Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра компьютерных технологий

М. Э. Дворкин

Методы минимизации необходимого числа цепей для секвенирования ДНК

Дипломная работа

Научный руководитель: Г. А. Корнеев

Санкт-Петербург
2010

Содержание

Глава 1. Задача восстановления ДНК

В данной главе приведен обзор предметной области.

Наследственная информация всех живых организмов зашифрована в молекулах дезоксирибонуклеиновой кислоты (ДНК) [24], которые организованы в более сложные структуры, называющиеся хромосомы. Именно информация, хранящаяся в молекулах ДНК, передается клеткам следующего поколения при репродукции.

Совокупность всей наследственной информации о данном организме называется геномом. У организмов, являющихся эукариотами (таковыми являются животные, растения и грибы) весь геном зашифрован в молекулах ДНК.

Структурная единица наследственности живого организма называется ген.

Геном эукариота содержит как огромное число генов, так и некодирующую ДНК. Так, например, более 98% генома человека не кодирует никакие белковые последовательности. Часть ДНК и вовсе является «мусорной ДНК» согласно некоторым исследованиям [20]. Так, например, эксперименты показывают, что можно удалить 1% генома мыши, никак не повлияв не ее фенотип, — не создав каких-либо наблюдаемых изменений в организме [16].

Однако, неизвестно никаких простых правил, позволяющих отделить кодирующую и некодирующую ДНК, поэтому рассматривается именно задача полного секвенирования генома, и только после решения этой задачи возмонжо исследование значений тех или иных участков генома.

1.1. Задача секвенирования ДНК

В общем случае, задача секвенирования биополимеров (sequencing) состоит в определении их аминокислотной или нуклеотидной последовательности. Результатом секвенирования является линейное (реже — кольцевое) описание биополимера, которое содержит всю информацию, которую несет данный полимер. В сущности, итогом секвенирования становится точное знание атомной структуры данной молекулы.

В настоящей работе рассматривается задача секвенирования молекулы ДНК: задача определения линейной последовательности нуклеотидов, составляющих данную молекулу ДНК.

Техники секвенирования ДНК можно разделить на два общих типа: проход от праймера (primer walking) и метод дробовика (shotgun sequencing; также встречаются переводы: дробовое секвенирование, шотган-секвенирование). Первый метод состоит в последовательном чтении нуклеотидов генома. На текущий момент [6] наиболее длинная (имеется в виду число нуклеотидов) молекула ДНК, успешно секвенированная данным методом состоит из примерно 7000 нуклеотидов. Таким образом, данный метод подходит лишь для секвенирования ДНК примитивных организмов или для секвенирования отдельных недлинных участков ДНК, «вырезанных» из молекулы ДНК какими-либо иными способами.

Метод дробовика назван так по аналогии с дробовым ружьем, пули, вылетающие из которого, распространяются во все стороны произвольно. При применении метода дробовика из молекулы ДНК выделяются короткие участки (длиной порядка сотен нуклеотидов). Эти участки считываются при помощи механизмов, аналогичных используемым в методе прохода от праймера.

Таким образом, результатом работы метода дробовика является массив коротких последовательностей нуклеотидов, каждая из которых встречается (как подстрока) в исследуемой молекуле ДНК. Такие последовательности будем называть цепями (также используется менее

благозвучный термин «риды» от англ. reads).

Метод дробовика применяется в биологических исследованиях повсеместно: для секвенирования клонированных хромосом бактерий [17] специфичных участков генома [25], небольших геномов организмов-прокариотов [12] и больших геномов организмов-эукариотов [18].

Следует понимать, что в силу множества факторов при чтении даже коротких последовательностей нуклеотидов аппаратные погрешности могут повлечь ошибку. При этом каждая цепь может являться не подстрокой искомой последовательности нуклеотидов, а строкой, слегка отличающейся от нее. Более строго говоря, логично ожидать, что редакционное расстояние [2] от цепи, полученной аппаратным путем, до некоторой подстроки искомой строки является либо нулевым, либо — в случае ошибки чтения — небольшим.

Качественные и количественные характеристики ошибок, получаемых при чтении последовательностей нуклеотидов, будут более подробно описаны в разд. 3.2.

1.2. Парные цепи

В настоящее время одной из эффективно используемых технологий чтения последовательностей нуклеотидов является технология Solexa [15].

При применении данной технологии, молекула ДНК разбивается сфокусированными акустическими волнами на участки (длиной порядка сотен нуклеотидов). К концам полученных участков приклепляются адаптеры SBS3, к которым, в свою очередь, прикрепляются (с разных сторон) адаптеры P5 и P7.

Полученная молекула, имеющая форму спирали, расщепляется на две части. Каждая из этих частей, таким образом, маркирована с обоих концов двумя адаптерами, которые притягиваются к парным адаптерам, располагающимся на плоской прозрачной поверхности, в следствие чего каждая часть предстваляет собой «мост», основания которого закреплены.

Каждый такой «мост» многократно подвергается процедуре копирования, так что в итоге на прозрачной поверхности оказывается около 1000 мостов, каждый из которых содержит одну и ту же последовательность нуклеотидов. При этом одинаковые мосты находятся на поверхности рядом друг с другом, а разные мосты разнесены на заметное (в данных масштабах) расстояние.

На следующей стадии к основаниям каждого моста начинают прикрепляться молекулы азотистых оснований, парные соответствующим нуклеотидам, принадледащим мосту, и флюоресцентно помеченные. При освещении лазером, каждое азотистое основание отражает свет соответствующей длины волны, который детектируется снаружи прозрачной поверхности.

После этого части молекул, отвечающие за отражение света, удаляются, и новые азотистые основания прикрепляются к следующим нуклеотидом каждого моста. Таким образом, на i-й итерации данного этапа определяется нуклеотид, стоящий на i-й позиции от основания моста. При этом иные нуклеотиды данного моста не влияют на это измерение.

Данный процесс происходит одновременно с миллионами таких мостов, что позволяет считывать огромное число последовательностей нуклеотидов одновременно.

1.3. Специфика данных

В настоящей работе представлен алгоритм секвенирования ДНК по данным, получаемым аппаратом Illumina's Genome Analyzer IIe [13].

Предлагаемый в настоящей работе алгоритм применим в случае работы с данными, полученными с помощью любого другого аппарата, выдающего в качестве выходных данных парные цепи — желательно, с информацией о качестве (вероятности ошибки) чтения каждого нуклеотида. При этом будет изменяться число цепей, необходимое для восстановления генома. Правильнее говорить о необходимом

коэффициенте покрытия — среднем числе цепей, которые покрывают один нуклеотид.

