} V_{j-1}^M(i) + \log a_{j-1}^{MD} \\ V_{j-1}^I(i) + \log a_{j-1}^{ID} \\ V_{j-1}^D(i) + \log a_{j-1}^{DD} \end{cases} \quad (8.4)$$

Здесь мы, во-первых, считали, что эмиссионные вероятности для состояния I совпадают с фоновыми, поэтому во втором уравнении нет вклада эмиссии, поскольку $\log(q_{x_i}/q_{x_i}) = 0$. Во-вторых, индексы при переходных вероятностях указывают на позицию в профиле, *откуда* происходит переход. Переходы $I \rightarrow D$ и $D \rightarrow I$ отвечают случаям, когда вставка в последовательность происходит сразу после делеции и наоборот когда делеция происходит сразу после вставки. Такие переходы представляются крайне

маловероятными, поэтому третью строку в уравнении (8.3) и вторую строку в уравнении (8.4) можно опустить. Рекурсии (8.2 – 8.4) очень похожи на рекурсии для выравнивания. Однако еще раз подчеркнем отличия от обычного выравнивания:

- Выравнивание с профилем не симметрично, т.е. делеция и вставка устроены по-разному, поскольку одна последовательность – это профиль, а вторая последовательность – обычная последовательность. Делеция означает пропуск некоторых позиций в профиле, а вставка – это добавление символов, которые не описаны в профиле.
- Эмиссионные вероятности зависят от позиций и не имеют отношения к матрице замен аминокислотных остатков
- Переходные вероятности зависят от позиций. Например, делеция важной позиции в профиле имеет маленькую (вплоть до нулевой) вероятность, в то время как некоторые позиции можно достаточно безбоязненно пропускать.

НММ Профили обычно применяются для описания семейств белков. В частности, в базе данных rFam для описания семейств белков используются такие профили и оценка принадлежности нового белка тому или иному семейству производится с помощью выравнивания профиля из базы данных с пользовательской последовательностью

Упражнение. Напишите рекурсии для алгоритма Вперед-Назад.

8.3.1 Оценка параметров

Параметры профильной НММ оцениваются с помощью построенных множественных выравниваний, см., например, рис.8.6. Оценка эмиссионных вероятностей в состоянии M делается оценкой частот встречаемости аминокислотных остатков с Байесовой поправкой. Псевдосчетчики и другие поправки будут обсуждены ниже.

Вероятности переходов в состояния I и D делается по частоте вставок и делеций в позициях выравнивания, по которому строится профиль. При этом в тех позициях, где делеции не наблюдались, ставятся предельно низкие вероятности соответствующих переходов ($M \rightarrow D$, $M \rightarrow I$). В тех позициях, где небольшая доля последовательностей имеет пропуски, мы считаем, что здесь допустимы переходы $M \rightarrow D$, а если в позициях большая доля пропусков, то мы считаем возможным поставить здесь переход $M \rightarrow I$. Вероятности переходов ставятся в зависимости от частоты соответствующих событий.

8.3.2 Псевдо-счетчики

Вероятности в профилях оцениваются как частоты встречаемости букв. Однако, если выравнивание тонкое, то такие оценки сделать нельзя. Более

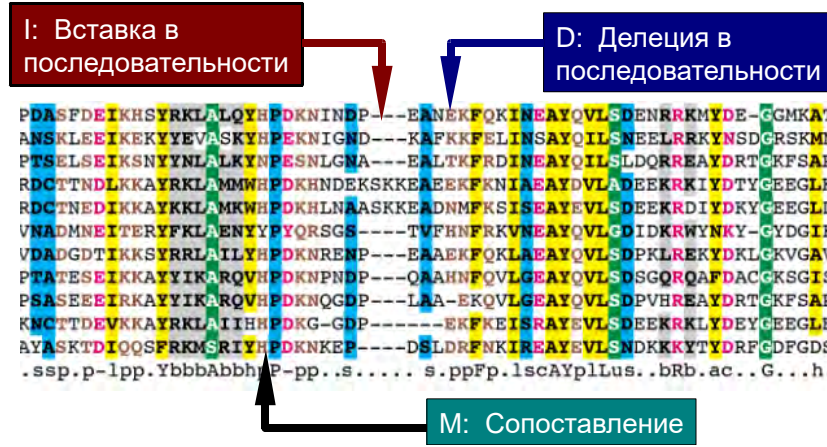


Рис. 8.6: Скрытая марковская модель профиля со вставками и с делециями.

того, если, например, выравнивание аминокислотных последовательностей тоньше 20, то заведомо некоторые аминокислотные остатки в принципе не могут в нем появиться. Например в выравнивании на рис.8.4 в колонке 4 мы видим остатки FLLTNMGEETK. Значит ли это, что здесь не может встретиться аргинин (R) или изолейцин (I)? Разумеется – нет. Для того, чтобы учесть такую возможность применяют Байесов подход, а именно к наблюдениям добавляют псевдосчетчики:

$$e_i(\alpha) = \frac{E_i^\alpha + \psi_i^\alpha}{\sum_{\alpha} E_i^\alpha + \psi_i^\alpha}$$

где E_i^α – количество наблюдаемых букв α в колонке i , ψ_i^α – количество как-бы наблюдений – псевдосчетчики. Псевдосчетчики могут зависеть от колонки и типа буквы, но не обязательно. Возникает вопрос – как нам задать эти псевдосчетчики?

Правило Лапласа. Простейший вариант псевдосчетчиков такой – просто приравняем все псевдосчетчики 1:

$$\psi_i^\alpha = 1$$

Правило Лапласа не обращает внимания ни на тип остатка, ни на колонку. Ясно, что такой подход не совсем подходит, поскольку частоты встречаемости аминокислот разные.

Учет частот встречаемости символов. Второй вариант, чуть более приближенный к реальности – это сделать псевдосчетчики пропорциональными

частотам встречаемости соответствующих символов в мире или в данном наборе последовательностей q_α :

$$\psi_i^\alpha = A \cdot q_\alpha$$

Такой подход использует тип символа, но не зависит от колонки.

Учет матрицы замен. Описанный ниже подход не имеет строгой вероятностной интерпретации, но в большой степени соответствует здравому смыслу. Действительно, если в колонке встретился, скажем изолейцин, то мы вправе ожидать в той же колонке родственную аминокислоту – валин, а также лейцин и метионин (это все алифатические разветвленные остатки). А вот встретить там же аргинин (положительно заряженный остаток) – значительно менее вероятно. Если же встретился аргинин, то там же ожидается лизин. Иными словами, изолейцин – немного валин и лейцин, а аргинин – немного лизин. Поэтому будем считать псевдосчетчики как сумму условных вероятностей встретить остаток b при уловии, что мы уже видим остаток a . Каждый остаток в колонке выравнивания породит некоторое множество вероятностей для всех остатков. Просуммируем эти вероятности:

$$\psi_i^\alpha = A \sum_{\beta \in column_i} \Pr(\alpha|\beta)$$

Здесь псевдосчетчики в разных колонках уже разные. Вероятности $\Pr(\alpha|\beta)$ можно оценить по большому набору стандартных выравниваний, так же, как это далось при получении матриц замен аминокислотных остатков. В большинстве случаев $\Pr(\alpha|\alpha) \gg \Pr(\alpha|\beta \neq \alpha)$. Поэтому основной вклад будет все равно от того же остатка. Методы введения псевдосчетчиков зависят от некоторой константы A . Для выбора этих констант используют разные соображения, приводящие к разным значениям. Обычно эту величину для второго метода (учет фоновых частот) принимают равной около 20 – чтобы соответствовать правилу Лапласа (почему?). Поскольку в последнем методе (учет матрицы замен) вклад $\Pr(\alpha|\alpha)$ будет превалировать, то это означает, что к уже наблюдаемому присутствию остатка $\Pr(\alpha|\alpha)$ мы еще добавляем его же, только с меньшим весом. Поэтому здесь оправдано использование достаточно большого значения константы A .

8.3.3 Взвешивание последовательностей

Рассмотрим колонку выравнивания (рис.8.7). Наивная оценка частот аминокислот (забудем пока про псевдосчетчики) показывает, что в этой колонке встречается в основном лейцин и изолейцин. Остальные аминокислотные остатки – минорны. Однако можно заметить, что в выравнивании сильно перепредставлены млекопитающие и особенно приматы. Поэтому частоты встречаемости аминокислотных остатков скорее характеризует свойства этого семейства белков для млекопитающих, чем общие свойства семейства. Иными словами, выборка последовательностей не сбалансирована.

<i>Homo</i>	L	Частоты встречаемости
<i>Chimp</i>	L	
<i>Macaque</i>	L	
<i>Dog</i>	I	
<i>Cow</i>	L	
<i>Chicken</i>	I	
<i>Lizard</i>	V	
<i>Fly</i>	M	
<i>Worm</i>	C	
<i>Yeast</i>	R	

Рис. 8.7: Колонка выравнивания

Простейший вариант балансировки выборки – это исключить некоторые последовательности. Например, оставить только одного представителя млекопитающих. Но при этом все равно останется перекося в сторону позвоночных, что тоже, наверное, неправильно.

Частоты встречаемости оцениваются как число наблюдений деленное на число испытаний: $f = n/N$. Число наблюдений – это просто сумма единиц: $n = \sum 1$. Давайте сбалансируем выборку, введя веса последовательностей. Те последовательности, которые перепредставлены будут иметь меньший вес, а те, которые недопредставлены – больший вес. Таким образом число наблюдений заменится:

$$n = \sum_{i=1}^n 1 \Rightarrow n = \sum_{i=1}^n \omega_i$$

где ω_i – вес последовательности номер i . Задав разумные веса последовательностей, мы получим совсем другие, скорректированные частоты (рис.8.8). Остается вопрос – что такое разумные веса? Для оценки разумных весов

Вид		Вес	Частоты встречаемости
<i>Homo</i>	L	0.3	
<i>Chimp</i>	L	0.3	
<i>Macaque</i>	L	0.4	
<i>Dog</i>	I	0.5	
<i>Cow</i>	L	0.5	
<i>Chicken</i>	I	0.9	
<i>Lizard</i>	V	0.9	
<i>Fly</i>	M	1.8	
<i>Worm</i>	C	1.8	
<i>Yeast</i>	R	2.7	

Рис. 8.8: Колонка выравнивания с весами

есть несколько эвристик. Они не имеют строгого математического обоснования, но единственное, что про них можно сказать — это их достаточная интуитивность. Здесь мы рассмотрим два подхода.

Многогранники Вороного

Концепция этого объекта достаточно простая. Представим себе, что на необитаемую землю пришли поселенцы. Первым делом они построили хижины в разных местах. Потом возник вопрос о границах. Решили, что границы между участками будут проводить по середине между хижинами. Вспоминая 6-й класс средней школы, понимаем, что заборы надо строить вдоль прямой, перпендикулярной отрезку, соединяющему хижины, и проходящей через середину этого отрезка. Построив заборы, получили некоторое разбиение. Такое разбиение называется многогранниками Вороного. Ясно, что

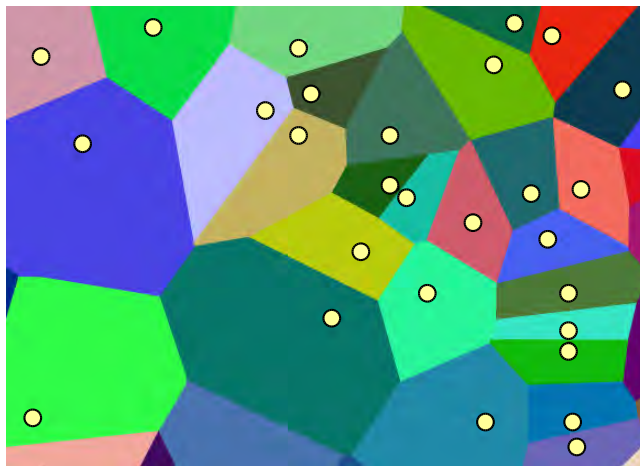


Рис. 8.9: Многогранники Вороного.

там, где плотность населения выше, там размер помещений меньше. Это является хорошей идеей для того, чтобы определить веса как площадь помещения, но как поместить на плоскость (или в иное линейное пространство) последовательности? Идея многогранников Вороного может быть обобщена на метрическое пространство.

Метрическое пространство. Метрическим пространством называется множество Ω , на котором определена функция $d : \Omega \otimes \Omega \rightarrow R^+$, ставящее в соответствие каждой паре объектов неотрицательное число, называемое расстоянием. При этом должны выполняться аксиомы:

- расстояние симметрично: $d(a, b) = d(b, a)$

- расстояние равно 0 тогда и только тогда, когда объекты совпадают:
 $d(a, b) = 0 \Leftrightarrow a = b$
- должно выполняться неравенство треугольника:
 $\forall a, b, c : d(a, b) \leq d(a, c) + d(c, b)$

На метрическом пространстве тоже можно определить многогранники Вороного. Если на всем пространстве мы выделим подмножество $A \subset \Omega$, то для каждого элемента $a \in A$ мы можем определить «поместье» $S(a)$ как множество элементов из Ω , которые ближе к a , чем к другим элементам множества A :

$$S(a) = \{\omega \in \Omega\} : \forall b \in A \quad d(b, \omega) > d(a, \omega)$$

Эта концепция широко используется в анализе данных, в частности, в задачах кластеризации. Примеров таких метрических пространств можно привести много. В частности, каждый человек несет в своем геноме множество полиморфизмов. Можно определить расстояние между двумя особями, как количество различающихся полиморфизмов. Таким примером может служить, в частности, множество всех последовательностей. Для любой пары последовательностей можно вычислить вес выравнивания. Но вес выравнивания не есть расстояние, поскольку чем ближе последовательности, тем выше вес. Преобразуем вес выравнивания в расстояние. Для каждого выравнивания можно определить уровень похожести:

$$Q = \frac{2 \cdot w(x, y)}{w(x, x)w(y, y)}$$

где $w(x, y)$ – вес оптимального выравнивания последовательностей x, y . Величина Q заключена в интервале $[0, 1]$. Если теперь взять от нее минус логарифм, получим расстояние:

$$d(x, y) = -\log Q = -\log \frac{2 \cdot w(x, y)}{w(x, x)w(y, y)}$$

Эта величина неотрицательна и заключена в интервале $[0, \infty]$. Теперь, имея расстояния хорошо бы для каждой последовательности из данного множества последовательностей, которые принадлежат нашему множественному выравниванию, вычислить размер поместья. Перебрать все возможные последовательности не представляется возможным. Для решения этой задачи применим метод Монте-Карло. Будем из какой-то модели генерировать случайные последовательности и определять, к какому помещью они принадлежат, т.е. к каким из данного набора они ближе. Сделав так достаточно много раз, мы получим размер поместий, как число случайных последовательностей, туда попавших.

Одним из способов генерации последовательностей может быть случайный выбор буквы в каждой колонке. Например, случайная последовательность будет выглядеть так – первая буква берется из восьмой последовательности нашего выравнивания, вторая буква – из третьей, третья буква –

из первой последовательности и т.д. Для определения расстояния от сгенерированной последовательности до реальной последовательности в выравнивании не обязательно строить оптимальное выравнивание пары последовательностей, а можно просто смотреть на множественное выравнивание, которое у нас уже есть и вычислять соответствующие веса.

Достоинство этого метода — он не зависит от эволюционного дерева. Недостаток — чтобы оценить веса последовательностей надо сделать довольно много испытаний.

Методы, основанные на эволюционных деревьях

Другие методы явно используют эволюционные деревья. При этом разумно использовать деревья, построенные на тех же последовательностях, поскольку в ряде случаев возможен горизонтальный перенос и дупликация генов. Поэтому история целых организмов здесь вряд ли поможет. Кроме того, нам необходимы не только топология, но и эволюционные расстояния. Итак, пусть для наших последовательностей есть *укорененное* взвешенное дерево последовательностей (рис.8.10). Есть несколько эвристических подходов определения весов, основанных на деревьях.

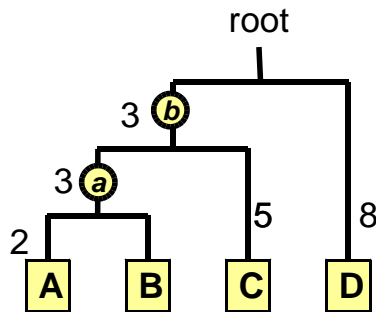


Рис. 8.10: Укорененное взвешенное дерево последовательностей.