Следует отметить следующие специфические для данного аппарата и для данного метода чтения данных особенности:

- Все цепи состоят из равного числа нуклеотидов. Если это не так, то можно дополнить более короткие цепи до необходимой длины нуклеотидами, помеченными как заведомо ошибочно считанные (вероятность ошибки составляет 75%). Дополнение можно проводить либо с одной стороны, либо с обеих сторон поровну — в зависимости от причин, по которым данная цепь оказалась короче остальных.
- Длина каждой цепи мала. В частности, в асимптотике времени работы алгоритма дополнительный множитель, равный средней длине цепи (например, 36), хоть и критичен, но не катастрофичен.
- Расстояние между парными цепями в исследуемой молекуле ДНК распределено согласно нормальному распределению (распределению Гаусса). Выполнения этого свойства логично ожидать, если на расстояние между цепями (в частном случае — на длину участков молекулы ДНК, остающихся после действия сфокусированных акустических волн) влияют множественные случайные факторы.
- Стандартное отклонение (разброс) нормального распределения расстояния между парными цепями (см. пред. пункт) меньше либо сравнимо по порядку с длиной одной цепи. Это свойство наиболее «хрупкое», так как к нему нет естественных оснований, и возможен эксперимент, в котором оно не выполняется.

Одной из целью создания предлагаемого алгоритма является решение задачи секвенирования генома печеночного сосальщика (Phylum Platyhelminthes), являющегося причиной описторхоза (вида гельминтоза), широко распространенного в Сибири (так как жители Сибири часто

употребляют в пищу строганину из сырой мороженой рыбы, которая, будучи недостаточно проварена, может вызвать заражение печеночным сосальщиком). Таким образом, полученные в настоящей работе результаты могут быть применены в биоинформатике, а затем и в медицине.

1.4. Обзор существующих алгоритмов восстановления ДНК

Так как современные технологии секвенирования ДНК не позволяют считывать геном какого бы то ни было организма-эукариота за один проход, рассматривают задачу сборки генома на основе фрагментов.

При этом в некоторых случаях используется наличие ранее восстановленного генома особи того же вида, что и исследуемая (reference genome). Ситуации, когда таковой геном отсутствует (либо не используется) называются сборкой генома *де-ново*. В настоящей работе рассматривается именно задача сборки де-ново на основе парных цепей.

Известные методы и идеи, лежащие в основе существующих алгоритмов сборки ДНК включают:

- Поиск эйлерова пути в графе де Бруйна — используется в системах EULER [21], Velvet [26], ABySS [8].
- Эвристический поиск гамильтонова пути — используется в системах Newbler [19], Celera [9], Arachne [7].

Предлагаемый в настоящей работе алгоритм использует в том числе и алгоритм поиска эйлерова пути, но не в графе де Бруйна, а в графе развилок (см. разд. 3.4).

Глава 2. Схема секвенирования ДНК

В данной главе описана общая схема предлагаемого алгоритма секвенирования ДНК.

Введем необходимые обозначения:

- n — число пар цепей во входных данных;
- L — длина каждой цепи (в нуклеотидах);
- μ — математическое ожидание расстояния между парными цепями (в нуклеотидах);
- σ — стандартное отклонение той же величины;
- G — искомая последовательность нуклеотидов (исследуемой молекулы ДНК);
- ϵ — доля ошибочно прочитанных нуклеотидов.

Определение 2.1. n-мерным графом Де Бруйна над алфавитом A с m символами называется [10] граф $< V, E >$, такой что

- $V = \{(u_1, \ldots, u_n) : u_1, \ldots, u_n \in A\}$, иными словами, множество V состоит из всех строк длины n над данным алфавитом A;
- $E = \{((u_1, u_2, \ldots, u_n), (v_1, v_2, \ldots, v_n)) : u_2 = v_1; \ldots; u_n = v_{n-1}\}$, иными словами, ребро соответствует приписыванию справа одного символа и удалению самого левого символа.

Пути в графе Де Бруйна соотвествуют строке над данным алфавитом, а вершины, составляющие путь, — подстрокам данной строки, имеющим длину n. Как будет показано ниже, в случае полного и безошибочного покрытия искомой строки подстроками равной длины, задача восстановления строки эквивалентна задаче поиска эйлерова пути в подграфе графа Де Бруйна [26]. Предлагаемый в данной работе алгоритм предполагает наличие ошибок в биоинформатических

данных, и не предполагает «идеального» покрытия (каждая подстрока соответствующей длины считана хотя бы один раз).

В связи с вышесказанным, введем следующее опередение.

Определение 2.2. Обобщенным n-мерным графом Де Бруйна над алфавитом с m символами будем называть граф $< V, E >$, такой что

- $V = \{(u_1, \ldots, u_k) : u_1, \ldots, u_k \in A; k \leq n\}$, иными словами, множество V состоит из всех строк длины не более n над алфавитом A;
- $E = \{((u_1, \ldots, u_k), (v_1, \ldots, v_m)) : u_k = v_{m-1}; u_{k-1} = v_{m-2}; \ldots\}$, иными словами, ребро соответствует приписыванию справа одного символа и удалению произвольного числа символов слева.

2.1. Вычислительная сложность задачи секвенирования ДНК

Когда размеры биоинформатических данных, являющихся входными данными для алгоритма, огромны (миллионы цепей, каждая из которых состоит из десятков нуклеотидов), вопрос об асимптотике времени работы алгоритма является первостепенным. Квадратичный (от размера входных данных) алгоритм, даже с чрезвычайно малой константой уже сомнителен в данных условиях — только лишь если алгоритм хорошо поддается параллелизации. Об алгоритмах с экспоненциальным временем работы не следует и помышлять.

В связи с вышесказанным, возникает вопрос о классификации решаемой задачи с точки зрения теории сложности.

Определение 2.3. Задачей о кратчайшей общей надстроке (SUPER-STRING) [22] формулируется следующим образом:
Вход. Набор строк $\{s_1, s_2, \ldots, s_n\}$ и натуральное число m.
Вопрос. Существует ли строка t, состоящая из не более чем m символов, такая что каждая строка s_i является подстрокой t?

Естественно, на практике более полезна данная задача в формулировке не «Существует ли строка...?», а «Найти кратчайшую строку, которая...». Однако именно в приведенной формулировке задача является вопросом с ответом да/нет, и потому можно говорить о ее классификации в рамках теории сложности.

Задача SUPERSTRING является NP-полной [2, 4]. Из этого следует, что не только ответ на вопрос о существовании, но и поиск само́й кратчайшей строки невозможно осуществить за полиномиальное время, если только не выполняется $P = NP$.

Обобщим данную задачу на случай парных цепей, предложив формализацию задачи, решаемой в настоящей работе.

Определение 2.4. Задачей о кратчайшей общей надстроке набора парных цепей (SUPERSTRING-PAIRS) будем называть задачу. формулирующуюся следующим образом:

Вход. Набор пар строк $\{(s_{11}, s_{12}), (s_{21}, s_{22}), \ldots, (s_{n1}, s_{n2}))\}$ над алфавитом {a, c, g, t}, натуральные числа g_1, g_2 и m.

Вопрос. Существует ли строка t, состоящая из не более чем m символов, такая что для любой пары (s_{i1}, s_{i2}) либо две строки s_{i1} и $\overleftarrow{s_{i2}}$, либо две строки $\overleftarrow{s_{i1}}$ и s_{i2} входят в t как подстроки на расстоянии между собой от g_1 до g_2 символов (включительно)?

Теорема 2.5. *Задача SUPERSTRING-PAIRS NP-полна.*

Доказательство. Докажем это, сведя задачу SUPERSTRING над алфавитом $0, 1$ к задаче SUPERSTRING-PAIRS. Пусть даны строки $\{s_1, s_2, \ldots, s_n\}$ и число m.

Построим набор пар строк следующим образом: для всех $i \in [1, n]$ будем формировать $s_{i1} = s_{i2}$ из s_i путем замены символов 0 на A, а символов 1 — на C.

Также положим $g_1 = 1$; $g_2 = 2m$; $m_2 = 2m + 1$. Здесь m_2 — это параметр уже задачи SUPERSTRING-PAIRS, а не SUPERSTRING.

Предположим, что на данный экземпляр задачи SUPERSTRING

ответ положительный, — искомая строка t существует. Рассмотрим тогда в кажестве предположительного сертификата для задачи SUPERSTRING-PAIRS строку $t_2 = tA\overleftarrow{t}$. Для всякого i, так как в строке t встречалась подстрока s_i, то в строке t_2, встречается подстрока $s_i = s_{i1}$ (в первой половине). С другой стороны, во второй половине строки t_2 встречается строка $\overleftarrow{s_i} = \overleftarrow{s_{i2}}$. Причем, эти два появления искомых строк в t_2 разделены хотя бы одним символом, и не более чем $2m$ символами, что и требуется для удовлетворения условия задачи.

Наоборот, предположим, что на поставленную задачу SUPERSTRING-PAIRS ответ положительный. Это значит, что существует строка t_2, удовлетворяющая всем условиям. Подсчитаем суммарное число символов A и C в строке t_2, назовем его n_1, суммарное число символов G и T — назовем n_2. Сумма этих двух чисел равна $|t_2|$ и не превосходит $m_2 = 2m + 1$, следовательно, меньшее из них не превосходит m.

Не умаляя общности, будем полагать, что $n_1 \leq m$. Второй случай рассматривается аналогично. Рассмотрим строку t, получающуюся из t_2 удалением всех символов G и T, и заменой всех символов A на 0 и C — на 1.

Поскольку по условию для любого i в строку t_2 как подстрока входила либо s_{i1}, либо s_{i2}, и при этом в имеющемся экземпляре задачи $s_{i1} = s_{i2}$, то в строку t входит s_i как подстрока. (Свойство «содержать данную подстроку», не пропадает в процессе удаления некоторых символов, которые в этой подстроке не встречаются).

Таким образом, полученная строка t имеет длину не более m и содержит в себе все s_i как подстроки. Следовательно, данные экземпляры задач SUPERSTRING-PAIRS и SUPERSTRING либо оба имеют положительный ответ, либо — оба отрицательный, и получено полиномиальное сведение по Карпу [4] задачи SUPERSTRING к задаче SUPERSTRING-PAIRS. Из этого следует, что задача SUPERSTRING-PAIRS — NP-полна. □

2.2. Общая схема алгоритма

Одной из ключевых структур, использующихся в предлагаемом алгоритме, является *граф возможных продолжений*, который будем обозначать G_c.

В ходе работы алгоритма граф G_c изменяется. Опишем начальную конструкцию этого графа. Это в некотором смысле обобщение конструкции несжатого суффиксного дерева [2] на случай работы с несколькими строками и их обратно-комплементарными.

Вершинам графа G_c соответствуют все подстроки всех цепей, имеющихся во входных данных, а также все подстроки строк, обратно-комплементарных цепям, встречающимся во входных данных. При этом если некоторая подстрока встречается несколько раз (в разных цепях или даже в одной), то такой подстроке соответствует только одна вершина графа G_c.

Ребра графа G_c ведут от вершины, соответствующей строке $(u_1, u_2, \ldots, u_n)$, к вершине, соответствующей строке $(u_1, u_2, \ldots, u_n{+}1)$, если такая вершина в графе присутствует. Таким образом, ребро соответствует приписыванию справа одного символа.

Граф G_c является подграфом обобщенного графа Де Бруйна, описанного в разд. 2.2.

Кроме ребер в данном графе будем рассматривать также суффиксные ссылки (по аналогии к суффиксными ссылками в суффиксных деревьях [2]). Суффиксная ссылка ведет от вершины, соответствующей строке $(u_1, u_2, \ldots, u_n)$, к вершине, соответствующей строке $(u_2, u_3, \ldots, u_n)$. Иными словами, суффиксная ссылка обозначает удаление одного символа из начала строки. Отметим, что (по построению) суффиксные ссылки ведут в существующие в графе G_c вершины из всех вершин, кроме той, что соответствует пустому слову. Будем говорить, что из этой вершины суффиксная ссылка не ведет никуда.

Алгоритм построения графа возможных продолжений G_c будет

описан в разд. 3.1.

В полученном графе методом динамического программирования вычисляются индуктивные функции, описанные в разд. 3.1.1, что позволяет обнаруживать ребра, соответствующие нуклеотидам которые (предположительно) были считаны ошибочно. Методы обнаружения таких ситуаций будут описаны в разд. 3.2.

Следующим этапом является выбор произвольных цепей с достаточно высоким качеством чтения составляющих их нуклеотидов и наращивание этих цепей влево и вправо по одному символу, с выполнением условия, что каждый наращиваемый символ является единственным возможным (разумным) продолжением строки в соответствующую сторону. Методы наращивания подстроки будут описаны в разд. 3.3.

Наращивание подстроки может быть остановлено либо в случае достижения начала или конца искомой последовательности G, либо в случае обнаружения существенной «развилки». Обеим этим ситуациям ставится в соответствие вершина в конструируемом *графе развилок* G_b. Строки же, полученные в результате алгоритма наращивания подстрок, ставятся в соответствие ребрам этого графа. Процесс построения графа развилок будет описан в разд. 3.3.2.

Ребрам этого графа назначаются вещественные веса; в разд. 3.4.1 будет описан алгоритм создания мультиграфа (графа с разрешенными кратными ребрами) на основании данных вещественных весов. В полученном графе искомая последовательность нуклеотидов G находится алгоритмом нахождения эйлерова пути.

В итоге, схема предлагаемого алгоритма секвенирования ДНК выглядит следующим образом:

- Построение графа возможных продолжений G_c;
- вычисление весов ребер и дополнительной информации, хранимой в вершинах и ребрах графа G_c методом динамического программирования на дереве;

- обнаружение ошибочно считанных нуклеотидов во входных данных и их исправление;
- наращивание подстрок до достижения развилок и тупиков;
- построение графа развилок G_b и создание в нем кратных ребер;
- нахождение эйлерова пути в полученном мультиграфе.

Глава 3. Этапы алгоритма секвенирования ДНК

В данной главе описаны этапы предлагаемого алгоритма секвенирования ДНК, в «хронологическом порядке» — в порядке их выполнения.

3.1. Построение частей графа G_c в оперативной памяти

Первым этапом предлагаемого алгоритма является построение графа возможных продолжений G_c. В связи с (ожидаемым) огромным размером данного графа, остро встает вопрос о возможном разбиении процесса построения данного графа, (а также о возможном разбиении самого графа), на части.