Метод Гернштейна-Соннхаммера-Чотьи

Этот метод весьма прост, и, хотя и не имеет математического обоснования, дает вполне разумные результаты. Основная идея метода состоит в том, чтобы распределить длины ветвей по весам последовательностей. На первом шаге веса приравниваются весам листьев. Получаем распределение весов 2:2:5:8. Далее распределяем вес ребра *a* по листьям пропорционально их актуальным весам. Потом распределяем вес ребра *b* по всем дочерним листьям пропорционально их весам. В общем виде на каждом шаге *i* мы

переопределяем веса по формуле:

$$\omega_{sq}^i = \omega_{sq}^{i-1} + \frac{\omega_{sq}^{i-1} \cdot t_i}{\sum_{leafs} \omega_{leaf}^{i-1} t_i}$$

где t_i – Вес очередного ребра или узла. Для примера, показанного на рис.8.10 процесс будет выглядеть так:

Лист	0	a	b
A	2	$2 + \frac{2 \cdot 3}{2+2} = 3.5$	$3.5 + \frac{3.5 \cdot 3}{3.5+3.5+5} = 4.375$
B	2	$2 + \frac{2 \cdot 3}{2+2} = 3.5$	$3.5 + \frac{3.5 \cdot 3}{3.5+3.5+5} = 4.375$
C	5		$5 + \frac{5 \cdot 3}{3.5+3.5+5} = 6.25$
D	8		8

Итого веса разделились в пропорции 35:35:50:64. Метод Гернштейна-Соннхаммера-Чотьи используется весьма широко благодаря простой реализуемости и скорости.

8.4 Поиск мотивов

Задача поиска мотивов является достаточно популярной в современной биоинформатике. Существуют даже соревнования в поиск мотивов. Суть задачи в следующем. Дано несколько последовательностей, в которых есть, например, сайты связывания транскрипционного фактора. Нам надо из анализа этих последовательностей понять, как устроен сам сайт связывания. Источники данных для этой задачи бывают разные (рис.8.11).

Иммунопреципитация хроматина (ChIP-seq). (рис.8.11A) В этом случае у нас есть выделенный белок. К этому белку можно сделать антитела. Мы фиксируем клетки формальдегидом, который пришивает ковалентно все белки к хроматину. Затем мы фрагментируем ДНК и с помощью антител к этому белку отбираем фрагменты ДНК, которые с ним связались. Затем мы секвенируем эти фрагменты. В результате получаем набор последовательностей, которые содержат сайт связывания.

SELEX. (рис.8.11B). SELEX расшифровывается как Systematic evolution of ligands by exponential enrichment. Для этого метода нам тоже нужен очищенный белок (транскрипционный фактор). Мы иммобилизуем его на колонке. Далее синтезируем набор случайных последовательностей ДНК, фланкированных известной последовательностью. Смесь этих фрагментов прогоняем через колонку в мягких условиях. Те фрагменты, которые имели хоть какую-то аффинность к белку – зацепились за него. Далее, с помощью фиксированных фланков методом полимеразной цепной реакции амплифицируем то, что связалось на колонке с белком. При этом условия амплификации такие, что допускают ошибки (мутации). Полученные фрагменты

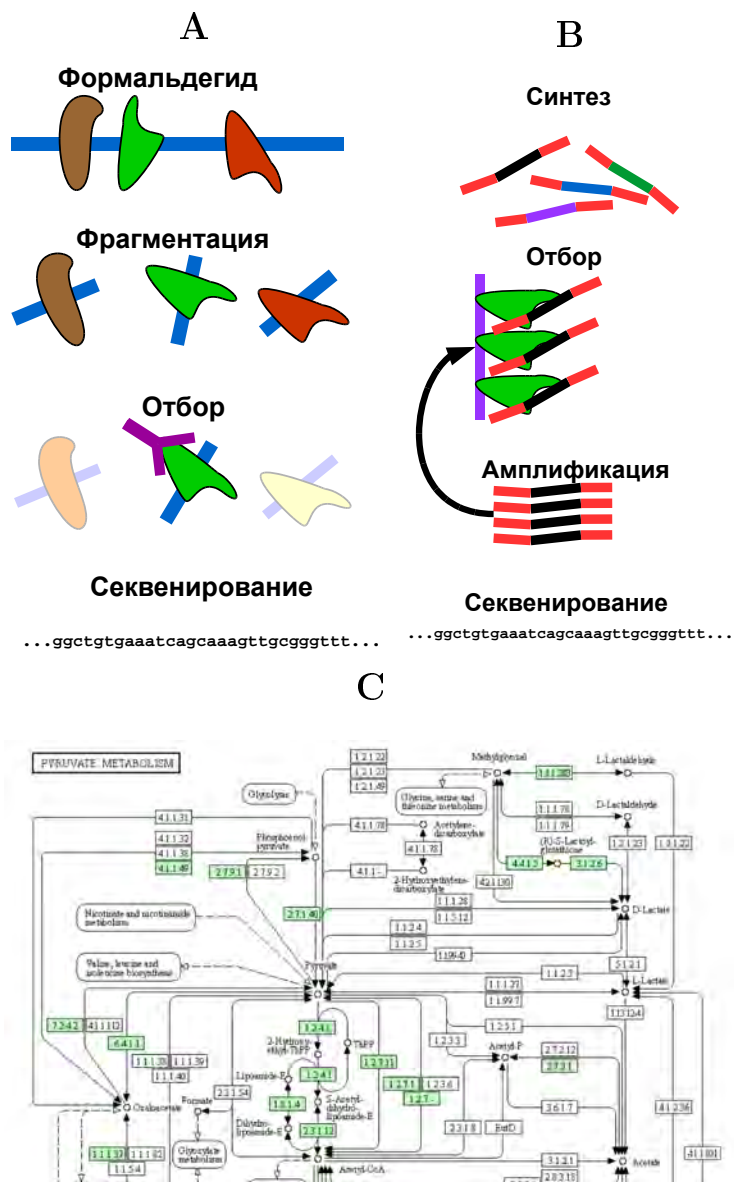


Рис. 8.11: Экспериментальные методы определения связывания транскрипционных факторов. А. ChIP-seq; В. SELEX. красные – это фиксированные фланки для амплификации; С. Метаболические пути.

снова прогоняем через колонку и опять амплифицируем. Эту процедуру повторяем много раз. В результате получаем набор фрагментов, которые

хорошо связываются с интересующим нас белком.

Анализ метаболических путей. (рис.8.11С) Этот метод чаще всего используется при анализе бактериальных геномов. Если есть некоторый метаболический путь, то можно предположить, что гены, кодирующие ферменты этого пути совместно регулируются. Обычно регуляторные участки расположены перед регулируемыми генами. Поэтому в качестве набора последовательностей, в которых мы ожидаем увидеть сайты связывания выберем области перед аннотированными генами, связанными с интересующим нас метаболическим путем.

Постановка задачи

Дано: Есть набор последовательностей. Число последовательностей варьирует от небольших десятков до десятков тысяч. Длина последовательностей – от десятков до тысяч символов.

Задача: Найти профиль, описывающий сайт связывания транскрипционного фактора. Здесь обычно предполагается, что длина сайта связывания заранее задана. При этом иногда не все последовательности обязаны иметь в своем составе сайт связывания и число сайтов связывания на одной последовательности может быть и больше 1. Эта задача, а точнее данные, имеют ряд особенностей и ограничений. Во-первых, если сайт короткий, а последовательности – длинные, то задача не разрешима по чисто статистическим причинам. Например, у нас есть некоторое количество последовательностей длиной 1000 нуклеотидов и ищем сайт размером 4 нуклеотида. Ясно, что любая четверка присутствует почти во всех последовательностях, поэтому задача оказывается вырожденной.

Если у нас ожидаемая длина сайта больше, но сайты могут быть вырожденными, то опять мы сможем практически в любой последовательности найти любые сайты связывания по случайным причинам. Если длина последовательностей порядка 1000, то необходимо, чтобы суммарное информационное содержание сайта существенно превышало 10.

8.4.1 Алгоритм MEME

Одним из наиболее популярных методов для предсказания сайтов связывания является алгоритм MEME (Multiple EM For Motif Elicitation). Он основан на процедуре максимизации ожидания (ЕМ), также как рассмотренный ранее (см. главу про скрытые марковские модели) алгоритм Витерби обучения или алгоритме Баума Вельча. Общая схема алгоритмов максимизации ожидания заключается в следующем. Берется некоторое начальное приближение для параметров (в нашем случае для позиционной весовой матрицы). Далее начинается цикл, на каждом шаге которого выполняется два этапа

Е (Expectation) – построение разметки сайтов при данных параметрах

М (Maximization) – по существующей разметке определение новых параметров (максимизация правдоподобия)

Цикл завершается, если параметры перестали меняться или достигнуто заданное количество оборотов цикла. Сходимость метода довольно быстрая – достаточно несколько десятков итераций. Эта процедура сходится к некоторому локальному экстремуму, поэтому в этой процедуре существенное значение имеет стартовое приближение.

Стартовое приближение. В алгоритме MEME для предсказания мотивов есть несколько эвристик. Мы предполагаем, что сайты связывания в выборке есть. Тогда, если некоторое слово W есть реальный сайт, то это слово должно иметь достаточно высокий вес относительно *правильной* весовой матрицы PWM. Поскольку алгоритм быстро сходится, то можно в качестве исходного приближения просто перебрать все слова заданной длины, которые встретились в нашем наборе данных. В качестве стартового приближения мы возьмем матрицу, построенную по слову W :

$$f_i(\alpha) = \frac{\delta(W_i, \alpha) + \psi}{|A| \cdot (1 + \psi)};$$

$$PWM_i(\alpha) = \log \frac{f_i(\alpha)}{q_\alpha} = \log \left(\frac{1}{q_\alpha} \cdot \frac{\delta(W_i, \alpha) + \psi}{|A| \cdot (1 + \psi)} \right)$$

где ψ – псевдосчетчик; q_α – фоновая частота символа α ; A – размер алфавита; $\delta(a, b)$ – символ Кронекера:

$$\delta(a, b) = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$$

Перебрав в качестве стартового приближения все слова, которые встретились в наборе данных, отберем профили с наибольшим информационным содержанием.

Одним из вариантов перебора слов в наборе последовательностей можно использовать эвристики, основанные на частотах встречаемости слов.

8.4.2 Gibbs Sampler

Для поиска мотивов можно использовать также методы стохастической оптимизации. Основная идея метода заимствована из физики. В качестве метафоры рассматривается модель системы, взаимодействующей с термостатом. После определенного времени (времени релаксации) распределения частот встречаемости состояний будет соответствовать распределению Гиббса. В предложенном методе также будет проводиться релаксация.

Алгоритм заключается в следующем (рис.8.12). Зададимся длиной слова W . Состоянием системы будем называть вектор позиций сайтов связывания pos_i . Если вектор позиций задан, то мы можем построить вероятностную модель, т.е. определить вероятности появления букв в позициях сайта $P_i(\alpha)$.

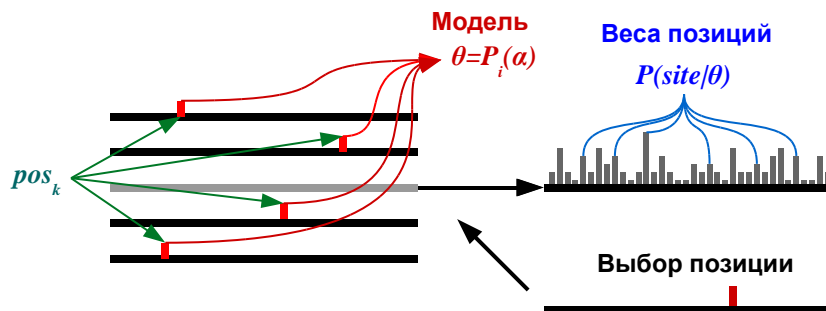


Рис. 8.12: Алгоритм Gibbs Sampler

Алгоритм начинается с того, что случайно разбрасываем позиции сайтов pos_i . Далее в цикле повторяем процедуру:

- выбираем случайную последовательность k из набора. По оставшимся последовательностям строим модель $\theta = P_i(\alpha)$
- По модели определяем веса позиций $P_s = P(site|s|\theta)$
- С вероятностями, пропорциональными P_s разыгрываем новую позицию кандидата сайта pos_k в этой последовательности и возвращаем последовательность в общий набор.

В процессе работы алгоритма набираем статистику — частоту посещения каждой позиции f_s^k . По набору наиболее посещаемых позиций предсказываем сайт. Этот алгоритм с большой вероятностью сходится к глобальному оптимуму, но обладает недостатками: 1) необходимо сделать много итераций; 2) Предполагает наличие одного сайта на последовательность.

8.4.3 Расширения, дополнения и обобщения

Методы, описанные выше, основаны на модели мотива, в котором все позиции независимы. Однако это далеко не всегда оправдано. В частности, связывание белков с ДНК зависит от локальных физических и геометрических свойств ДНК, таких, как гибкость, шаг спирали и пр. Эти свойства зависят не только от отдельных нуклеотидов, но от их пар. Поэтому модели должны учитывать марковские свойства сайтов связывания. В настоящее время используют более сложные модели, в частности, учитывающие марковские свойства мотивов. Однако, усложнение модели приводит к увеличению числа параметров, а ограниченная обучающая выборка часто не позволяет получить хорошие оценки параметров.

Часто сайты связывания обладают некоторыми особенностями. Например, большинство сайтов связывания являются палиндромами — самокомплементарными последовательностями, как, например `aagacgtctt`, или пря-

мыми повторами типа **gacttgacttgactt**. Это связано с тем, что фактор транскрипции связывается с ДНК в виде олигомера.

Вероятностные свойства фона также сложнее, чем предполагает модель независимого порождения. В частности, в геномах избегаются некоторые комбинации нуклеотидов. Кроме того, разные слова по чисто вероятностным причинам имеют разные распределения частот встречаемости. При поиске мотивов в геномах часто привлекают дополнительные данные, такие, как доступность хроматина.

Можно принимать во внимание, что в эукариотах сайты связывания транскрипционных факторов образуют регуляторные модули, которые содержат несколько сайтов связывания одного или нескольких разных транскрипционных факторов. В последнее время делаются попытки применить к этой задаче методы машинного обучения, такие как метод опорных векторов (SVM), варианты случайного леса, а также глубокие нейронные сети.

Небольшое биологическое отступление. Мотивы связывания транскрипционных факторов – вырождены. Имея модель, например позиционную весовую матрицу, можно построить идеальный сайт связывания, просто выбрав буквы с наибольшим весом. Оказывается, что идеальные сайты связывания практически не встречаются. И для этого есть серьезные биологические причины. Дело в том, что транскрипционные факторы не должны очень прочно связываться – им надо время от времени покидать ДНК, иначе регуляция не будет адекватно работать. Известен пример, когда искусственно создали оптимальный промотор *E.coli*. Оказалось, что он не может эффективно работать, поскольку сигма-субъединица прочно связывается с ДНК и не дает РНК полимеразе начать элонгацию. Описанные методы применимы (и применяются) также для поиска специфических сигналов в белках.

Глава 9

Множественное выравнивание

9.1 Постановка задачи

До сих пор мы предполагали, что множественное выравнивание дано. Теперь пора разобраться с тем, откуда берутся множественные выравнивания. Биологически множественным выравниванием называется такой способ записи последовательностей друг под другом так, чтобы *гомологичные* остатки были расположены друг под другом, т.е. принадлежали одной колонке. Здесь надо обратить внимание на то, что в биологии гомологичными называют остатки, которые имеют общего предка. Но для того, чтобы установить факт гомологичности необходимо знать всю эволюционную историю, что, в большинстве случаев недоступно. На практике используется суррогат понятия гомологичных остатков. Гомологичными остатками считают остатки, которые занимают аналогичные позиции в пространственной структуре белков.

Золотой стандарт. Используя суррогатное определение, в качестве золотого стандарта для алгоритмов построения множественного выравнивания используют множественные выравнивания пространственных структур. Для этого разработан ряд баз данных, в которых собраны множественные выравнивания, построенные с помощью сравнения пространственных структур.

Задача: Дан набор последовательностей. Построить алгоритм, который для данного набора построит множественное выравнивание, которое будет соответствовать золотому стандарту. Если этот алгоритм будет хорошо работать на выборках из золотого стандарта, то можно *предположить*, что этот алгоритм будет правильно строить множественные выравнивания и в

тех случаях, когда для последовательностей нет пространственных структур.

Вообще-то это не совсем очевидно. Например, для трансмембранных белков есть очень мало разрешенных пространственных структур. Поэтому мы не имеем золотого стандарта для таких белков. Обычно в этот набор входит от десятка до нескольких тысяч последовательностей. Множественное выравнивание позволяет делать очень многие вещи. Например, можно строить профили, находить консервативные и не консервативные позиции. Еще интересным приложением множественных выравниваний является предсказание пространственных структур белков. Если для одного из белков в выравнивании известна структура, то эту структуру можно экстраполировать на остальные белки в выравнивании.

9.1.1 Качество выравнивания

По данному набору последовательностей можно построить очень большое количество разнообразных множественных выравниваний. Хотелось бы для каждого возможного выравнивания определить число, которое характеризовало бы качество выравнивания без обращения к пространственным структурам. Тогда можно было бы среди всех возможных выравниваний выбирать наилучшие в смысле этой меры.