Оценим число вершин в графе возможных продолжений. Для начала рассмотрим случай отсутствия ошибок в биоинформатических данных.

Тогда для каждой цепи верно следующее: либо она, либо строка, обратно-комплементарная к ней, является подстрокой искомой последовательности нуклеотидов ДНК.

У искомой последовательности нуклеотидов G всего $|G| - L + 1$ подстрок длины L, $|G| - L + 2$ подстрок длины $L - 1$, ..., $|G|$ подстрок длины один. Следовательно, суммарно, если предположить что $L \ll |G|$, получается почти ровно $L|G|$ подстрок. Если же добавить к ним их обратно-комплементарные, то всего получается $2L|G|$ строк. Все вершины графа G_c соответствуют некоторым из этих строк. Таким образом, число вершин в графе G_c не превышает $2L|G|$.

Эта оценка является оценкой сверху по той причине, что некоторые подстроки совпадают — одна и та же строка может встречаться в G в нескольких различных позициях. (например, различных подстрок длины

один на самом деле не $2|G|$, а четыре — {a, c, g, t}. Точнее, формально говоря, не более четырех).

Теперь перейдем к случаю наличия ошибок в биоинформатических данных. Будем использовать предположение, что ошибочное прочтение нуклеотида приводит к любой из трех других (ошибочных) букв с вероятностью 1/3. (Этот факт подтверждается в экспериментальных данных, см. разд. 1.3).

Допустим, что (1) все ошибочно считанные нуклеотиды пришлись на различные цепи, и (2) каждый ошибочно считанный нуклеотид породил не встречающийся в G префикс. Таким образом, каждая цепь $\alpha B\gamma$ (где B — ошибочно считанный нуклеотид) удовлетворяет тому свойству, что αB не встречается в G как подстрока. Тогда рассматриваемая ошибка породит в графе G_c целую серию вершин, соответствующих строке γ — будет добавлено $|\gamma| + 1$ дополнительных вершин. Математическое ожидание величины $|\gamma|$ составляет $(|L| - 1)/2$ (минимальное значение — ноль, максимальное — $(|L| - 1)$; распределение считаем равномерным).

Таким образом, всего получается $2n\epsilon|L|/2 = \epsilon n|L|$ дополнительных вершин, индуцированных ошибочно считанными нуклеотидами. И всего в графе G_c имеем $(2|G| + \epsilon n)|L|$ вершин.

Условие (2) может не выполняться, если (нестрого говоря) ошибочно считанный символ располагается где-то в начале цепи. Вероятность того, что строка αB встречается в G как подстрока, вычисляется следующим образом: вероятность ее появления в какой-либо конкретной позиции в G составляет $4^{-(|\alpha+1|)}$, вероятность «непоявления», соответственно, $1 - 4^{-(|\alpha+1|)}$, а вероятность «непоявления» нигде в строке G в таком случае $(1 - 4^{-(|\alpha+1|)})^{|G|-|\alpha|}$. Рассматривая только первые два члена ряда Тейлора для указанной величины, можно оценить вероятность появления подстроки αB в G как $4^{-(|\alpha+1|)}|G|$.

Следовательно, если $|\alpha| > log_4|G|$, то вероятность наличия αB в G мала. Математическое ожидание величины $|\alpha|$ составляет $(|L| - 1)/2$

(аналогично величине $|\gamma|$, см. выше), поэтому в случае $|L| > 2log_4|G|$ будет верно следующее утверждение: для более чем половины цепей с ошибочно считанными нуклеотидами вероятность появления строки αB в качестве подстроки $|G|$ меньше $1/2$. Иными словами, более половины цепей с ошибоными нуклеотидами порождают новые префиксы, и условие (2) в большинстве случаев выполняется.

Перейдем к рассмотрению условия (1). Оно не выполняется в тех ситуациях, когда одна цепь содержит не менее двух ошибочно считанных нуклеотидов. Сначала вычислим вероятность каждой цепи обладать этим свойством. Она равняется $1 - (1 - \epsilon)^{|L|} - |L|(1 - \epsilon)^{|L|-1}\epsilon$ (из единицы вычитается вероятность не содержать ошибок и вероятность содержать ровно одну ошибку). Считая ϵ малым и оставляя только старший ненулевой член ряда Тейлора по ϵ (в данном случае — второй), получим вероятность $|L|(|L| - 1)\epsilon^2$.

Однако, что более важно, отметим, что всякое «попадание» двух (или более) ошибочно считанных нуклеотидов на одну и ту же цепь уменьшает размер получающегося графа G_c (относительно ситуации, если «при прочих равных» ошибочно считанные риды «попадут» в различные цепи). И для оценки сверху числа вершин в графе G_c условие (1) можно считать выполненным — в случае его невыполнения, оценка на число вершин окажется тем более верной.

3.1.1. Весовая функция и дополнительная информация в графе G_c

В графе G_c каждому ребру e аттрибутируется весовая функция w_e и дополнительная информация — набор пар цепей p_e, в которых встречается строка, соответствующая вершине, являющейся концом ребра e. При том, если эта строка встречается в некоторой паре цепей более одного раза, то такая пара цепей включается в p_e соответствующее число раз. (Такая ситуация случается почти исключительно только с весьма короткими строками — длины $log_4|L|$ и меньше).

Весом w_e ребра e назовем сумму по всем цепям, содержащим подстроку, соответствующую конечной вершине ребра e, вероятностей того, что все нуклеотиды данной подстроки считаны корректно. Так, если во входных данных имеется цепь $\alpha\beta C\delta$, то в графе G_c (по определению) есть ребро, которое ведет из вершины, соответствующей строке β в вершину, соответствующую строке βC. К весу этого ребра прибавляется вероятность того, что все нуклеотиды в данном вхождении строки βC в соответствующую цепь были считаны безошибочно. Вычисляется эта вероятность как произведение вероятностей каждого нуклеотида быть считанным правильно, что известно непосредственно из характеристики качества данного нуклеотида во входных данных.

Определение 3.1. В дальнейшем будем говорить о *весе вершины* в графе G_c как о весе (единственного) ребра, ведущего в данную вершину. Данное определение неприменимо к корню графа G_c — будем говорить, что вес корня неопределен. (Действительно, вес вершины, нестрого говоря, характеризует частоту ее появления в искомой последовательноси, а говорить о появлении пустой строки как подстроки искомой последовательности — бессмысленно).

Оценим для произвольной подстроки α искомой последовательности нуклеотидов математическое ожидание веса вершины, ей соответствующей. Будем полагать, что в G строка α (либо $\overleftarrow{\alpha}$) встречается один раз. Если она встречается $k > 1$ раз, то математическое ожидание веса соответствующей вершины окажется в k раз больше.

Рассмотрим произвольную цепь β. Вероятность того, что она соответствует участку G, целиком содержащему α равна $\frac{L-|\alpha|+1}{|G|-L+1}$ (отношение числа позиций β, покрывающих вхождение α в G, к числу всех возможных позиций). Вероятность того, что β соответствует такому участку, и все нуклеотиды, образующие α, были считаны правильно, составляет $\frac{L-|\alpha|+1}{|G|-L+1} \cdot (1-\epsilon)^{|\alpha|}$. Наконец, математическое ожидание числа цепей, которые содержат правильно считанную подстроку α составляет

$w_0(|\alpha|) = \frac{L-|\alpha|+1}{|G|-L+1} \cdot (1-\epsilon)^{|\alpha|} \cdot 2n$.

Эту величину будем называть обозначать $w_0(|\alpha|)$ (в условиях конкретных входных данных эта величина является функцией от длины строки α) и называть *ожидаемым весом ребра*.

3.1.2. Параллелизация построения графа G_c

Граф G_c, в котором, как было показано выше, порядка $(2|G|+\epsilon n)|L|$ вершин, не может (для разумных размеров экспериментальных данных) быть записан в оперативную память современного компьютера (даже если речь идет лишь о хранении структуры графа, но не о хранении дополнительной информации на его ребрах).

Возникает естественный вопрос о возможности разбить граф G_c на части такого размера, что каждая из них по отдельности умещается в оперативную память. Тогда для каждой из них можно будет независимо запустить алгоритмы обхода графа, и вычислить индуктивные функции, описанные в разд. 3.2.

Зафиксируем максимальный объем оперативной памяти, который может быть выделен для хранения части графа G_c. Будем для простоты говорить, что данное ограничение сверху установлено не на размер памяти (в байтах), а на максимальное количество вершин V_{MAX}, которое может иметь часть графа G_c, хранимая в оперативной памяти. (Поскольку рассматриваемые части графа G_c суть деревья, число ребер на единицу отличается от числа вершин. Поэтому вычислить максимальное число вершин, которое помещается в оперативную память, вычислить не составляет труда).

Воспользуемся древовидной структурой графа G_c. Предположим, что некоторое его поддерево H не вмещается в указанный лимит на число вершин V_{MAX}. Поддерево H состоит из корня и (не более чем) четырех его поддеревьев.

Логично предположить, что такая ситация имеет место в «верхних»

слоях графа G_c — на расстоянии менее $\log_4 |G|$ от корня, и следовательно, логично предположить, что четыре поддерева имеют примерно поровну вершин (иначе получится, что некоторые короткие строки встречаются в искомой последовательности чаще других). Приведенные соображения являются неформальными предпосылками к следующему алгоритму.

Алгоритм 3.2. Процедура построения части графа G_c, являющейся поддеревом вершины, соответствующей строке α. Просматриваем по порядку все цепи из входных данных. Для каждой цепи при помощи алгоритма Кнута-Морриса-Пратта [3] проверям, является ли α ее подстрокой, либо подстрокой строки, обратно-комплементарной к этой цепи. Если не является, то переходим к следующей цепи. Если является (введем обозначения: пусть цепь или строка, обратно-комплементарная к ней, была составлена как $\beta\alpha\gamma$), то добавляем в конструироемое суффиксное дерево последовательность вершин, соответствующих строке $\alpha\gamma$. Если в результате этого добавления (или в его процессе) число вершин дерева превысило предварительно зафиксированный лимит V_{MAX}, то (понимаем, что часть графа G_c для подстроки α не вмещается в оперативную память) стираем из памяти построенное дерево и запускаем ту же самую процедуру рекурсивно (последовательно) для всех строк αx, где $x \in \{a, c, g, t\}$. Если все входные данные обработаны, и лимит не превышен, то построенное дерево записывается на жесткий диск (в порядке обхода: сначала корень, затем — рекурсивно — его поддеревья; это соответствует лексикографическому порядку строк).

Оценим время работы представленного алгоритма. Для оценки времени работы применим амортизационный анализ [3]. Разобьем все акты чтения входных данных на два типа: осмысленные (в результате которых часть дерева была успешно построена и записана на диск) и бессмысленные (когда результатом чтения стало превышение заранее зафиксированного лимита V_{MAX}).

Рассмотрим некоторый суффикс β некоторой цепи или

строки, обратно-комплементарной к ней, $\alpha\beta$, который в итоге будет соответствовать вершине — листу построенного суффиксного дерева. Он был добавлен в записанную на жесткий диск часть построенного графа G_c в алгоритме 3.2 при просмотре всех цепей и поиске в них подстроки α. Это значит, что данная цепь была также считана при просмотре префиксов строки α длины $0, 1, 2, \ldots, |\alpha| - 1$.

Поскольку в алгоритме 3.2 для поиска подстроки в строке применяется алгоритм Кнута-Морриса-Пратта, использующий (при фиксированной искомой строке) линейное относительно строки, в которой производится поиск, время работы, то каждый просмотр входных данных занимает линейное относительно их размера время: $O(n|L|)$.

Теперь оценим число частей, которые порождает алгоритм 3.2. К сожалению, оценить размер каждой части снизу невозможно: рекурсивный запуск описываемой процедуры может привести к тому, что будет названо отдельной частью даже дерево из одной вершины.

Однако, можно воспользоваться иным соображением: для каждой части дерева G_1 рассмотрим G_2 — наибольшую (по числу вершин) из четырех частей, рассмотренных вместе с G_1 в соответствующей рекурсивной процедуре. (При этом G_2 может совпадать с G_1). Так как на предыдущем уровне рекурсии рассматриваемая часть не вместилась в оперативную память, то число вершин в ней больше V_{MAX}, и следовательно, число вершин в G_2 не меньше $V_{MAX}/4$. (Действительно, если в наибольшей из четырех частей вершин меньше, чем $V_{MAX}/4$, то суммарно в четырех частях вершин меньше, чем V_{MAX}, а с учетом корневой вершины рассматриваемого поддерева — не больше чем V_{MAX}, что ведет к противоречию.)

Итак, у каждой части дерева G_1 есть «старший брат» G_2 достаточно большого размера. И обратно, каждое поддерево может являться «старшим братом» не более чем четырем частям дерева.

Рассмотрим произвольный слой графа G_c (множество вершин,

соответствующее строкам одинаковой длины). Рассмотрим все поддеревья с корнями в вершинах этого слоя. Сумма вершин в них не превышает размера графа G_c (будем обозначать его $|G_c|$). Те из них, которые являются чьими-то «старшими братьями» имеют не менее $V_{MAX}/4$ вершин, следовательно, их не более $|G_c|/(V_{MAX}/4) = 4|G_c|/V_{MAX}$.

Всего в графе $L+1$ слоев, при этом вершина нулевого слоя (слоя, состоящего только из корня дерева) не является ничьим «старшим братом», поэтому суммируя ограничения по всем слоям, получаем, что общее число «старших братьев» не превышает $4L|G_c|/V_{MAX}$. Наконец, так как не более четырех частей, загружаемых в оперативную память, могут иметь «общего старшего брата», то общее число получаемых в алгоритме 3.2 частей не превышает $16L|G_c|/V_{MAX}$.

3.2. Обнаружение и исправление ошибок в данных

Применим метод динамического программирования [1] для вычисления некоторых индуктивных функций [5], определенных на вершинах графа G_c.