Есть несколько способов сконструировать такую меру. Здесь мы будем считать, что эта мера аддитивна, т.е. качество выравнивания есть сумма качества по всем колонкам выравнивания минус некоторый штраф за вставки/делеции:

$$S(\text{alignment}) = \sum_{\text{columns}} S(\text{column}) - \Delta$$

где $S(\text{column})$ – вес колонки; Δ – штраф за делеции. Аддитивность веса выравнивания предполагает, что колонки выравнивания независимы, что, вообще говоря, неверно.

Энтропийная оценка.

Пусть в каждой колонке есть характерные вероятности встречи каждой буквы $P_i(\alpha)$. Тогда, предполагая, что последовательности независимы (в рамках некоторой модели генерации последовательностей), можно оценить вероятность колонки:

$$P(\text{column}) = \prod_{\alpha} p_i(\alpha)^{c_i(\alpha)}$$

где $c_i(\alpha)$ – количество встреч буквы α в колонке i . Здесь мы не используем полиномиальное распределение, поскольку важно, что та или иная буква встретила в определенной последовательности. Тогда вероятность выравнивания будет $P(\text{alignment}) = \prod_{\text{columns}} P(\text{column})$. Логарифм вероятности

$P(column)$ определяет энтропию колонки, а вес выравнивания будет равен сумме весов колонок:

$$S = -\log P(alignment) = \sum_{columns} \sum_{\alpha} c_i(\alpha) \log P_i(\alpha)$$

Для оценки вероятности $P_i(\alpha)$ применим стандартную оценку:

$$P_i(\alpha) = \frac{\tilde{c}_i(\alpha)}{\sum_{\alpha} \tilde{c}_i(\alpha)}$$

где $\tilde{c}_i = c_i(\alpha) + \psi_i(\alpha)$ – число встреч буквы α в колонке i , поправленное с учетом псевдо-счетчиков. Этот метод оценки качества выравнивания обладает рядом недостатков. Во-первых, он не учитывает сходство аминокислотных остатков. Во-вторых, он не удобен алгоритмически.

Сумма весов пар

Самым популярным способом оценки веса колонки является сумма весов пар (рис.9.1). Для оценки веса колонки мы просто перебираем все пары остатков в колонке и суммируем соответствующие элементы матрицы сопоставления остатков:

$$S_{sum}(column) = \sum_{k < l} s(x_i^k, x_i^l) \quad (9.1)$$

где i – номер колонки, k – номер последовательности, $s(\alpha, \beta)$ – матрица сопоставления символов.

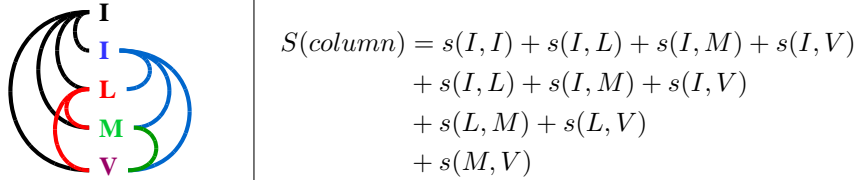


Рис. 9.1: Вычисление суммы весов пар

На самом деле такой подход не обоснован с точки зрения теории вероятностей. Вспомним, что матрица сопоставления аминокислотных остатков есть логарифм отношения правдоподобий модели выравнивания и случайной модели: $s(\alpha, \beta) = \log \Pr(\alpha, \beta | A) / q(\alpha)q(\beta)$. Если у нас в колонке i наблюдаются символы x_i^k , где k – номер последовательности, то правдоподобие будет:

$$\begin{aligned}
 S_{likelihood}(column) &= \log \frac{\Pr(x_i^1 \dots x_i^N | A)}{\prod q(x_i^k)} \\
 \neq S_{sum}(column) &= \sum_{k < l} \log \frac{\Pr(x_i^k, x_i^l | A)}{q(x_i^k)q(x_i^l)}
 \end{aligned}$$

Для правильной оценки правдоподобия нам необходимо очень много параметров, получить которые не представляется возможным. Более того, наборы параметров будут зависеть от толщины выравниваний. С другой стороны, с точки зрения здравого смысла оценка (9.1) представляется вполне разумной. Кроме того, этот способ оценки веса выравнивания очень удобен алгоритмически.

9.2 Динамическое программирование

Поскольку у нас есть функция качества множественного выравнивания, хочется ее напрямую оптимизировать методом динамического программирования. Пусть, например, у нас есть три последовательности. Если при парном выравнивании мы просматривали двумерную матрицу, в которой строки соответствовали первой последовательности, а столбцы – второй последовательности, в случае трех последовательностей нам надо просмотреть трехмерное пространство (рис.9.2), каждое измерение которого соответствует одной последовательности.

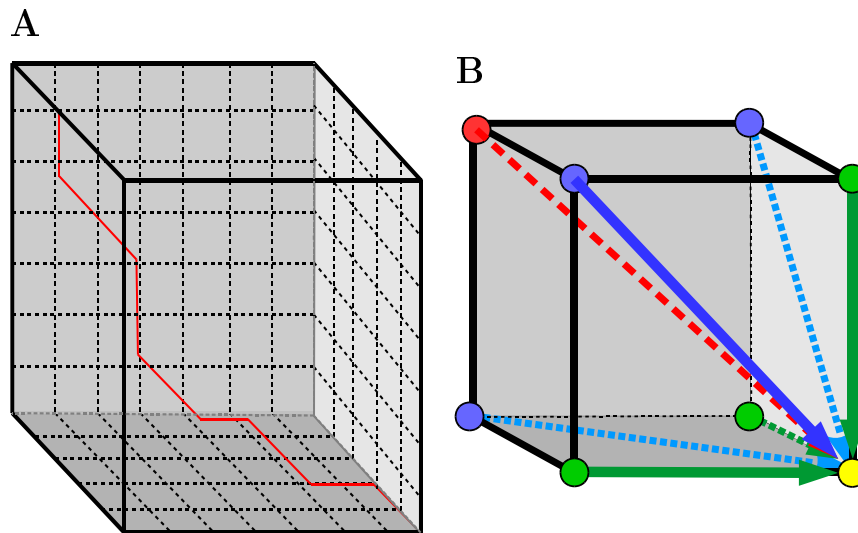


Рис. 9.2: Динамическое программирование для трех последовательностей. А – пространство просмотра; В – элементарная ячейка.

Рекурсия для этого случая будет иметь вид (здесь использованы линей-

ные штрафы):

$$w(i, j, k) = \begin{cases} w(i-1, j-1, k-1) + s(x_i^1, x_j^2) + s(x_i^1, x_k^3) + s(x_j^2, x_k^3) \\ w(i-1, j-1, k) + s(x_i^1, x_j^2) - d \\ w(i-1, j, k-1) + s(x_i^1, x_k^3) - d \\ w(i, j-1, k-1) + s(x_j^2, x_k^3) - d \\ w(i-1, j, k) - 2d \\ w(i, j-1, k) - 2d \\ w(i, j, k-1) - 2d \end{cases} \quad (9.2)$$

Время работы этого алгоритма для трех последовательностей имеет порядок $7 \times L^3$, где L — длина последовательностей. Для случая N последовательностей в каждой ячейке число входящих ребер равно $2^N - 1$ — это число вершин многомерного куба минус 1. Для построения выравнивания надо просмотреть L^N вершин. Итого время работы алгоритма

$$T(N, L) = O(2^N \cdot L^N) = O((2L)^N)$$

Если для трех-четырех последовательностей компьютерные мощности достаточны, то для множественного выравнивания $N = 10$ последовательностей длиной $L = 100$ требуется порядка $2^{10} \cdot 100^{10} \simeq 10^{23}$ операций. При скорости 10^{10} операций в секунду для построения выравнивания потребуется 10^{13} секунд. В году примерно $3 \cdot 10^{10}$ секунд. Итого получается около 300 лет! На практике часто приходится строить выравнивания сотен последовательностей длиной порядка 1000 символов. Таким образом, динамическое программирование практически не применимо для построения множественного выравнивания в реальных условиях.

9.3 Прогрессивное выравнивание

Таким образом, подход, связанный с оптимизацией веса выравнивания оказался не перспективным. Вместо оптимизации можно обратиться к эволюционному подходу. Основываясь на весах парных выравниваний, можно построить кластерное дерево. Это дерево называется *путеводным деревом*, *guide tree*. Можно надеяться, что это дерево как-то отражает эволюционные события (рис.9.3). Будем строить выравнивание последовательно от листьев к корню. Такой подход называется *прогрессивным выравниванием*.

9.3.1 Основной алгоритм

В узлах дерева обычно имеются в виду предковые последовательности. Однако, мы не знаем предковых последовательностей и, более того, не можем их надежно восстановить. Поэтому будем считать, что там расположены стопки выравненных последовательностей. Про первый уровень (узлы a,b,c)

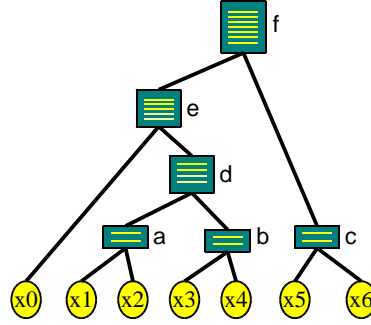


Рис. 9.3: Дерево, построенное по последовательностям

все понятно – там расположены парные выравнивания. Далее при построении очередной стопки выровненных последовательностей, например в узле d будем строить выравнивание стопок последовательностей (рис.9.4).

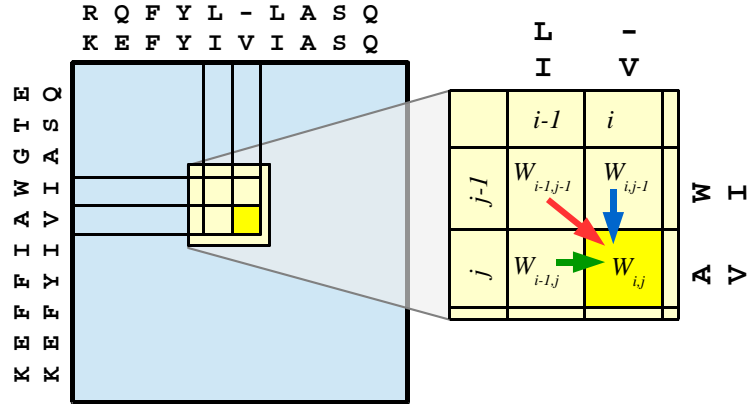


Рис. 9.4: Динамическое программирование для стопок

Для динамического программирования можно написать рекурсию:

$$w(i, j) = \begin{cases} w(i-1, j-1) + S(i, j) \\ w(i-1, j) - \Delta \\ w(i, j-1) - \Delta \end{cases} \quad (9.3)$$

При выравнивании стопок обычно используют линейный штраф за делецию Δ . Теперь определимся с вкладом за сопоставление $S(i, j)$. Используем в качестве веса сопоставления вес суммы пар (уравнение 9.1). Обозначим одну стопку набором последовательностей $\{x^k\}$, а другую $\{y^k\}$. Сумма пар для соединенных стопок распадется на три составляющие (рис.9.5).

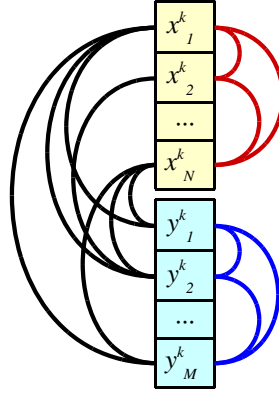


Рис. 9.5: К вычислению суммы пар для выравнивания

$$S(i, j) = \sum_{l < k} s(x_i^l, x_i^k) + \sum_{l < k} s(y_j^l, y_j^k) + \sum_{l, k} s(x_i^l, y_j^k)$$

Первая и вторая суммы относятся только к стопкам $\{x^k\}$, $\{y^k\}$, в то время, как последняя сумма отвечает за перекрестное сравнение. В отличие от настоящих последовательностей, в стопках встречаются символы ‘-’ и в соответствующих суммах может появиться сравнение буквы с пробелом и пробела с пробелом. Поэтому в таблицу весов сопоставления символов вводится еще один дополнительный символ ‘-’: $s(-, \alpha)$. Вес сопоставления пробелов обычно полагают равным 0: $s(-, -) = 0$. Этот метод был впервые предложен Фенгом и Дулитлом в 1987 г. Этот метод с рядом улучшений и эвристик лежит в основе весьма популярной программы ClustAlW. Программа отличается достаточно высокой скоростью и производит достаточно качественные выравнивания при уровне сходства до 40%.

9.3.2 Улучшение выравнивания

Важно отметить, что описанный алгоритм построения прогрессивного выравнивания является эвристическим и **не строит оптимальное** выравнивание. Выравнивание может зависеть от порядка сборки, т.е. от дерева. Основная проблема, которая возникает при прогрессивном выравнивании, заключается в следующем. Если на каком-то этапе мы ошибочно сделали вставку в последовательность, то эта вставка, во-первых, дойдет до самого корня и будет в финальном выравнивании. Хуже то, что ошибка в выравнивании на каком-то этапе при продвижении вверх по дереву будет порождать ошибки на следующих этапах.

Чтобы как-то справиться с ситуацией применяют процедуру уточнения выравнивания. Один из способов улучшения выравнивания заключается в следующем (рис.9.6). В произвольном месте разрезаем путеводное дерево.

Отдельно выравниваем в соответствии с поддеревьями, а потом строим выравнивание полученных стопок. Если новое выравнивание имеет лучшее качество (лучший вес), то оно принимается. Эта процедура повторяется много раз. Метод, основанный на разрезании путеводных деревьев называется

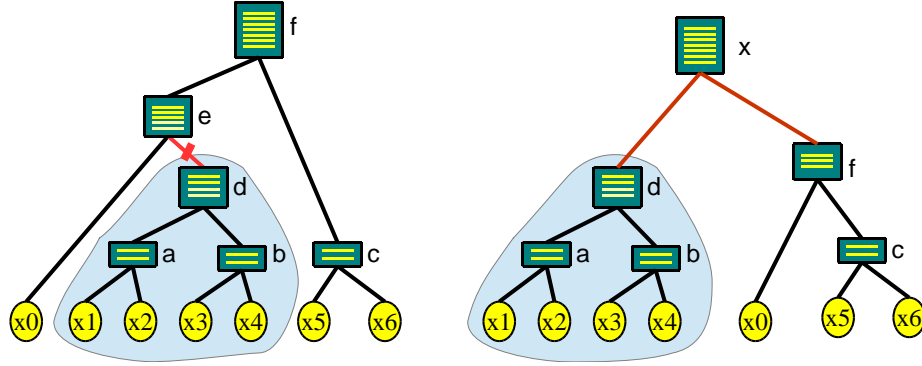


Рис. 9.6: Уточнение выравнивания

ется улучшением (refinement) выравнивания.

9.3.3 t-Coffee

Описанный выше алгоритм является основой для многих методов. В основе семейства алгоритмов t-Coffee лежит построение библиотек консервативных фрагментов, полученных из анализа парных и тройных выравниваний и коррекция схемы весов.

9.3.4 Muscle

- При выравнивании профилей используется весовая функция:

$$S(x_i y_j) = (1 - G_i^x)(1 - G_j^y) \log \sum_{\alpha} \sum_{\beta} f_{\alpha}^x(i) f_{\beta}^y(j) p_{\alpha\beta} / p_{\alpha} p_{\beta}$$

где G_i^x – частота делеции в позиции i в профиле x ;

$f_{\alpha}^x(i)$ – частота встречаемости буквы α в колонке;

$p_{\alpha\beta}$, p_{α} , p_{β} – априорные частоты встречаемости пар букв и букв в выравниваниях

- После построения прогрессивного выравнивания перестраивается дерево, и по новому дереву снова строится выравнивание.
- Используется уточнение выравнивания.

9.3.5 ProbCons

Алгоритм ProbCons является одним из самых точных. Идея алгоритма заключается в следующем. Мы применим алгоритм Вперед-Назад и определим вероятности сопоставления символов для всех пар последовательностей:

$$P(x_i^k \circ x_j^l) = \frac{f_{k,l}^M(i, j) \cdot b_{k,l}^M(i, j)}{P(x^k, x^l)}$$

где k, l – номера последовательностей в выборке. Время такой обработки равно $T \sim O(L^2 N^2)$, где L, N – средняя длина последовательности и количество последовательностей. Теперь строим множественное выравнивание, но при этом вместо матрицы замен аминокислотных остатков используем позиционно-зависимые веса сопоставлений. В качестве веса сопоставления будем использовать не $P(x_i^k \circ x_j^l)$, как можно предположить, а то, насколько хорошо два символа $x_i^k \circ x_j^l$ сопоставляются с точки зрения остальных последовательностей:

$$S(x_i^k \circ x_j^l) = \sum_{z \neq k, l} \sum_s P(x_i^k \circ x_s^z) \cdot P(x_s^z \circ x_j^l)$$

здесь мы суммируем по всем последовательностям x^z и по всем позициям s в последовательности x^z . Время выполнения этой операции (фактически умножения матриц) составляет L^3 и еще производится суммирование по всем последовательностям – N , причем это надо сделать для всех пар последовательностей N^2 итого $T \sim L^3 N^3$. Т.е. этот алгоритм достаточно ресурсоемкий. К тому же в памяти надо хранить все позиционные матрицы сопоставления $P(x_i^k \circ x_j^l)$. При построении множественного выравнивания используется стратегия прогрессивного выравнивания с путеводным деревом, штрафы за вставки-делеции ставятся равными 0: $\Delta = 0$. Но при вычислении вероятностей сопоставления использовались обычные штрафы за делеции и исходные весовые матрицы. Как видно, этот метод, несмотря на высокое качество, является весьма затратным – надо помнить все матрицы вероятностей сопоставления, а время вычислений достаточно большое.