Определение 3.3. Для каждой вершины v найдем $c(v)$ — «наиболее вероятное продолжение» префикса, соответствующего вершине v. Можно также назвать эту функцию наиболее весомым продолжением.

Если у вершины v нет детей, то определим $c(v)$ как ε (пустую строку). Если же из вершины v ведет хотя бы одно ребро, то среди всех исходящих ребер выберем имеющее наибольший вес. Пусть это ребро помечено символом B (ведет из вершины, соответствующей строке α в вершину, соответствующую строке αB), а его концом является вершина u. Определим $c(v)$ как конкатенацию B и $c(u)$. Таким образом, наиболее вероятное продолжение есть последовательность символов, которая получается, если идти от данной вершины каждый раз по исходящему ребру с наибольшим весом.

Так как значение функции $c(v)$ зависит лишь от части графа G_c, являющейся поддеревом с корнем v, то вычисление функции $c(v)$ можно проводить независимо для разных частей графа G_c, полученных в результате выполнения алгоритма 3.2.

Рассмотрим одиночную ошибку во входных данных — цепь $\alpha B\gamma$, такую что в искомой последовательности нуклеотидов не присутствует подстрока $\alpha B\gamma$ (равно как и $\overleftarrow{\alpha B\gamma}$), но присутствует подстрока $\alpha B'\gamma$ (либо $\overleftarrow{\alpha B'\gamma}$) для некоторого нуклеотида B'. Если строки αB и $\overleftarrow{\alpha B}$ также не присутствует как подстроки в G, то имеет место следующая ситуация: у вершины, соответствующей подстроке αB, наиболее вероятным продолжением является строка γ.

В то же время у вершины, соответствующей строке $\alpha B'$, в ее поддереве присутствует либо строка γ, либо хотя бы некоторый ее префикс — по той причине, что строка γ является продолжением строки $\alpha B'$ в G (или в $\overleftarrow{G}$), и разумно ожидать покрытия соответствующей позиции G хотя бы одной цепью из входных данных).

3.3. Правила наращивания подстрок

Основная вычислительная задача в предлагаемом алгоритме секвенирования ДНК — обход графа G_c в правильном порядке — таком, чтобы в результате обхода получались строки, принадлежащие искомой последовательности нуклеотидов G (или $\overleftarrow{G}$), причем имеющие как можно большую длину.

Для этого в предлагаемом алгоритме отведен этап, называющийся *наращивание подстрок*. Общая схема его такова: в каждый момент имеется текущая строка s, о которой можно с уверенностью сказать, что она принадлежит G или $\overleftarrow{G}$. Затем, в соответствии с информацией, хранящейся в графе G_c, данная строка наращивается вправо и влево настолько, насколько это возможно, — либо до ситуации, когда выбор между двумя нуклеотидами сделать не представляется возможным (до развилки), либо

до края искомой последовательности нуклеотидов.

В качестве начального состояния для алгоритма наращивания подстрок выбирается любая цепь из входных данных, такая что все ее нуклеотиды имеют максимальный показатель качества. После того, как в ходе работы алгоритма начальная строка с обеих сторон будет достроена до развилок или до концов G, тот же алгоритм запускается для другой начальной строки — еще одной цепи с максимальным качеством. В разд. 3.4 будет показано, как объединять полученные в результате работы данного этапа результаты.

Далее будет описан алгоритм наращивания имеющейся строки вправо. После окончания его работы из строки s получается строка st. Теперь данную строку следует нарастить влево. Заметим, однако, что граф G_c «существенно несимметричен» — в нем хранятся для всех подстрок входных данных их возможные продолжения вправо. Однако, изящный выход позволяет найти специфика задачи: ведь продолжение вправо строки α есть не что иное как продолжение влево строки $\overleftarrow{\alpha}$. Следовательно, для продолжения влево строки st следует положить в качестве текущей рассматриваемой строки строку $\overleftarrow{st} = \overleftarrow{t}\,\overleftarrow{s}$, и применить к ней тот же самый алгоритм наращивания вправо. Результатом этого станет некоторая строка $\overleftarrow{t}\,\overleftarrow{s}\,p$, поэтому результатом наращивания в обе стороны строки s является $\overleftarrow{p}\,st$. (Впрочем, последнего применения функции $\overleftarrow{\ }$ можно было не проводить — алгоритм, описанный в разд. 3.4, не изменяет результат своей работы, если вместо строки α ему предложить на вход строку $\overleftarrow{\alpha}$)

3.3.1. Текущее состояние при наращивании подстрок

Опишем текущее вычислительное состояние алгоритма наращивания подстроки вправо.

Имеется строка st, где t — строка, соответствующей некоторой (текущей) вершине v графа G_c. (В начальный момент $s = \varepsilon$, а t — это некоторая цепь из входных данных).

Кроме того, для улучшения оценки возможных продолжений, хранятся цепи, парные к цепям, присутствующим в конце строки st. При этом, если в процессе построения строки st (то есть в процессе наращивания некоторой строки вправо) очередной ее символ был получен на основе имевшихся во входных данных цепей $\alpha_1, \alpha_2, \ldots, \alpha_k$, то в текущее состояние также входят парные к ним цепи $\overleftarrow{\beta_1}, \overleftarrow{\beta_2}, \ldots, \overleftarrow{\beta_k}$, вместе с информацией о том, на каких позициях их следует ожидать. (Зная μ — математическое ожидание расстояния между парными цепями, и позицию строки α_i в строке st, математическое ожидание позиции $\overleftarrow{\beta_i}$ есть сумма этих двух величин. Речь, естественно, идет о случае, когда $\overleftarrow{\beta_i}$ находится в G правее, чем α_i, в противном случае, запоминать $\overleftarrow{beta_i}$ и вовсе необязательно.)

В каждый момент хранятся только те цепи, которые еще могут быть уложены в продолжение строки st, а не хранятся цепи:

1. (критерий 1) которые уже попали в строку st, причем уместились в ней целиком (без «торчащего» вправо суффикса);
2. (критерий 2) для которых математическое ожидание из позиции в st «намного левее» конце st.

Говоря «намного левее», имеется в виду ситуация, когда разность между концом строки st и математическим ожиданием позиции цепи превышает $K\sigma$, где σ — стандартное отклонение (см. главу 2), а K — эмпирический коэффициент.

Разумным представляется выбрать коэффициент K, удовлетворяющий такому свойству: если x — случайная величина, имеющая нормальное распределение с параметрами $\mu = 0$, $\sigma = 1$, то вероятность $p(x > K) < 1/n$. В этом случае, вероятность того, что согласно критерию (2) будет удалена из рассмотрения цепь, которая на самом деле встретится правее конца строки st, крайне мала.

3.3.2. Правила выбора среди нескольких ребер со значительными весами

Итак, пусть имеется текущая строка st и множество цепей, которые ожидаются правее конца st вместе с математическими ожиданиями их позиций.

Рассмотрим ребра, исходящие из вершины, соответствующей строке t, в графе G_c. В разд. 3.1 было показазано, что математическое ожидание веса ребра, если оно соответствует действительно присутствующему в G (или в $\overleftarrow{G}$) переходу составляет $w_0(|t|)$. Ребра, веса которых много меньше этой величины, будем называть *незначительными* (формально, отсутствующие ребра также можно называть незначительными), а ребра, веса которых сравнимы с этой величиной или больше ее, — *значительными*.

Возникают три существенных случая.

Первый случай — все ребра, исходящие из вершины, соответствующей строке t, незначительные. Это означает, что на основании только цепей из входных данных, имеющих вид $\alpha t \beta$ нельзя продолжить строку st: таких цепей недостаточное число, либо в них содержится слишком большое число ошибочно считанных нуклеотидов. Выход из данной ситуации один — следует перенести первый символ строки t в конец строки s. Иными словами, следует перейти в вершину более высокого слоя в графе G_c, не меняя при этом текущее вычислительное состояние (строка st при данном переходе не изменяется, равно как и множество ожидаемых справа цепей). Отметим также, что совершаемый переход в графе G_c является проходом по суффиксной ссылке [23].

Второй случай (наиболее благоприятный) — наличие ровно одного исходящего значительного ребра из вершины, соответствующей строке t. Пусть это ребро соответствует символу B. Это означает, что имеющиеся входные данные дают основание однозначно продлить строку st до строки stB. В таком случае алгоритм действий таков: текущая рассматриваемая

строка становится stB, в графе совершается переход в вершину, соответствующую строке tB и происходят описанные ниже изменения в хранимом множестве ожидаемых цепей.

Во-первых, некоторые цепи следует удалить из множества в соответствии с критериями (1) и (2) (см. разд. 3.3.1). Так, если после добавления символа B к строке st некоторая цепь стала теперь целиком умещаться в нее (некоторая цепь является суффиксом строки stB), то эту цепь следует перестать хранить в соответствии с критерием (1). Действительно, обе цепи соответствующей пары уже нашли свое место в наращиваемой строке, и помочь в выборе будущих продолжений они уже не смогут. Также и цепи, которые не являются подстроками st, но математическое ожидание позиции которых стало слишком далеко слева от конца st, согласно критерию (2) также должны быть исключены (так как бессмысленно ожидать, что они появятся и помогут сделать выбор в пользу того или иного символа).

Во-вторых, вместе с ребром, которое ведет из вершины, соответствующей строке t, в вершину, соответствующую строке tB, хранятся все пары цепей, которые дали информацию о существовании такого перехода (см. разд. 3.1.1). Рассмотрим любую пару $(a, \overleftarrow{b})$. Пусть a — та из двух цепей, которая содержит строку t. Теперь определим, где располагается строка $\overleftarrow{b}$ — левее или правее строки a.

Если строка st достаточно длинна, а именно содержит более чем $\mu + K\sigma$ символов, то можно проверить, встречается ли строка $\overleftarrow{b}$ левее a в строке st — например, проверить все позиции от $\mu - K\sigma$ до $\mu + K\sigma$ символов влево от позиции, в которой находится a. Однако можно воспользоваться более быстродейственными методами, описанными в разд. 3.3.3.

Если строка st коротка, то строка $\overleftarrow{b}$ может располагаться как слева от начала st (и, в частности, слева от a), так и справа от a. Нет каких-либо доводов, позволяющих предпочесть один из вариантов другому, а потому следует добавить $\overleftarrow{b}$ в множество ожидаемых цепей, но пометить ее как

«ожидаемую наполовину» — как цепь, вероятность появления которой в будущем вдвое меньше.

Наконец, третий случай — из вершины соответствующей строке t выходит более одного значительного ребра. Это означает, что на основании одного только окончания t строки st невозможно сделать вывод о том, какой символ следует выбрать для наращивания строки. Здесь в ход вступают хранимые в текущем вычислительном состоянии ожидаемые цепи.

Пусть символы, соответствующие значительным ребрам, ведущим из вершины, соответствующей строке t, — $B_1, B_2, \ldots, B_k$ (таковых от двух до четырех). Пусть вершины, в которые ведут значительные ребра — соответственно $v_1, v_2, \ldots, v_k$.

Для оценки того, насколько каждый из этих символов подходит как продолжение st, используются следующие критерии:

1. Переберем пары цепей, хранимые в графе G_c как дополнительная информация к ребру, ведущему в v_i. Рассмотрим одну из этих пар $(a, \overleftarrow{b})$, пусть a — та цепь, которая содержит строку tB_i, а $\overleftarrow{b}$ — парная ей цепь. Если строка st достаточно длинная, и $\overleftarrow{b}$ содержится в ней левее чем a на число символов от $\mu - K\sigma$ до $\mu + K\sigma$, то символ B_i точно может быть продолжением st — пара $(a, \overleftarrow{b})$ является тому доказательством. Если $\overleftarrow{b}$ не встречается левее a, то можно предположить, что $\overleftarrow{b}$ встретится правее. Сделав это предположение, приложим $\overleftarrow{b}$ к уже известным ожидаемым цепям, используя методы, описанные в разд. 3.3.3. Если $\overleftarrow{b}$ «хорошо укладывается» с остальными цепями, то это весомый аргумент в пользу того, что B_i может являться продолжением строки st.
2. Среди хранимого множества ожидаемых цепей есть располагающиеся на таких позициях, что их префикс совпадает с некоторым суффиксом строки st, а их суффикс «выпирает» вправо. У таких цепей рассматривается первая буква «выпирающего

суффикса». Если размер пересекающегося участка такой цепи и строки st достаточно велик (превышает $log_4|G|$), а нуклеотид, являющийся первым символом суффикса, считан правильно, то этот символ точно может быть продолжением st.

Если по итогам приведенных критериев остается более двух возможных продолжений строки st, то из этого можно сделать вывод, что алгоритм наращивания подстроки дошел до такой ситуации, когда имеющиеся входные данные существенно не позволяют определить следующий символ. Отсюда следует, что (с достаточной степенью достоверности) в искомой последовательности нуклеотидов G присутствует подстроки stB_i и stB_j.

Такую ситуацию будем называть *развилкой*, соответствующей строке st. В случае попадания в такую ситуацию, алгоритм наращивания подстроки вправо заканчивает свою работу, и полученная строка наращивается влево. (Точнее говоря, она заменяется обратно-комплиментарной к ней и наращивается вправо).

Соответствующая этой развилке вершина добавляется в *граф развилок* G_b. Строке ust, полученной после наращивания в обе стороны, и с обеих сторон «упершейся» в развилки, ставится в соответствие ребро в графе развилок G_b. Если с одной или двух сторон алгоритм наращивания «уперся» в отстутствие символа, который может быть использован для наращивание (что предположительно соответствует началу или концу искомой последовательности нуклеотидов G), то соответствующая вершина в графе G_b также создается, но помечается не как развилка, а как тупик.

В качестве дополнительной информации к ребру графа развилок записывается строка ust, а также сумма весов набора цепей, которые целиком уместились в строку ust (эта информация вычисляется в ходе работы алгоритма наращивания подстроки). Дальнейшие методы работы с графом G_b описаны в разд. 3.4.