9.3.6 MAFT

Этот метод использует преобразование Фурье для поиска сегментов с высоким уровнем сходства. Преимущество этого подхода заключается в том, что при небольшом сдвиге одного сегмента относительно другого Фурье спектр не сильно меняется. Это позволяет быстро выделить похожие сегменты. Далее используется стандартная процедура прогрессивного выравнивания с использованием выделенных пар сегментов в качестве опорных. Алгоритм достаточно быстрый при неплохом качестве. Особенно широко используется для выравнивания нуклеотидных последовательностей.

9.3.7 Dialign

От прогрессивного выравнивания отличается метод Dialign. Он основывается на поиске консервативных сегментов. Для поиска таких сегментов можно использовать алгоритмы поиска сигналов, рассмотренные ранее. Допустим, мы нашли наилучший сигнал, который встретился во всех последовательностях (рис.9.7, красные сегменты). Мы их располагаем друг под другом, затем ищем более слабые консервативные участки слева и справа от полученного выравнивания сегментов. Их тоже закориваем. Теперь каждая последовательность разбита на 4 сегмента. В каждом из них снова ищем консервативные сегменты, и т.д. Когда нам не удастся найти такие сегменты, можно для фрагментов применить обычное прогрессивное выравнивание.



Рис. 9.7: Алгоритм Dialign

Глава 10

Вторичная структура РНК

10.1 Функции РНК

Рибонуклеиновые кислоты принимают участие во многих биологических процессах. Все функции РНК можно условно разделить на несколько основных классов. При этом некоторые РНК могут относиться к нескольким классам.

Информационная – РНК этого класса являются хранилищем и переносчиком информации. К этому классу можно отнести матричные РНК и геномные РНК вирусов.

Структурная – РНК участвуют в формировании различных структур. Это, прежде всего, рибосомная РНК. Сюда же можно отнести РНК, участвующие в формировании клеточных и ядерных структур. Например РНК MALAT1 участвует в формировании спеклов – ядерных структур.

Каталитическая – Это РНК, которые являются катализаторами различных реакций. В частности, реакцию образования пептидной связи при синтезе белка катализирует рибосомная РНК. Интроны I и II типа вырезают сами себя. РНК в сплайосомах также катализирует реакции вырезания интронов.

Регуляторная – Многие РНК участвуют в регуляции различных процессов. В частности, это микро РНК, участвующие в регуляции трансляции эукариотических мРНК, рибопереключател и Т-боксы – структуры бактериальных мРНК, которые регулируют экспрессию генов.

Направляющая – РНК часто играет направляющую роль, когда та, или иная РНК определяет мишень для действий белковых ферментов. Примером таких РНК являются многие малые ядрышковые РНК (snRNA), которые направляют белки к сайтам модификации рибосомных

РНК, РНК системы CRISPR-CAS, направляющие специфические белки к определенным сайтам на ДНК. Еще один пример РНК такого типа – это РНК, направляющие систему редактирования РНК к определенным сайтам.

Транспортная – Это транспортные РНК, которые являются медиаторами для переноса аминокислотных остатков к рибосоме.

Другие – Для синтеза ДНК необходимы РНК затравки – праймеры. Теломеразная РНК служит матрицей для синтеза теломерных повторов. Структура РНК в окрестности сайта инициации трансляции мРНК в бактериях может понижать уровень трансляции. Структура РНК в конце бактериальных генов является сигналом терминации транскрипции.

Неизвестная – Есть много типов РНК с не до конца изученной функцией.

- piRNA
- Vault RNA
- lncRNA

Есть гипотеза «РНК мира», суть которой заключается в том, что жизнь возникла на основе РНК, которая одновременно выполняла как информационную, так и ферментативную роль. Был искусственно создан РНК-фермент, катализирующий собственную репликацию (Johnston W., Unrau P., Lawrence M., Glasner M., Bartel D. RNA-catalyzed RNA polymerization: accurate and general RNA-templated primer extension. // Science (2001); Vol. 292, no. 5520.)

10.2 Структура РНК

РНК, как правило, является однонитевым полинуклеотидом, состоящим в основном из нуклеотидов четырех типов – аденина (А), цитозина (С), гуанина (G) и урацила (U). На самом деле реальное многообразие нуклеотидов, входящих в РНК, больше, поскольку во многих природных РНК есть еще некоторое количество модифицированных оснований, поскольку происходят пост-транскрипционные модификации, в том числе конверсия урацила в тимин, метилирование оснований, замена отдельных нуклеотидов в результате редактирования, например, замена аденина на инозин (A→I) и пр. Однако, количество модифицированных оснований невелико. Возможность образовывать уотсон-криковские пары позволяет этой молекуле образовывать большое количество разнообразных структур.

Полинуклеотиды являются гораздо более простыми молекулами, чем белки, поскольку они состоят всего из четырех оснований, в отличие от 20 в белках, при этом свойства мономеров гораздо более однородны, чем аминокислотные остатки. У РНК, в отличие от ДНК, в качестве остова используется рибоза, а не дезоксирибоза. Это, в частности, приводит к тому,

что двунитевая спираль находится в А-форме, которая, в отличие от ДНК, находящейся в В-форме, имеет 11 оснований на виток. Также как и для белков различают уровни структурной организации РНК (рис.10.1):

Первичная структура – Это просто последовательности нуклеотидов.
(Никогда не говорите «*первичная последовательность*»!, поскольку эта фраза предполагает наличие *вторичной последовательности*, что является нонсенсом)

Вторичная структура – Это совокупность уотсон-криковских взаимодействий

Третичная структура – Это пространственная структура молекулы

Четвертичная структура – Это пространственная структура молекулярных комплексов, таких, как рибосомы и, как правило, включает в себя кроме РНК белки.

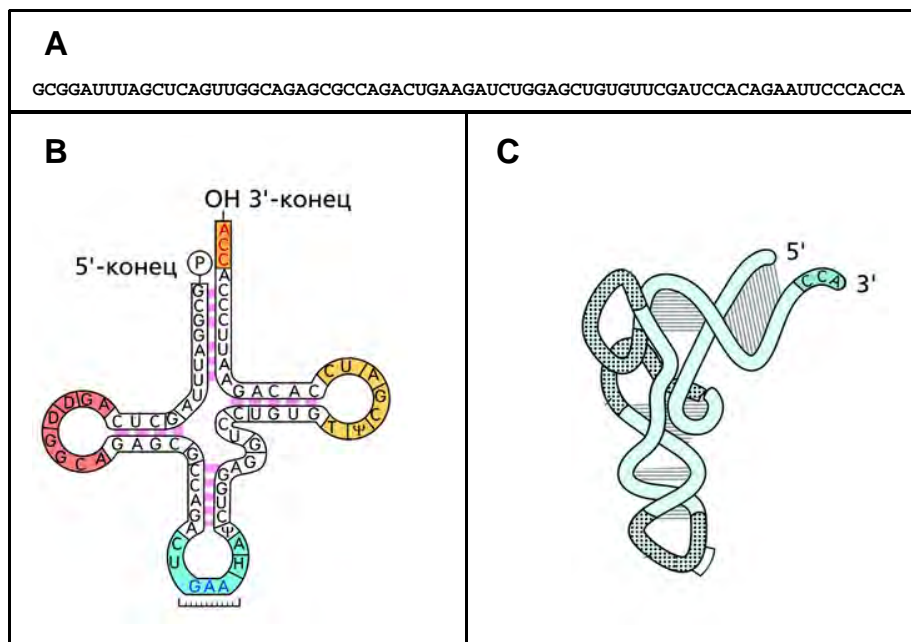


Рис. 10.1: Первичная (А), вторичная (В) и третичная (С) структуры тРНК

Часто предполагается, что структуры более высокого уровня определяются структурами более низкого уровня. Например, вторичная структура определяется первичной, а третичная – вторичной. Этот подход отражает представление о том, что энергия каждого следующего уровня намного ниже, чем энергия предыдущего уровня. Строго говоря, такой подход не совсем

верен, но он позволяет достаточно эффективно решать задачу предсказания вторичной структуры и получать интересные и важные результаты.

В 1965 году в лаборатории Р.Холли была впервые определена последовательность тРНК^{Ala} (Нобелевская премия 1968 г.). Для определения первичной структуры тРНК были применены разнообразные биохимические методы. Во-первых, была разработана схема выделения тРНК одного типа. Далее с помощью панкреатической рибонуклеазы (расщепляет у 3'концов пиримидинов) и с помощью рибонуклеазы T1 такадиастазы (выделен из мицелия грибов рода *Aspergillus*, расщепляет у 3' конца гуанинов) он независимо расщепил ранее выделенную аланиновую тРНК.

Было получено 19 и 17 фрагментов соответственно. Эти фрагменты были разделены хромофотографией. Далее, с помощью ступенчатой деградации фосфодиэстеразой змеиного яда, была получена последовательность каждого фрагмента. Далее фрагменты были объединены в единую последовательность. В целом, процедура похожа на процедуру определения аминокислотной последовательности инсулина Сэнгером (его первая Нобелевская премия). Обращу внимание, что в то время еще не было электрофореза в геле.

Получив последовательность, Р.Холли предположил, что ее можно уложить в структуру клеверного листа. Определение последовательностей других тРНК позволило убедиться, что структура клеверного листа является универсальной для этого типа молекул. В 1978 году с помощью рентгеноструктурного анализа была получена окончательная структура тРНК двумя независимыми группами, которая полностью подтвердила структуру, предположенную Р.Холли.

В настоящее время разработан ряд экспериментальных подходов к массовому определению вторичной структуры с использованием технологий второго поколения секвенирования. В основе этих подходов лежит обработка разными реагентами, которые способствует модификациям нуклеотидов, которые не находятся в спаренном основании.

10.2.1 Элементы вторичных структур РНК

Обычно во вторичной структуре выделяют петли и спирали. Спираль — это непрерывная последовательность нуклеотидных пар. Непрерывная последовательность неспаренных оснований, ограниченная с двух сторон спаренными основаниями. 5' и 3' хвосты молекул не являются структурными элементами. Более строгое определение:

Определение. При заданной структуре S нуклеотид k считается *доступным* для пары $(i < j)$, если НЕ существует пары $(r < s)$ такой, что $i < r < k$ & $k < s < j$. Множество доступных нуклеотидов для любой пары называется *петлей*. (i, j) — *закрывающая* пара. Среди петель выделяются 4 основных типа:

Шпилька (hairpin) замыкает собой спираль, т.е. ее концы принадлежат одной спирали.

Выпячивание (bulge) петля, соединяющая две спирали так, что по одной нити спирали спаренные основания идут непрерывно, а по другой — происходит выпячивание.

Внутренняя петля (internal loop) петля, соединяющая две спирали

Мульти-петля (Muti-loop) петля, соединяющая более, чем две спирали.

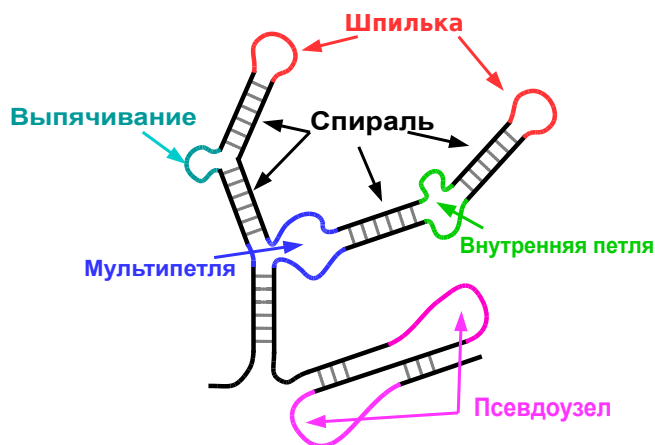


Рис. 10.2: Элементы вторичной структуры РНК

Псевдоузлы. Кроме петель и спиралей есть еще структурные элементы, называемые псевдоузлами. Пусть у нас есть две пары спаренных оснований с координатами $[i_1, j_1]$, $[i_2, j_2]$. Эти две пары оснований образуют псевдоузел, если $i_1 < i_2 < j_1 < j_2$ (рис.10.3А). Основные типы псевдоузлов показаны на рис.10.3В.

10.2.2 Представление структур РНК

Ниболее популярный способ представления вторичных структур — это рисунок, представляющий топологическую схему, см., например, рис.10.4А. Другой способ — это круговая диаграмма (рис.10.4В), которая часто бывает удобной при рассмотрении алгоритмов. Но это просто рисунки и они не применимы при рассмотрении алгоритмов. В компьютере можно представить структуру в виде массива спиралей (рис.10.4С) или массива спаренных оснований (рис.10.4Д).

Важным случаем являются структуры без псевдоузлов. Тогда структура может быть представлена в *виде правильной скобочной структуры*

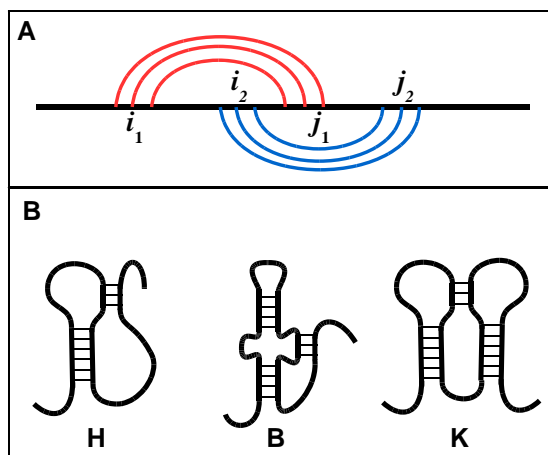


Рис. 10.3: Псевдоузлы. А – взаимное расположение спаренных оснований. В – основные типы псевдоузлов.

(рис.10.4Е) или в виде дерева (рис.10.4F). При графовом представлении структуры в качестве ребер рассматриваются спирали, а в качестве вершин – петли. Количество возможных вторичных структур без псевдоузлов для случайных бернуллиевских последовательности длиной L оценивается величиной 1.8^L

10.3 Постановка задачи

Задача предсказания вторичной структуры РНК по последовательности:

Дано Последовательность РНК

Задача Предсказать вторичную структуру РНК

Золотой стандарт. Итак, мы хотим разработать алгоритм, который, получив на входе последовательность, предскажет сеть уотсон-криковских пар. Для того, чтобы проверить, насколько правильно работает алгоритм, необходимо иметь достаточно большое количество примеров правильной вторичной структуры РНК. В качестве *золотого стандарта* можно использовать вторичные структуры, полученные с помощью рентгеноструктурного анализа. Кроме того, в качестве золотого стандарта используют консервативные вторичные структуры.

Далее мы будем рассматривать только структуры *без псевдоузлов*. Это обусловлено несколькими обстоятельствами. Во-первых, в реальных структурах псевдоузлы достаточно редко встречаются, а количество оснований, вовлеченных в образование псевдоузлов, невелико. Но при этом они иногда

метим, что если структура S составлена из двух не пересекающихся структур (структур без общих оснований) $S = S_1 + S_2$, то мощность структуры будет равна сумме мощностей под-структур $W(S) = W(S_1) + W(S_2)$ (рис.10.5).

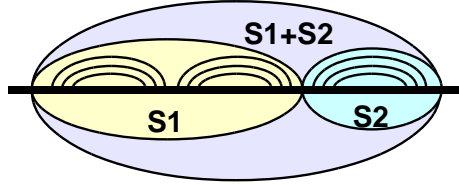


Рис. 10.5: Конкатенация структур

10.4.1 Динамическое программирование

Допустим, мы знаем оптимальные мощности всех подсегментов этого фрагмента $S(k, m); k \geq i, m \leq j$. Тогда можно найти оптимальную структуру для сегмента $[i, j+1]$. Для этого нам надо понять, спаривать ли основание $j+1$, и, если спаривать, то с кем. Тогда есть несколько возможностей построить структуру для сегмента $[i, j+1]$ (рис.10.6). Во-первых, основание $j+1$ ни с кем не спарено. Тогда структура (множество спаренных оснований) не отличается от структуры на сегменте $[i, j]$, и ее мощность будет равна $W(i, j+1) = W(i, j)$.

Второй вариант – это $j+1$ спарено с i . Тогда мощность структуры равна $W(i, j+1) = W(i+1, j) + 1$. Наконец, структура $[i, j+1]$ может быть конкатенацией двух подструктур $S[i, j+1] = S(i, k-1) + S(k, j+1)$, причем нуклеотид j спарен с нуклеотидом k .

Если все упомянутые подструктуры были оптимальными, то для того, чтобы найти оптимальную структуру, надо выбрать максимум из упомянутого. Рекурсивное вычисление максимального числа спаренных оснований основано на том, что мощность больших структур опирается на мощность меньших структур. Для инициации рекурсии заметим, что самая маленькая структура имеет длину 0 и ее мощность равна 0.

$$\begin{aligned}
 W(i, i) &= 0 \\
 W(i, j+1) &= \max \begin{cases} W(i, j) \\ W(i+1, j) + 1 \\ \max_{k \in (j+1)} W(i, k-1) + W(k, j+1) + 1 \end{cases} \quad (10.1)
 \end{aligned}$$

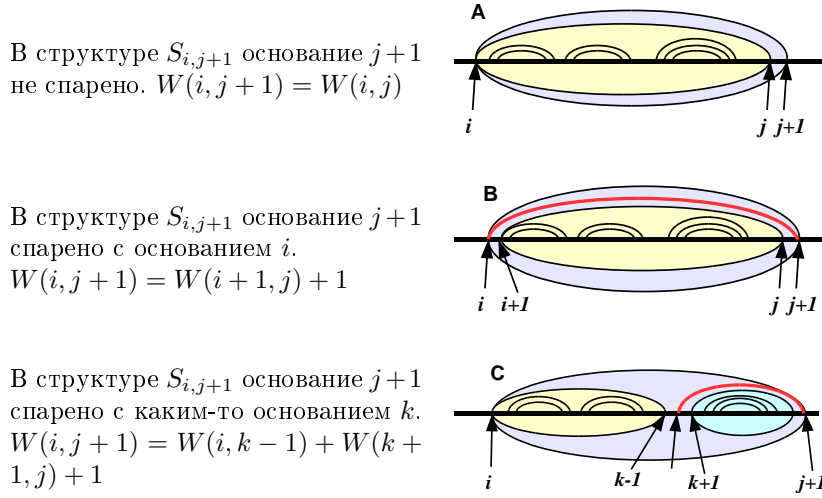


Рис. 10.6: Структура $[i, j+1]$. А – структура $[i, j]$ не отличается от структуры $[i, j]$; В – в структуре $[i, j+1]$ $j+1$ спарен с i ; С – структура $[i, j+1]$ является конкатенацией двух структур. Красным отмечено спаривание нуклеотида j

10.4.2 Восстановление структуры

Рекурсия (10.1) позволяет лишь определить максимальное число спаренных оснований. Для того, чтобы восстановить структуру, нам необходимо запомнить выбор максимума. В переменной $\pi(i, j)$ будем запоминать с каким нуклеотидом было спарено основание j для оптимального сегмента $[i, j]$. Тогда рекурсия (10.1) примет вид:

$$\begin{aligned}
 W(i, i) &= 0; & \pi(i, i) &= 0 \\
 W(i, j+1) &= \max \begin{cases} W(i, j); & \pi(i, j+1) = 0 \\ W(i+1, j) + 1; & \pi(i, j+1) = i \\ \max_k W(i, k-1) + W(k+1, j) + 1; & \pi(i, j+1) = k \end{cases}
 \end{aligned} \tag{10.2}$$

С помощью запомненной переменной $\pi(i, j)$ мы можем восстановить оптимальную структуру для любого фрагмента последовательности $[i, j]$, в том числе и для всей последовательности $[1, L]$.

Возьмем фрагмент $[i, j]$. Если $\pi(i, j) = 0$, то это значит, что в оптимальной для этого сегмента структуре j не спарено, а оптимальная структура такая же, как и для сегмента $[i, j-1]$. Тогда проверяем $\pi(i, j-1)$, и так далее, пока не встретим не нулевой элемент. Когда найдем ненулевой элемент, то он либо равен i , тогда имеем спаривание $i \circ j$, либо равен некоторому $k \neq i$. Это значит, что структура состоит из двух подструктур $[i, k-1]$ и $[k, j]$. Псевдокод восстановления структуры выглядит так:

```
1  GetStruct(i, j){
```

```

2   while(pi[i,j] == 0 & j > i) j--;
3   if(i==j) return;
4   print(i , j);
5   if(pi[i,j] == i){
6       GetStruct(i+1,j-1);
7   }
8   else{
9       k=pi[i,j];
10      GetStruct(i,k-1);
11      GetStruct(k+1,j-1);
12  }
13 }

```

На рис.10.7 показан пример обратного прохода. Выделены ячейки матрицы, через которые проходит оптимальный путь. Переход $1 : 17 \rightarrow 1 : 16$ соответствует поиску ненулевого элемента в столбце (строка 2 в коде). Переходы $1 : 16 \rightarrow 2 : 15 \rightarrow 3 : 14$; $8 : 14 \rightarrow 9 : 13 \rightarrow 10 : 12$; $3 : 7 \rightarrow 4 : 6$ соответствуют переходу по диагонали (строка 6), переход-разветвление $3 : 14 \rightarrow (8 : 14 + 3 : 7)$ соответствует расщеплению структуры на две подструктуры (строки 9–11)

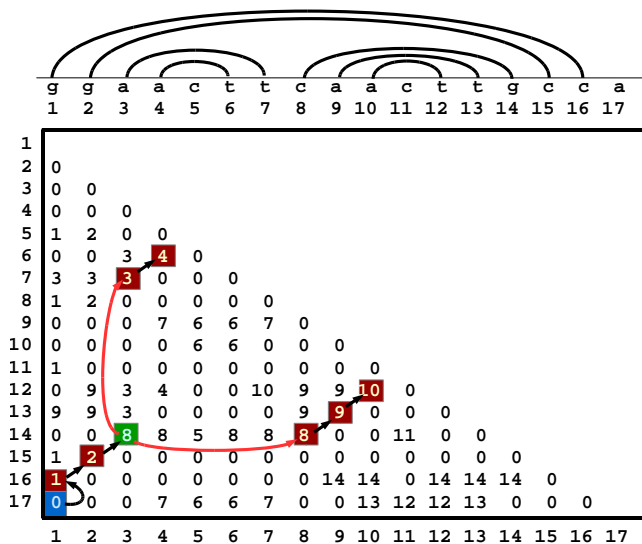


Рис. 10.7: Пример обратного прохода в алгоритме Нуссинофф

10.4.3 Условно-оптимальная структура

Пусть нам из независимых источников известно, что некоторая пара $(i \circ j)$ образует комплементарную пару. Тогда разумно поставить вопрос: найти

оптимальную структуру, при условии, что данные нуклеотиды спарены. Простейший подход заключается в том, чтобы для выделенной пары нуклеотидов в рекурсии (10.2) именно для этой пары вместо 1 поставить большую (очень большую) премию за образование пары. Далее можно применить алгоритм Нуссинофф и получить структуру с желанной парой.

Другой подход основан на алгоритме внешнего просмотра (рис.10.8). Для любой пары позиций (i, j) вторичную структуру можно разбить на две части – внешнюю и внутреннюю. К внутренней части отнесем все пары нуклеотидов r, s , для которых выполнено $i \leq r < s \leq j$, а к внешней – все остальные (рис. 10.8A). Отметим, что если $i \circ j$ – спарены, то эта пара относится к *внутренней* подструктуре. Такое разбиение хорошо представлять на круговой диаграмме (рис. 10.8B).

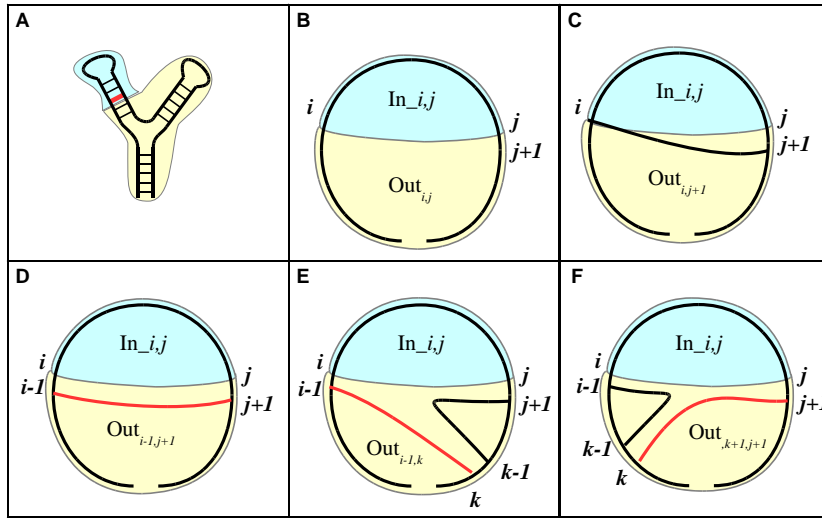


Рис. 10.8: Алгоритм внешнего просмотра. А – разбиение структуры на внутреннюю (голубая) и внешнюю (желтая); В – круговое представление разбиения на внешнюю и внутреннюю; С, D, E, F – варианты для внешней рекурсии

Условно-оптимальная структура $S|i \circ j$ состоит из внешней и внутренней подструктур $S = In \cup Out$ и, если они оптимальны (приносят максимум числа спаренных оснований), то и общая структура также оптимальна. Основной алгоритм (10.2) вычисляет оптимальную внутреннюю структуру для любой пары оснований. Для внешней структуры надо строить новый алгоритм, который, впрочем, не сильно отличается от внутреннего алгоритма (10.2).

Итак, рассмотрим пару (i, j) . Обозначим $W^O(i, j)$ мощность внешней структуры. Рассмотрим нуклеотид $j + 1$. Он может не спариться ни с кем (рис.10.8C). Тогда мощность внешней структуры $Out(i, j)$ будет равна мощности внешней структуры $Out(i, j + 1)$ $W^O(i, j + 1)$.

Другой вариант – внешняя структура $Out(i, j)$ имеет спаривание $(i -$

1) $\circ (j + 1)$ (рис.10.8D). Тогда мощность внешней структуры $Out(i, j)$ будет равна мощности внешней структуры $Out(i - 1, j + 1)$ плюс 1: $W^O(i, j) = W^O(i - 1, j + 1) + 1$. Еще нуклеотид $j + 1$ может быть спарен с каким-то нуклеотидом k . При этом может оказаться, что $k > j + 1$ (рис.10.8E) или $k < i - 1$ (рис.10.8F). В обоих случаях внешняя подструктура разбивается на внешнюю и внутреннюю подструктуры. В результате получаем рекурсию:

$$W^O(i, j) = \max \begin{cases} W^O(i, j + 1); & \pi^O(i, j) = 0 \\ W^O(i - 1, j + 1) + 1; & \pi^O(i, j) = i - 1 \\ \max_{k > j+1} W^O(i - 1, k) + W^I(j + 1, k - 1) + 1; & \pi^O(i, j) = k \\ \max_{k < i-1} W^O(k, j + 1) + W^I(i - 1, k - 1) + 1; & \pi^O(i, j) = k \end{cases} \quad (10.3)$$

Здесь, мы переменную W , которая в формулах (10.2) была переобозначена как W^I чтобы подчеркнуть, что это мощность внутренней подструктуры. Рекурсия (10.2) идет от маленьких структур к бóльшим. Рекурсия (10.3) идет от бóльших структур к мёньшим. Поскольку для всей последовательности внешняя структура – пустая, то начальное значение $W^O(1, l) = 0$. Восстановление внешней структуры происходит аналогично восстановлению внутренней структуры (`GetStruct(i, j)`) (см. выше)

```
SearchOutStruct (i, j){
    k=pi~o[i, j];
    if(k < i){
        GetStruct(k-1, i+1);
        SearchOutStruct(k, j+1);
    }
    else{
        GetStruct(j+1, k-1);
        SearchOutStruct(i-1, k);
    }
}
```

10.5 Энергия вторичной структуры

Алгоритм Нуссинофф на тестовых структурах дает достаточно слабое качество – не более 50% предсказанных структур для золотого стандарта соответствуют правильной структуре. По-видимому, при этом подходе мы что-то не то оптимизировали. С точки зрения физики представляется разумным минимизировать энергию. Какую? Поскольку для всех биологических процессов предполагается постоянство давления, то в равновесном состоянии минимизируется свободная энергия Гиббса:

$$\Delta G = \Delta H - T\Delta S \quad (10.4)$$

где ΔH – энтальпия, ΔS – энтропия системы. В биологии традиционно принято считать энергию в килокалориях на моль (*ккал/моль*). Чтобы понимать масштаб величин, значение RT при комнатной температуре равно 0.6 ккал/моль, где R – газовая постоянная. Это средняя энергия тепловых флуктуаций. Ясно, что энергия структуры определяется взаимодействием спаренных оснований. При этом энергия взаимодействия состоит из двух компонент (рис.10.9А). Во-первых, это водородные связи между основаниями на разных цепях, а во-вторых – это взаимодействие соседних оснований вдоль цепи – стекинг. Оказывается, что для РНК вклад стекинга в энергию весьма велик.

Даже свободная одонитевая РНК при комнатной температуре сохраняет стекинг (данные, полученные микрокалориметрией и с помощью кругового дихроизма). При этом энергия спирали в основном дает вклад в энтальпию. Наличие стекинга, в частности, приводит к тому, что *плавление* спиралей происходит кооперативно, т.е. весьма невероятно, что одна пара оснований в середине спирали выйдет из спирали. Нуклеотидная спираль плавится по механизму "все-или-ничего".

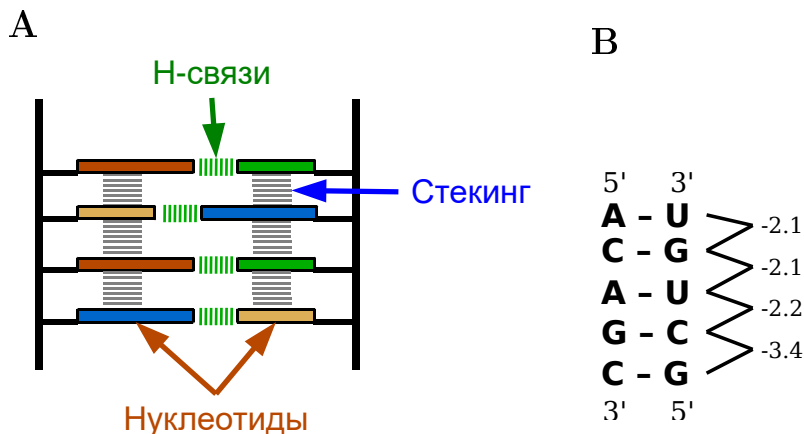


Рис. 10.9: Взаимодействие спаренных оснований. А – водородные связи и стекинг; В – пример расчета энергии спирали.

Немного о полимерах Другим игроком в формуле (10.4) является энтропия. Мы помним, что энтропия – это логарифм числа микросостояний, соответствующих макросостоянию. Для свободного полимера есть очень большое возможное число конформаций (рис.10.10А). Представим себе, что все молекулы своими концами прикреплены к двум плоскостям (рис.10.10В). Число конформаций по-прежнему достаточно велико. Теперь одну из стенок подвинем (рис.10.10С). Число допустимых конформаций заметно сократилось – все молекулы вытянулись, следовательно энтропия уменьшилась, а (свободная) энергия увеличилась. Поэтому, чтобы растянуть полимер надо

приложить силу, хотя мы не напрягли ни одной химической связи. Эластичность полимеров определяется *энтропией*. Чтобы в этом убедиться Можно поставить два домашних опыта.

Опыт 1. Берем простую резинку (ту, которой обычно перехватывают пачку долларов). Натягиваем ее и держим некоторое время в натянутом состоянии. Потом резко снимаем напряжение и подносим к губам, поскольку губы весьма чувствительны к температуре. Она холодная! При снятии напряжения резинка поглощает тепло, чтобы полимерные молекулы приобрели свободу. Мало того резинка далеко не сразу придет к исходной длине, и ее упругость будет не такой, как в начале опыта – ей надо поглотить тепло и восстановить свою энтропию. Поскольку упругость определяется энтропией, то чем теплее, тем более упругим будет полимер.

Опыт 2. Берем дощечку. Вбиваем два гвоздика (или две кнопки). Наматываем между ними капроновую (она упругая – это важно!) нитку с некоторым натяжением. Кладем в морозильник. Через 15 минут вынимаем. От натяжения не осталось и следа – нить провисла. Оставляем конструкцию в тепле. Нить снова натянулась. При охлаждении полимерная нить расширяется, а при нагревании – сжимается!

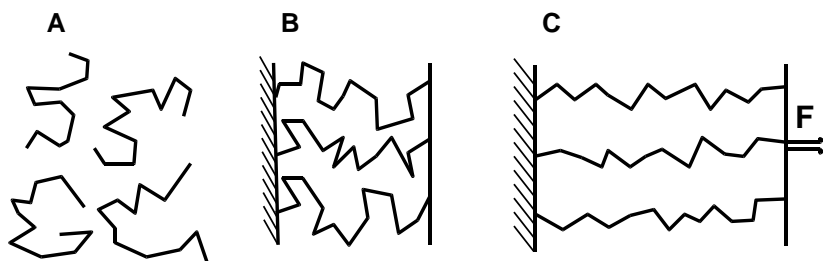


Рис. 10.10: Полимеры. А – свободные полимерные молекулы; В – молекулы, прикрепленные к стенкам; С – к стенкам приложена сила

В структуре РНК петли – это свободные полимеры, у которых сближены концы. Если концы петли не сближены, то этот полимер может принимать очень много конформаций. Если же появилось ограничение – концы сближены, то количество конформаций уменьшилось, энтропия также уменьшилась, а энергия – увеличилась. Из физики полимеров известно, что изменение свободной энергии идеального полимера при фиксации концов полимера равна

$$\Delta G = \frac{3}{2} RT \ln L$$

где L – число звеньев полимера. Идеальный полимер – это полимер, у которого объем звеньев равен 0, и он может свободно изгибаться в любом направлении. Можно показать, что эффект исключенного объема и ограничения в гибкости полимера, обусловленные валентными углами, приводит к тому, что вместо числа звеньев надо использовать число звеньев, делен-

ное на некоторую константу, которая называется персистентной длиной. В результате формула примет вид

$$\Delta G = \frac{3}{2}RT \ln L + B$$

где B – некоторая константа, зависящая от свойств полимера. Эта формула верна для достаточно длинных полимеров. Итак, чтобы вычислить свободную энергию структуры, надо вычислить энергию спиралей и энергию петель. Для определения параметров было проведено множество экспериментов с искусственными олигонуклеотидами, в которых методами микрокалориметрии напрямую измерялись термодинамические параметры.

Энергия спирали Энергия спирали вычисляется как сумма энергий стекингов. Для вычисления энергии стекинга используют таблицы (например, табл.10.1).

Таблица 10.1: Таблица для энергии стекинга

5' 3'	A=U 3' 5'	C=G 3' 5'	G=C 3' 5'	G=U 3' 5'	U=A 3' 5'	U=G 3' 5'
A=U	-1.1	-2.1	-2.2	-0.6	-0.9	-1.4
C=G	-2.1	-3.3	-2.4	-1.4	-2.1	-1.4
G=C	-2.4	-3.4	-3.3	-1.5	-2.2	-2.5
G=U	-1.3	-2.5	-2.1	-0.5	-1.4	-1.3
U=A	-1.3	-2.4	-2.1	-1.0	-0.9	-1.3
U=G	-1.0	-1.5	-1.4	-0.3	-0.6	-0.5

Вычислим, например, энергию спирали (рис. 10.9). Первая пара A=U, причем A находится на 5' конце. Смотрим в таблицу строку A=U. Вторая пара C=G, стекинг $\begin{smallmatrix} A=U \\ C=G \end{smallmatrix}$. Ищем в этой строке столбец C=G, получаем -2.1. Вторая пара C=G, а за ней A=U. В таблице ищем строку C=G и столбец A=U. Получаем -2.1. Далее пара A=U и G=C. Энергия -2.2. Следующий стекинг -G=C и C=G. Энергия -3.4. Итого $-2.1-2.1-2.2-3.4 = -9.8$. Обратим внимание, что матрица (10.1) несимметрична, поэтому надо следить за направлением 5' → 3'.

Для энергии петель также используют таблицы (например, Табл.10.2). Если длина петли больше, чем есть табличные значения, то используют формулу. Для разных типов петель используют разные строки таблицы. Для шпилек при $L=3..5$ кроме энтропии есть некоторое напряжение структуры и значительный исключенный объем. При вычислении энергии выпячивания (Bulge) сохраняют стекинг. Для внутренних петель и для мультипетель L равна суммарной длине петель плюс количество ветвей: $L = \sum L_i + n_{loops} - 2$. В современных программах используют более сложные

Таблица 10.2: Таблица для энергии петель

Loop length	1	2	3	4	5	6	7	8
Internal		1.3	1.5	1.7	1.8	2.0	2.2	2.3
Bulge	3.8	2.8	3.2	3.6	4.0	4.4	4.6	4.7
Hairpin			5.7	5.6	5.4	5.9	5.9	5.6

схемы вычисления энергии, в частности, при вычислении энергии петель учитывают нуклеотидный состав петли и тип замыкающей пары. При вычислении энергии стекинга учитывают также возможные неканонические пары, такие, как А=С, поскольку они, хоть и не образуют водородные связи, сохраняют стекинг.

10.5.1 Алгоритм Зукера

Казалось бы, в алгоритме Нуссинофф можно было бы вместо того, чтобы на каждое спаривание добавлять 1, добавлять энергию и направить алгоритм на минимизацию (max заменить на min). Однако для предсказания вторичной структуры РНК с помощью минимизации энергии алгоритм Нуссинофф применить не удастся. Основная проблема заключается в том, что энергия петель положительна.

Дело в том, что алгоритм работает изнутри наружу, от маленьких структур к большим. При образовании первой пары нуклеотидов в петле мы должны дать положительную энергию. Это не выгодно с точки зрения алгоритма, поэтому первая пара не может быть распознана в принципе (рис.10.11). То же самое можно сказать и про вторую и последующие пары – ведь они при образовании формируют петлю.

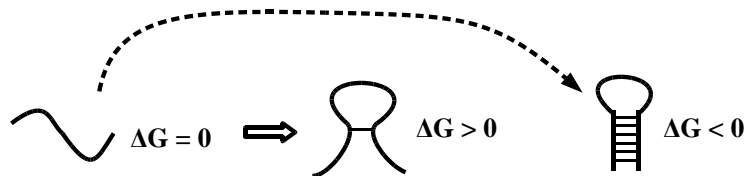


Рис. 10.11: Образование структуры требует преодоления энергетического барьера

Зукер предложил модификацию алгоритма динамического программирования, который позволяет преодолевать энергетический барьер формирования петли. Сначала введем определение.

Определение При заданной структуре S нуклеотид k считается *доступным* для пары $(i < j)$, если НЕ существует пары $(r < s)$ такой, что $i < r <$

k & $k < s < j$. Множество доступных нуклеотидов для любой пары называется *петлей*. (i, j) – *закрывающая* пара (рис.10.12А). Стекинг – частный случай петли. Тогда энергия структуры это сумма энергий петель (включая стекинг):

$$\Delta G = \sum_{loops} \Delta G_{loop}$$

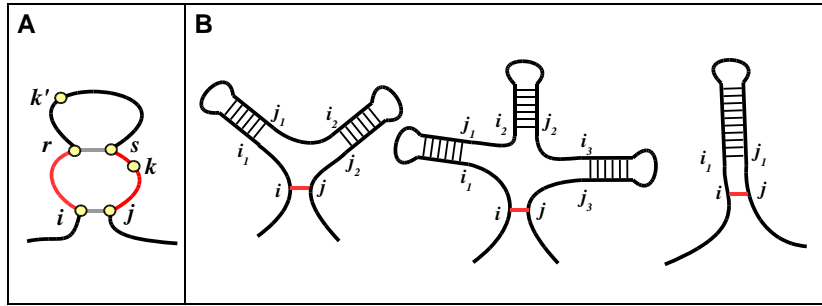


Рис. 10.12: Петли. А – Определение петли (красная). $i \circ j$ – закрывающая пара. Нуклеотид k доступен для пары $i \circ j$. Нуклеотид k' не доступен для пары $i \circ j$ – его экранирует пара $r \circ s$. В – перебор возможных петель для пары $i \circ j$

Для построения алгоритма динамического программирования введем две переменные.

$W(i, j)$ – оптимальная энергия структуры фрагмента $[i, j]$

$V(i, j)$ – оптимальная энергия структуры фрагмента $[i, j]$ при условии, что нуклеотиды $i \circ j$ спарены.

Для этих переменных можно написать рекурсию:

$$V(i, j) = \min_{i < i_1 < j_1 < i_2 < j_2 < \dots < j_n} \sum_k V(i_k, j_k) + \Delta G_{loop}(i_1, j_1; i_2, j_2; \dots) \quad (10.5)$$

$$W(i, j) = \min \begin{cases} W(i+1, j), & i \text{ не спарено} \\ W(i, j-1), & j \text{ не спарено} \\ V(i, j), & i \text{ спарено с } j \\ \min_k W(i, k-1) + W(k, j) & i \text{ спарено с } k \end{cases} \quad (10.6)$$

Рекурсия, также как и в алгоритме Нуссинофф идет от меньших фрагментов к большему. Для восстановления структур, также как и в алгоритме Нуссинофф запоминаем, где реализуется максимум. При работе алгоритма значение энергии $V(i, j)$ может быть и положительным. Однако, когда соберется закрывающая спираль, энергия $V(i, j)$ станет отрицательной, таким

образом энергетический барьер будет преодолён. При вычислении значения $V(i, j)$ надо перебрать множество вариантов петель (рис.10.12В). Время рекурсии (10.6) равно $T \sim O(L^3)$, однако время работы рекурсии (10.5) оказывается порядка $T \sim O(2^L)$, что очень много.

Дело в том, что мы должны перебрать все возможные варианты внутренних подструктур. Основную трудность в вычислении $V(i, j)$ представляют мультипетли – именно на них происходит большой перебор. Для решения этой проблемы применяется множество эвристик и упрощений. Например, ставятся ограничения на размер мультипетли и на количество ветвлений. Кроме того, если предположить, что энергия петли линейно зависит от длины, то удастся кардинально сократить перебор.

10.5.2 mFold – качество предсказания

Алгоритм Зукера реализован в виде веб-сервера mFold, а также доступен исходный код программы, который очень внятно написан и его можно адаптировать под свои задачи. Однако, испытания алгоритма на последовательностях из «золотого стандарта» показывают, что только около 60% последовательностей сворачиваются в правильную структуру. Для этого есть несколько причин:

- Большинство последовательностей в золотом стандарте – это последовательности тРНК. Но некоторые основания в этих молекулах подвержены посттранскрипционным модификациям, что влияет на склонность тех или иных оснований вступать в комплементарные взаимодействия.
- тРНК имеют консервативные третичные взаимодействия, в частности, две петли (D и T ψ) образуют псевдоузел типа kissing loops.
- Для уникального и эффективного сворачивания гетерополимер (РНК или белок) должны обладать специфическим спектром энергий – энергия правильной структуры должна существенно отличаться от энергии остальных допустимых структур (рис.10.13). Если для белков это, как правило, верно, то для РНК спектр энергий достаточно плотно заполнен в окрестности минимума. Высокая плотность спектра около оптимума говорит о том, что оптимальная структура не устойчива. Стоит чуть пошевелить параметры, как оптимальная структура перестанет быть таковой, а оптимальной станет другая структура.

Есть еще несколько особенностей. Можно показать, что случайные бернуллиевские последовательности очень хорошо сворачиваются во вторичную структуру (рис.10.14). В среднем только треть остатков остаются не спаренными. Сервер mFold дает весьма наглядную выдачу. Кроме того, он предоставляет дополнительные возможности. Можно, например, добавить ограничения, которые позволяют задать некоторые спаривания. Если из каких-либо соображений пользователь знает про спаривание некоторых оснований, то можно внести эти ограничения.

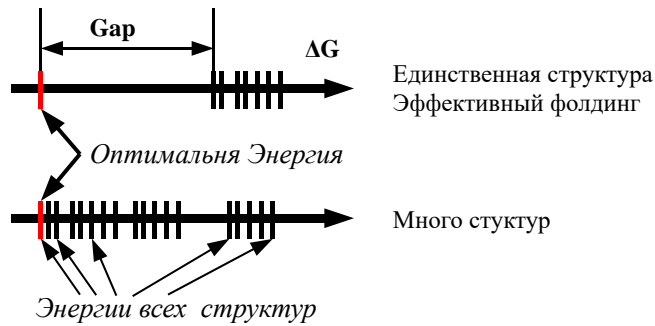


Рис. 10.13: Энергетические спектры

С одной стороны, такие ограничения являются полезными, а с другой — позволяют фиксировать спаренные основания и рисовать *любые* структуры, которые приятны пользователю. При этом в соответствующей статье можно честно сказать, что нарисованная структура получена сервером mFold, но при этом можно умолчать, что использовались ограничения. Поэтому надо проявлять определенную осторожность при анализе предсказанных структур. Возникает, естественно, вопрос — а как предсказывать и анализировать вторичную структуру? Есть несколько ответов. Во-первых, можно попробовать искать субоптимальные структуры, а во-вторых — искать эволюционно консервативные структуры.

10.5.3 Статистическая сумма

Прежде, чем искать субоптимальные структуры, оценим вероятность того, что какая-то пара нуклеотидов спарена. Для этого надо просуммировать вероятности всех структур, в которых указанные нуклеотиды образуют комплементарные пары. Вероятность реализации структуры S определяется из распределения Больцмана:

$$\Pr(S) = \frac{e^{-\frac{\Delta G}{RT}}}{Z}; \quad Z = \sum_{all \ structures} e^{-\frac{\Delta G}{RT}}$$

величина Z называется статистической суммой. Вероятность того, что нуклеотиды $i \circ j$ спарены — это сумма вероятностей структур, в которых $i \circ j$ спарены:

$$\Pr(i \circ j) = \frac{Z(i \circ j)}{Z} \quad (10.7)$$

величину $Z(i \circ j)$ будем называть условной статистической суммой. Если мы зафиксируем две позиции i, j . То любая структура распадается на внешнюю и внутреннюю подструктуры (рис.10.8). Полная энергия структуры распа-

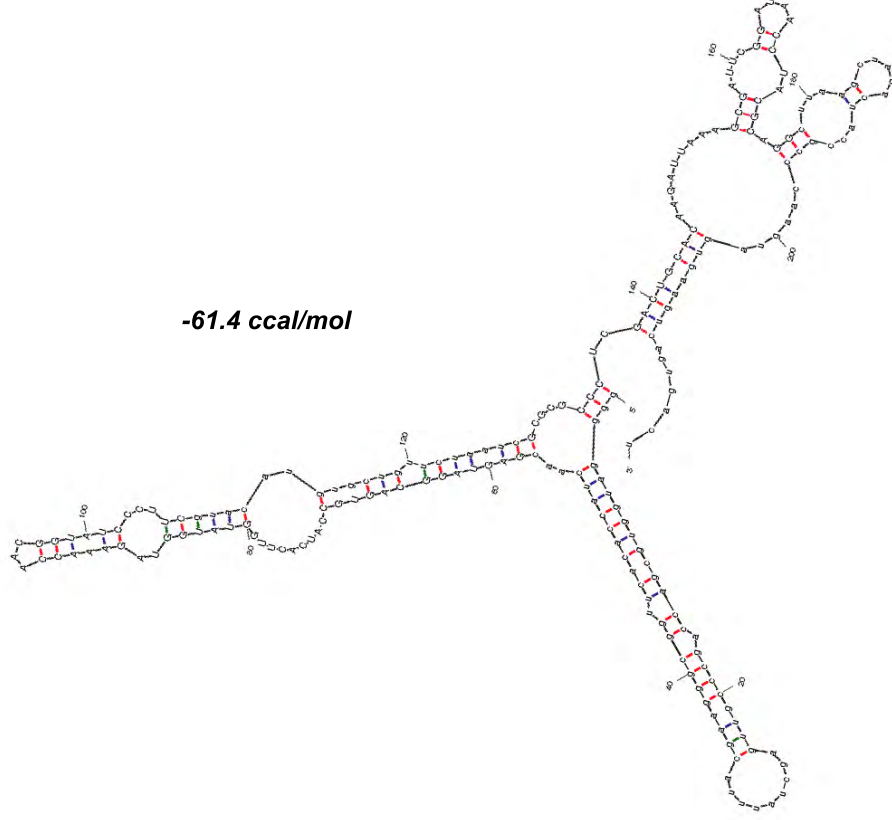


Рис. 10.14: Оптимальная структура случайной сгенерированной последовательности.

дается на энергию внешней и внутренней структур:

$$Z(i, j) = \sum e^{-\frac{\Delta G}{RT}} = \sum e^{-\frac{G^{in} + G^{out}}{RT}} = \sum e^{-\frac{G^{in}}{RT}} \times \sum e^{-\frac{G^{out}}{RT}} = I(i, j) \times O(i, j)$$

Теперь можно построить рекурсию для подсчета внутренней ($In(i, j)$) и внешней ($Out(i, j)$) статистических сумм. Расчет внутренней статистической суммы (рис.10.15):

$$\begin{aligned}
 I(i, j) &= \sum_{i < i_1 < j_1 < i_2 < \dots < j} \exp \left(-\frac{\Delta G_{loop}(i, i_1, j_1, \dots, j) + \Delta G_{i_1 \circ j_1} + \dots}{RT} \right) \\
 &= \exp \left(-\frac{\Delta G_{loop}(i, i_1, j_1, \dots, j)}{RT} \right) \prod_{i_s j_s} I(i_s, j_s)
 \end{aligned} \tag{10.8}$$

Суммирование производится по всем возможным комбинациям промежуточных позиций. Здесь возникает та же проблема большого перебора, что и при вычислении $V(i, j)$ в алгоритме Зукера. И она решается с помощью тех же эвристик. Аналогично получаем внешнюю статистическую сумму

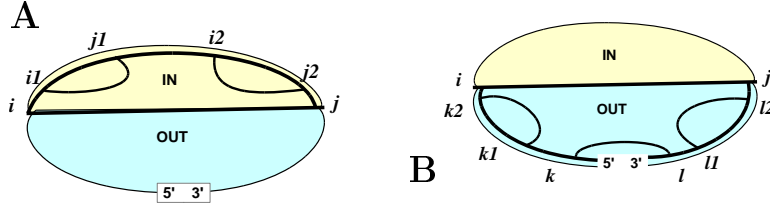


Рис. 10.15: Внутренняя (А) и внешняя (В) статистические суммы.

(рис.10.15). При этом учитываем, что только одна из дуг (kl) проходит через концы последовательности и дает вклад O , а остальные элементы – это внутренние подструктуры:

$$\begin{aligned}
 O(i, j) &= \sum_{i < i_1 < j_1 < i_2 < \dots < j} \\
 &\exp \left(- \frac{\Delta G_{loop}(i, k_1, k_2, \dots, j) + \Delta G_{k_1, k_2} + \dots + \Delta G_{k, l} + \dots + \Delta G_{l_1, l_2}}{RT} \right) \\
 &= \exp \left(- \frac{\Delta G_{loop}(i, k_1, \dots, k, l, l_1 \dots, j)}{RT} \right) \prod_s I(k_s, k_{s+1}) \prod_s I(l_s, l_{s+1}) \cdot O(kl)
 \end{aligned} \tag{10.9}$$

Теперь, вычислив внутренние и внешние статистические суммы, мы получим условную статистическую сумму $Z(i, j)$. Полная статистическая сумма есть просто $Z = Z(0, L) = I(0, L) \cdot O(0, L)$. Поскольку внешняя статистическая сумма для сегмента $[0, L]$ равна 1, то $Z = I(0, L)$. Окончательно имеем:

$$\Pr(i \circ j) = \frac{I(i, j) \cdot O(i, j)}{I(0, L)} \tag{10.10}$$

Эта вероятность дает гораздо больше информации, чем просто предсказанная структура. В частности, она дает представление о надежности предсказания того или иного спаривания. Она также позволяет строить субоптимальные структуры.

10.5.4 Субоптимальные структуры

Для построения субоптимальных структур используем вероятности спаривания (10.10). Допустим, мы нашли оптимальную структуру (рис.10.16 красные точки). Выберем среди потенциальных спариваний *не* принадлежащих оптимальной структуре новое спаривание $i_1 \circ j_1$. Причем выбор сделаем методом Монте-Карло с вероятностями, пропорциональными вероят-

ностям спаривания (10.10). Построим условно-оптимальную структуру при условии спаривания $i_1 \circ j_1$.

Субоптимальная структура может иметь некоторое количество общих пар с оптимальной структурой, но она гарантированно не совпадет с оптимальной. Чтобы получить еще одну субоптимальную структуру, исключим из потенциальных спариваний как оптимальную, так и субоптимальную структуру. Так же выберем еще одну новую пару для инициации следующей субоптимальной структуры. Так можно повторить несколько раз и получить заранее заданное количество *разных* субоптимальных структур. Веб-сервер mFold предоставляет возможность получить заданное количе-

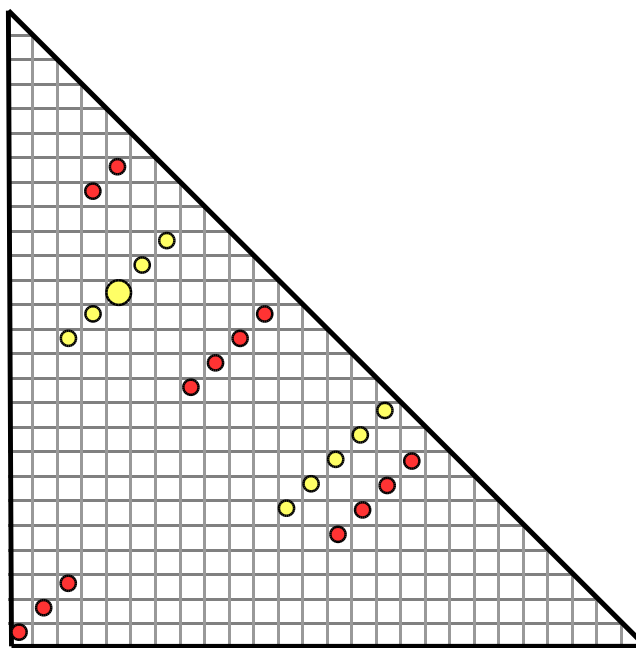


Рис. 10.16: Построение субоптимальной структуры. Красные – спаривания на оптимальной структуре, большой желтый кружок – выбранное альтернативное спаривание, желтые – спаривания на субоптимальной структуре.

ство субоптимальных структур. При тестировании на «золотом стандарте» правильные структуры как правило, находятся среди субоптимальных. У пользователя часто возникает недоумение – при повторении запроса он получает разные наборы субоптимальных структур. Надо понимать, что при генерации субоптимальных структур используется метод Монте-Карло с генератором случайных чисел, поэтому на разных запусках программы мы *можем* получать разные результаты.

Глава 11

Консенсусные структуры РНК

11.1 Консенсусные вторичные структуры

Для функциональных РНК, у которых есть гомологи, наиболее эффективным методом является поиск консервативных структур. Именно таким образом (с привлечением дополнительных биохимических данных) были предсказаны вторичные структуры рибосомных РНК. Это открытие, в частности, позволило Карлу Ричарду Вёзе открыть новое царство – царство архей. Этот же подход позволил открыть структуру и механизм действия рибопереклюателей. Здесь есть следующие задачи:

- Дан набор последовательностей, которые, по-видимому, должны иметь общую структуру. Задача – описать общую структуру.
- Дан набор последовательностей с общей структурой, создать алгоритм, который позволил бы проверять, подходит ли новая последовательность под этот профиль.

Первая задача была решена, в частности, при предсказании структуры тРНК, рРНК, рибопереклюателей и других структур. Вторая задача возникает, например, когда в новом геноме необходимо предсказать гены тРНК, рРНК и других функциональных РНК.

11.1.1 Метод ковариаций

Одним из подходов к проблеме предсказания вторичной структуры РНК основан на анализе множественных выравниваний (рис.11.1) Сохранение структуры при эволюции предполагает коррелированные замены в спиральных участках. например замена $A \rightarrow G$ должна сопровождаться компенсаторной заменой в комплементарной позиции $U \rightarrow C$, чтобы сохранить

Рис. 11.1: Множественное выравнивание тРНК. Предпоследняя строка показывает консервативные спиральные участки, последняя строка – консенсус.

The diagram illustrates three types of RNA secondary structure motifs:

- Single Stem-Loop:** A single stem-loop structure with the sequence A=U, G=C, G=U, C=G, and 5' 3' ends.
- Multi-stemmed Structure:** A multi-stemmed structure with the sequence G=C, G=U, C=G, and 5' 3' ends.
- Complex Multi-stemmed Structure:** A complex multi-stemmed structure with the sequence CGGA---UCUG, CGGG---CCUG, GAGA---UUUC, and 5' 3' ends.

сованность замен в двух колонках A и B можно вычислить их взимную информацию:

$f_{\alpha\beta}$ – частота одновременной (в одной последовательности) встречи буквы α в колонке A и буквы β в колонке B ; f_α, f_β – частоты встречаемости букв

α и β в колонках A и B . Числитель дроби отслеживает согласованность замен, знаменатель – частоты встречаемости букв в колонках. Заметим, что если две колонки идентичны, то числитель равен знаменателю, логарифм равен 0 и мы никакой информации о согласованности замен не получим. Еще заметим, что если колонка A согласована с колонкой B , а колонка B согласована с колонкой C , то колонка A также будет согласована с колонкой C .

Метод, основанный на анализе согласованных замен, называется методом ковариаций. Для того, чтобы он был применим необходимо два условия: 1) последовательности должны быть не слишком далекими, чтобы можно было построить достоверное выравнивание; 2) последовательности не должны быть слишком близкими, чтобы получать заметные значения взаимной информации. Для восстановления структур можно применить алгоритм Нуссинофф, с некоторыми отличиями. Во-первых, рассматривается не отдельная последовательность, а целое множественное выравнивание, во-вторых, за образование одной пары мы будем давать не 1, а значение соответствующей взаимной информации.

Кстати, одним из экспериментальных подходов к подтверждению функциональности вторичных структур заключается в том, что вводится, как правило, одиночная замена, разрушающая спаривание, и, тем самым, структуру, а затем вводится компенсаторная замена, восстанавливающая спаривание. Если РНК с одной заменой не функциональна, а с двумя – функциональна, то есть достаточные основания считать, что структура важна. В реальных случаях делают несколько независимых замен.

11.1.2 Одновременное выравнивание последовательностей и структур

Санкофф предложил алгоритм одновременного выравнивания и построения общей структуры для двух последовательностей. Этот алгоритм достаточно громоздкий, чтобы его здесь рассматривать, но отметим, что его вычислительная сложность составляет $T \sim O(L^6)$. На его основе предложен алгоритм LocARNA (local alignment RNA), который с использованием нескольких эвристик и алгоритма Санкофф строит прогрессивное множественное выравнивание. Сейчас этот алгоритм является одним из лучших алгоритмов одновременного построения множественного структурного выравнивания РНК.

11.2 Порождающие грамматики

Значительная часть предыдущих лекций была посвящена *моделям* порождения последовательностей. В частности, скрытые марковские модели – это модели порождения последовательностей. Знаменитый лингвист Ноам Хомский в конце 50 годов построил иерархию порождающих грамматик, которая легла в основу теории формальных языков.

Итак, для описания формальных грамматик введем понятия. Дан конечный алфавит Σ . Элементы языка (буквы) будем называть *терминальными* символами и обозначать малыми буквами $a \cdots z \in \Sigma$.

Есть еще *нетерминальные* символы, которые будем обозначать заглавными символами $A \cdots Z$. Для простоты понимания можно считать, что это промежуточные подстроки.

Грамматика – это способ описать слова из заданного языка $L \subseteq \Sigma^*$ – подмножества слов из множества всех возможных слов над языком. Для описания используют *правила подстановки*. Есть один выделенный нетерминальный символ S , который соответствует исходной строке. Правила подстановки это способ заменить одну группу символов (терминальных и нетерминальных) на другую группу символов. Правило подстановки записывается в виде $\alpha \rightarrow \beta$, где α , β – это комбинации из терминальных и нетерминальных символов. Среди правил подстановки должно быть хотя бы одно правило, в левой части которого стоит нетерминальный символ S , обозначающий входную строку.

Чтобы понять, что происходит рассмотрим такую грамматику:

$$\begin{array}{lll} 1 & S & \rightarrow W_1 a \\ 2 & W_1 & \rightarrow W_2 b \\ 3 & W_2 & \rightarrow W_1 a \\ 4 & W_2 & \rightarrow \epsilon \end{array}$$

Здесь ϵ означает пустую строку. Каждое правило – это подстановка. Первое правило обозначает, что строка S может быть представлена как буква a и что-то еще, которое мы обозначили (нетерминальным) символом W_1 . Из первого правила следует, что строка обязательно кончается на a . Второе правило говорит, что подстрока типа W_1 заканчивается на b . Третье правило говорит, что подстрока типа W_2 заканчивается на a . Последнее правило говорит, что W_2 может быть пустой строкой. Эта грамматика порождает строки типа $baba \cdots baba$. Действительно, сначала применяем правило 4. Получаем пустую строку W_2 . К W_2 можно применить только правило 2. Получим строку W_1 . К W_1 можно применить либо правило 3 и получить строку ba либо правило 1 и закончить генерацию строки. Грамматика

$$\begin{array}{lll} 1 & S & \rightarrow W_1 a \\ 2 & W_1 & \rightarrow W_1 b \\ 3 & W_1 & \rightarrow \epsilon \end{array}$$

порождает строки вида $bbb \cdots ba$. Обратим внимание, что нетерминальный символ может появляться в левой и в правой части несколько раз. В левой и правой части могут появляться несколько нетерминальных символов. Например,

$$\begin{array}{lll} 1 & S & \rightarrow W_1 W_2 \\ 2 & W_1 & \rightarrow W_1 a \\ 3 & W_1 & \rightarrow \epsilon \\ 4 & W_2 & \rightarrow W_2 b \\ 5 & W_2 & \rightarrow \epsilon \end{array} \tag{11.2}$$

порождает строку $aaa \cdots bbb$, причем может быть любое (в том числе 0) количество символов a и b . Действительно, правила 2,3 порождают поли- a , а правила 4,5 порождают поли- b , а правило 1 объединяет эти строки.

Грамматика порождает строку. Однако часто стоит задача – дана грамматика и строка. Надо определить, может ли данная строка быть порождена данной грамматикой, и какая последовательность правил была использована. Эта задача называется задачей разбора (parsing) строки. Вот разбор строки $aabbb$ грамматикой (11.2)

$$\begin{aligned} S &\rightarrow W_1 W_2 \rightarrow W_1 a W_2 \rightarrow W_1 a a W_2 \rightarrow a a W_2 \\ &\rightarrow a a W_2 b \rightarrow a a W_2 b b \rightarrow a a W_2 b b b \rightarrow a a b b b \end{aligned}$$

При порождении строки $aabbb$ была применена следующая цепочка правил: 1,2,2,3,4,4,5. Для упрощения записи грамматик можно использовать объединенные правила. Например, вместо двух правил $W \rightarrow aW$; $W \rightarrow bW$ напишем правило $W \rightarrow aW \mid bW$, а вместо пары правил $W_1 \rightarrow aW_2$; $W_3 \rightarrow bW_2$ напишем $W_1 \rightarrow abW_3$.

11.2.1 Иерархия грамматик

Ноам Хомский ввел следующие классы грамматик:

Регулярные грамматики Используются только правила вида
 $W \rightarrow aW$, $W \rightarrow a$

Контекстно-свободные грамматики Правила вида
 $W \rightarrow \beta$
 где β – любая комбинация терминальных и нетерминальных символов.

Контекстно-зависимые грамматики Правила вида
 $\alpha_1 W \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$
 где $\alpha_1, \alpha_2, \beta$ – любая комбинация терминальных и нетерминальных символов.

Неограниченные грамматики Правила вида
 $\alpha_1 W \alpha_2 \rightarrow \gamma$

Обратим внимание на то, что указанные типы грамматик образуют вложенную структуру (рис.11.3). Регулярные грамматики эквивалентны регулярным выражениям. Построим, например, регулярную грамматику для выражения $[ab]^*(bb)^+$, что значит повторить несколько раз (в том числе, 0 раз) пару символов ab , потом повторить не менее одного раза комбинацию bb .

$$\begin{array}{lll} 1 & S & \rightarrow W_1 \\ 2 & W_1 & \rightarrow W_2 \\ 3 & W_1 & \rightarrow abW_1 \\ 4 & W_2 & \rightarrow bbW_2 \\ 5 & W_2 & \rightarrow bb \end{array}$$



Рис. 11.3: Классификация грамматик по Хомскому

Нетерминальные символы W_1 , W_2 соответствуют поли- ab и поли- bb соответственно. Правила 4,5 порождают комбинацию $(bb)^+$, правила 2,3 приписывают перед серией bb^+ символы ab . Наконец, из этого получаем финальную строку. Разбор регулярных выражений обсуждался в курсе алгоритмов.

11.3 Контекстно-свободные грамматики и структура РНК

Как уже упоминалось, структура РНК без псевдоузлов эквивалентна скобочному выражению. Скобочные выражения встречаются весьма часто и не только в арифметических и алгебраических формулах. Например, язык HTML также построен на скобочных выражениях, поскольку в нем есть открывающие и закрывающие тэги, которые, по сути, являются скобками разных типов. Скобочные выражения не могут быть описаны с помощью регулярных грамматик, поскольку открывающая скобка предполагает появление закрывающей через некоторое неизвестное заранее количество символов.

Для описания скобочных выражений лучше всего подходят контекстно-свободные грамматики, поскольку в этом классе грамматик можно писать правила типа $W_1 \rightarrow oW_2c$, где o, c означают открывающую и закрывающую скобки. Для РНК характерно образование комплементарных пар, которые, при исключении псевдоузлов, могут быть представлены в виде скобочных выражений, поэтому для описания вторичной структуры РНК лучше всего подходят контекстно-свободные грамматики.

Пусть, например, мы хотим описать структуру типа две спирали-шпильки (рис.11.4А). При этом потребуем, чтобы петля была не короче 3. Грамма-

тика будет примерно такой:

$$\begin{array}{lll}
 1 & S & \rightarrow HH \\
 2 & H & \rightarrow aHt \mid cHg \mid gHc \mid tHa \\
 3 & H & \rightarrow aL_3t \mid cL_3g \mid gL_3c \mid tL_3a \\
 4 & L_3 & \rightarrow aL_3 \mid cL_3 \mid gL_3 \mid tL_3 \\
 5 & L_3 & \rightarrow aL_2 \mid cL_2 \mid gL_2 \mid tL_2 \\
 6 & L_2 & \rightarrow aL_1 \mid cL_1 \mid gL_1 \mid tL_1 \\
 7 & L_1 & \rightarrow a \mid c \mid g \mid t
 \end{array} \quad (11.3)$$

Здесь правила 4-6 формируют петлю длиной от 3 нуклеотидов. Наличие трех нетерминалов L_i обеспечивает минимальную длину петли. Правило 3 задает начало спирали – спираль должна иметь хотя бы одно спаривание, правило 2 – удлиняет спираль. Правило 1 завершает формирование структуры, объединяя шпильки.

11.3.1 Разбор строки

Одной из задач, связанных с грамматиками, является задача разбора строки. Дана грамматика и строка. Надо указать последовательность правил, которые генерируют данную строку. Вложение строки в контекстно-свободную грамматику может быть представлено в виде дерева (рис.11.4)

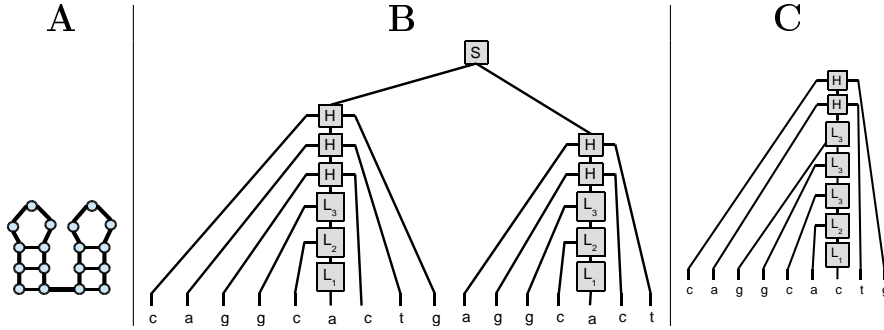


Рис. 11.4: структура (А) и дерево вложения строки в грамматику (В). С – альтернативное вложение левой шпильки в грамматику

Видно, что в дереве разбора узлы имеют разное количество ветвей. *Нормальным представлением* контекстно-свободной грамматики называется такое представление, когда правая часть правила имеет вид $W \rightarrow a$ или $W \rightarrow W_1W_2$. Контекстно-свободную грамматику всегда можно привести к каноническому виду, введя новые нетерминалы и новые правила. Например, если есть правило $W \rightarrow cWg$, которое содержит справа три символа, то его можно заменить на правила $W_c \rightarrow c$; $W_g \rightarrow g$; $W_l \rightarrow W_cW$; $W \rightarrow W_lW_g$, т.е. первое и второе правила просто порождают терминальные символы, W_l помнит, что было слева, и третье правило определяет левую часть, наконец,

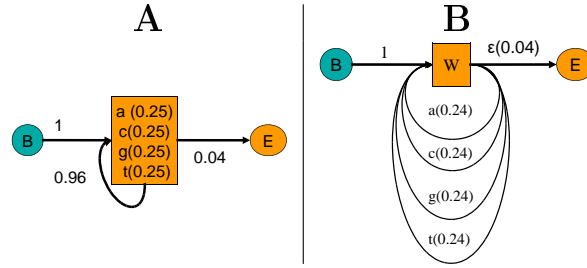


Рис. 11.5: Автоматы Мура (А) и Мили (В), порождающие бернуллиевские последовательности.

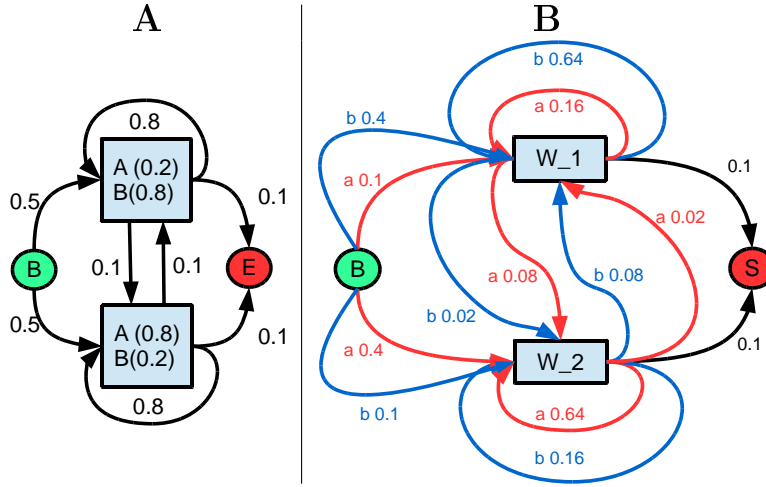


Рис. 11.6: НММ двух состояний (А) и автомат Мили (В)

Сначала мы считаем, что есть полная строка, которая может быть представлена как один из двух нетерминалов. Вероятность полной строки равна 1 (строка дана). Вероятности правил 11,12 определяют вероятности нетерминалов W_1 , W_2 . Далее, в зависимости от первого символа, определяем вероятности нетерминалов. В результате получаем граф, аналогичный графу состояний НММ (рис.11.7). С помощью динамического программирования находим оптимальный (наиболее вероятный) путь на этом графе.

Упражнение 1. Напишите рекурсию для разбора строки с помощью регулярной грамматики для примера рис.11.6В. Не забудьте про обратный проход.

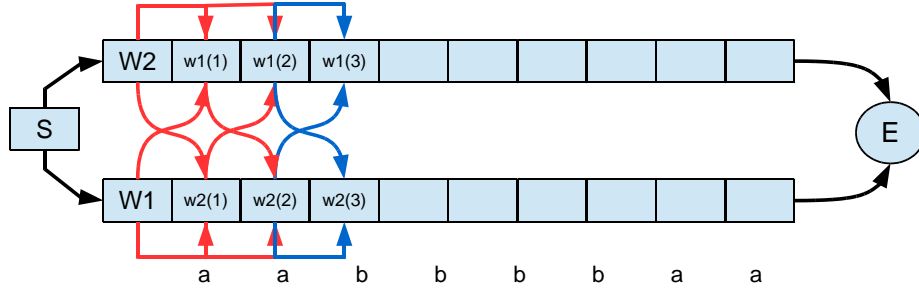


Рис. 11.7: Граф разбора строки.

Упражнение 2. Напишите рекурсию для разбора строки с помощью регулярной грамматики для общего случая.

Упражнение 3. Реализуйте рекурсию для примера рис.11.6В на каком-нибудь языке программирования.

11.3.3 Вероятностные контекстно-свободные грамматики.

Для разбора строки в вероятностной контекстно-свободной грамматике, также как и НММ можно поставить две задачи. 1) найти наиболее вероятное вложение строки в грамматику и 2) найти вероятности использования того или иного правила для фрагмента последовательности, например, вероятность того, что i и j спарены и принадлежат антикодновому стеблю (аналог forward-backward).

Контекстно-свободная грамматика описывает структуру РНК. Каждое правило и каждый нетерминал соответствует определенному элементу вторичной структуры. Поэтому решение задачи вложения последовательности в грамматику дает нам наилучший способ установить соответствие последовательности заданной структуре.

Поиск наилучшего вложения последовательности в грамматику.

Введем обозначения. Пусть в грамматике есть нетерминальные символы $W_1 \dots W_M$. Пусть грамматика приведена к нормальной форме и у нас есть только правила вида

$$W_v \rightarrow a \quad (11.4)$$

$$W_v \rightarrow W_x W_y \quad (11.5)$$

Вероятности, соответствующие правилам типа (11.4) обозначим $e_v(a)$ и будем называть эмиссионными, а вероятности правил (11.5) обозначим $t_v(k, l)$

и будем называть переходными. Для поиска наилучшего вложения строки x в грамматику применим динамическое программирование, аналогичное алгоритму Нуссинофф. Только будем заполнять трехмерную матрицу $\gamma(i, j, v)$, соответствующую паре позиций i, j и нетерминалу W_v . Также как и в алгоритме Нуссинофф, заполнять матрицу будем изнутри-наружу. Для начала заполним все позиции i, i, v , соответствующие отрезкам нулевой длины. Здесь возможны только правила типа (11.4):

$$\gamma(i, i, v) = e_v(x_i)$$

если для какого-то нетерминала нет правила типа (11.4), то соответствующую эмиссионную вероятность считаем равной 0. Далее используем только правила типа (11.5) Найдем наилучшую вероятность поддерева разбора подстроки $x_i \cdots x_j$ с корнем в нетерминале W_z (рис.11.8). Вероятность

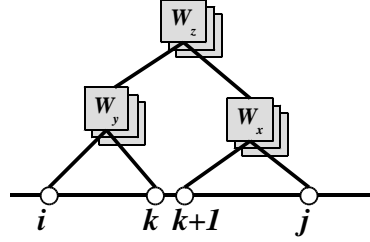


Рис. 11.8: Оптимальный разбор строки.

нетерминала W_z при использовании правила (11.5) есть произведение вероятностей дочерних нетерминалов $\Pr(W_z) = \Pr(W_x) \cdot \Pr(W_y)$. Чтобы найти наилучшую вероятность нам надо перебрать все промежуточные значения k и все возможные дочерние нетерминалы x, y :

$$\gamma(i, j, z) = \max_{k=i \cdots j-1} \max_{x, y} \gamma(i, k, x) \cdot \gamma(k+1, j, y)$$

заметим, что при переборе возможных k мы просматриваем также интервалы нулевой длины, здесь мы и используем эмиссионные вероятности. Чтобы найти оптимальный разбор, т.е. чтобы восстановить оптимальное дерево разбора, нам надо запоминать, где реализовался максимум $\pi(i, j, z)$ — это тройка чисел (k, x, y) :

$$\pi(i, j, z) = \arg \max_{k=i \cdots j-1, x, y} \gamma(i, k, x) \cdot \gamma(k+1, j, y)$$

Для восстановления оптимального дерева разбора используется алгоритм, аналогичный восстановлению структуры в алгоритме Нуссинофф, только мы еще восстанавливаем нетерминалы:

```
1 GetTree(){
```

```

2   RestoreTree(0,L,S)
3 }
4 RestoreTree(i,j, W){
5   print(i,j,W);
6   if(i==j) return;
7   k =pi(i,j,W).k;
8   Wx=pi(i,j,W).x;
9   Wy=pi(i,j,W).y;
10  RestoreTree(i,k,Wx);
11  RestoreTree(k+1,j,Wx);
12 }

```

здесь строки 1-3 означают, что восстановление дерева разбора для всей строки является частным случаем восстановления дерева для подстроки и любого нетерминала. Строка 5 печатает оптимальный нетерминал для сегмента i, j . Строка 6 – если нет дочерних отрезков, то и восстанавливать нечего. Далее, строки 7-9 расшифровывают содержание $\pi(k, x, y)$. Строки 10, 11 восстанавливают поддерева.

При построении грамматики мы понимали, что тот или иной нетерминал соответствует петле или спариванию определенных нуклеотидов. Поэтому восстановленное дерево позволяет нам восстановить структуру данной последовательности. Этот алгоритм называется алгоритмом СΥК (Cocke-Yonger-Kasami). Время работы алгоритма требует заполнения трехмерной матрицы $L \times L \times M$, причем на каждом шаге надо вычислить максимум по $L \times M \times M$ параметрам. Поэтому время работы имеет порядок $T \sim L^3 M^3$.

Существует аналог алгоритма вперед-назад, позволяющий оценить вероятность того, что тот или иной нетерминал (т.е. элемент структуры) соответствует тому или иному фрагменту последовательности. Это алгоритм внутрь-наружу (inside-outside). Алгоритм устроен так же как и алгоритм вычисления статистических сумм – так же вычисляются внутренние и внешние суммы, отличие только в том, что суммирование производится еще и по всем возможным нетерминалам.

11.4 Описание структуры РНК. Ковариационные модели.

При рассмотрении белковых семейств мы определили профиль как способ описания семейств, причем в качестве основного предположения использовалось предположение о независимости позиций в последовательностях. Это позволило в качестве профиля использовать скрытые марковские модели. Для РНК предположение о независимости позиций является слишком сильным, поскольку для РНК характерны дальние взаимодействия, определяющие вторичную структуру. Поэтому скрытые марковские модели не могут описать профили структур РНК. Для описания семейств используют

ковариационные модели и соответствующие им вероятностные контекстно-свободные грамматики.

Ковариационная модель является обобщением скрытых марковских моделей и служит профилем для структурированной РНК. Эти модели включают в себя, с одной стороны, предпочтения использования букв (так же как эмиссионные вероятности в НММ) и возможности вставок и делеций, а также требования к образованию комплементарных пар. Если рассматривать эти модели, как модели генерации последовательностей, то в области петель они устроены так же, как и НММ профили (рис.11.9А,В). В области спиралей они порождают символы парами, но при этом допускают делеции и вставки рис.11.9С. На рис.11.9В *MP* означает сопоставление комплемен-

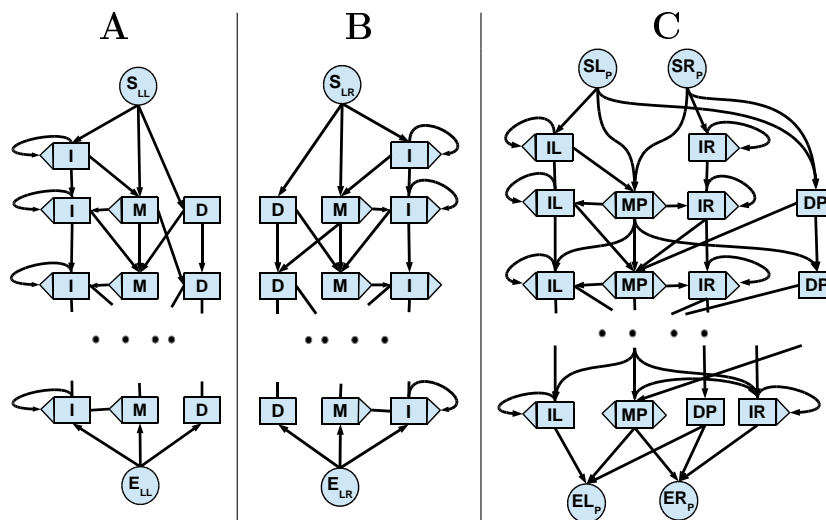


Рис. 11.9: Модели для петли (А,В) и спирали (С). Треугольнички показывают где происходит эмиссия символа – слева или справа.

тарных пар модели. *DP* – делеция пары, т.е. в представленной последовательности нет соответствующей пары. *IL*, *IR* – вставки в левый и в правый тяж спирали. Каждому узлу на схеме соответствует нетерминал, а каждой стрелке – набор правил с перечислением возможных эмиссий. Каждая спираль имеет два входа и два выхода, а каждая петля – один вход и один выход. Далее мы можем применить механизм ЛЕГО для конструкции полной модели (рис.11.10).

Для выравнивания последовательности с ковариационной моделью модель можно преобразовать в вероятностную контекстно-свободную грамматику и применить алгоритм СУК.

Основной базой данных по РНК является база данных **rfam**, RNA families. В этой базе данных собрано большое количество семейств РНК с разными функциями, есть множественные выравнивания и ковариационные модели.

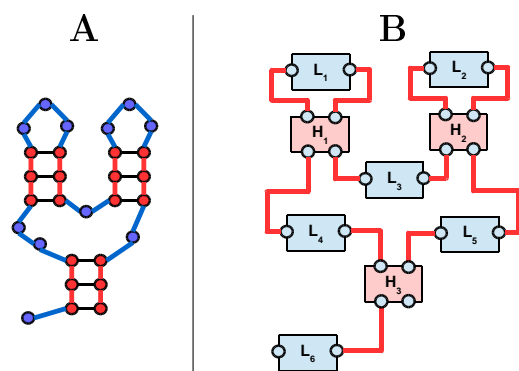


Рис. 11.10: Сборка модели из частей. А – моделируемая структура, В – Сборка модели из составных частей.