3.3.3. Методы сопоставления парных цепей

Рассмотрим задачу о сопоставлении ожидаемых цепей, поставленную в разд. 3.3.2.

Имеется набор цепей S, ожидаемых справа от текущей строки, и про каждую из них известно математическое ожидание позиции, в которой она должна встретиться как подстрока.

Рассматривается возможность приписывания к текущей строке некоторого символа. Это приведет к тому, что некоторое множество цепей T также попадет в множетсво ожидаемых цепей, и их надо будет «уложить» вместе с S.

Идеальная ситуация такова: для каждой ожидаемой цепи α и из множества S, и из множества T существует позиция p_α, так что выполняется следующее соотношение: для любых двух цепей α и β, если сдвинуть начало цепи β на $p_\beta - p_\alpha$ символов относительно начала цепи α, то «перекрывающиеся» их части совпадают (либо отличаются только в нуклеотидах, ошибочно считанных в хотя бы одной из этих двух цепей).

Формально говоря, для того, чтобы удовлетворить приведенное выше условие, можно присвоить всем цепям значения p на достаточно большом расстоянии друг от друга, так чтобы расположенные соответствующим образом цепи и вовсе не перекрывались. Во избежание таких ситуаций, введем дополнительное условие, что полученные значения p должны с достаточной достоверностью укладываться в нормальные распределения ожидаемых позиций для соответствующих цепей.

Итак, задача состоит в нахождении взаимного расположения нескольких цепей, обеспечивающего наибольшие области «перекрытия». При этом речь идет только о цепях, а не о строчках, обратно-комплиментарных к ним.

Для каждой цепи α построим суффиксное дерево S_α. Пользуясь алгоритмом Укконена [23], это можно сделать за время $O(L)$. Затем будем добавлять эти деревья по очереди в (изначально пустое) дерево S_*.

Если в какой-то момент некоторая вершина из добавляемого дерева уже присутствует в графе S_*, следовательно, была найдена общая подстрока двух (или более) рассматриваемых цепей.

Будем строить (верхнетреугольную) матрицу M, в которой для каждый двух цепей α и β будем хранить длину их наибольшей общей подстроки, а также относительный сдвиг (начала β относительно начала α) при котором достигается «перекрытие» по этой наибольшей общей подстроке. В процессе добавления суффиксных деревьев для отдельных строк в единое дерево построить такую матрицу не составляет труда.

Упорядочим значения матрицы M по убыванию длины. Иными словами, будем рассматривать пары цепей в порядке уменьшения длины их наибольшей общей подстроки, пусть очередная пара — (α, β). Составим строку γ как своеобразное объединение строки α и строки β, сдвинутой на соответствующее число символов (эта величина записана в матрице M). В тех позициях, где (после сдвига) одна строка имеет символ, а другая не имеет, в γ запишем единственный имеющийся символ. В тех же позициях, где обе строки имеют символ, если эти два символа совпадают, то в γ записываем именно такой символ, если же они различаются, то записываем тот из них, который имеет более высокую характеристику качества.

Для полученной строки γ с помощью алгоритма Укконена построим суффиксное дерево S_γ и добавим его в (на этот раз уже непустое) дерево S_*. Расширим матрицу M на еще один столбец и одну строчку, соответствующую строке γ, и в процессе добавления дерева S_γ в дерево S_* будем вычислять значения матрицы M, затрагивающие строку γ.

После этого пометим строки α и β, а также столбцы и строчки матрицы M, им соответствующие, как удаленные из рассмотрения — вместо них теперь рассматривается строка γ, в определенном смысле объединяющая их.

Этот процесс продолжается до тех пор, пока либо не останется в множестве рассматриваемых одна строка (все строки успешно «уложены»

вместе), либо очередное (в порядке убывания) значение из матрицы M не будет столь малым, что объединение соответствующих строк не имеет смысла.

3.4. Эйлеров обход графа развилок со взешенными ребрами

В результате нахождения развилок в процессе наращивания подстрок (см. разд. 3.3) строится граф развилок G_b. Вершины графа соответствуют ситуациям, в которых алгоритм наращивания подстроки остановился в связи с невозможностью принятия решения о следующем символе. Также некоторые вершины соответствуют не развилкам, а тупикам. Ребра соответствуют строкам, полученным в результате наращивания.

В каждой вершине, являющейся развилкой, а не тупиком, хранятся веса исходящих ребер (перенесенные из соответствующих ребер графа G_c). Также каждое ребро имеет вес, равный суммарному весу цепей, попавших в соответствующую этому ребру строку.

Эти веса позволяют судить о частоте, с которой встречаются строки, соответствующие ребрам, в искомой последовательности нуклеотидов. Действительно, если из одной вершины исходят два ребра, и веса соответствующих им ребер в графе G_c относятся приблизительно как $p : q$, и веса этих ребер относятся как $p : q$, то логично предположить, что в искомой последовательности соответствующие подстроки встречаются kp и kq раз.

Данное соображение позволяет эффективно использовать имеющиеся статистические данные для восстановления искомой последовательности. Основанием к этому служит идея создания кратных ребер в графе G_b.

3.4.1. Создание кратных ребер на основе весов

Рассмотрим ребро, соответствующее строке α, которая действительно присутствует в искомой последовательности G. Оценим математическое ожидание числа цепей из входных данных, которые целиком «укладываются» как подстроки в α. Рассмотрим любую цепь β. Вероятность попадания β в ту часть последовательности G, которая соответствует строке α составляет $\frac{|\alpha|-L+1}{|G|-L+1}$. Вероятность того, что цепь β считана без ошибок составляет $(1-\epsilon)^L$. Следовательно, математическое ожидание числа цепей, которые целиком попадают в α и при этом считаны правильно составляет $w_1(|\alpha|) = \frac{|\alpha|-L+1}{|G|-L+1} \cdot 2n(1-\epsilon)^L$ (это является фукнцией от длины строки α в условиях конкретных входных данных).

Теперь для ребер, которые соответствуют строкам достаточно большой длины (большей L), зная вес этого ребра w_α можно оценить, сколько раз строка α встречается в G: это число равно $w_\alpha / w_1(|\alpha|)$.

Округляя эту величину до ближайшего целого, получаем величину n_α — ожидаемое число появлений строки α в G. Именно столько кратных ребер в графе G_b следует создать — из вершины, являющейся началом рассматриваемого ребра, в вершину, являющуюся его концом.

Полученный мультиграф (граф, в котором разрешены кратные ребра) становится невзвешенным. Вершине, соответствующей началу и концу последовательности G соответствуют в нем некоторые вершины, хотя и неизвестно, какая из них — чему соответствует.

Теперь искомая последовательность G, если прослеживать появление ее подстрока на ребрах графа G_b есть некоторый путь f (возможно непростой — содержащий одну и ту же вершину более одного раза), начинающийся и заканчивающийся в вершинах, соответствующих концам G.

Если некоторое ребро в результате «клонирования» ребер стало присутствовать в графе G_b в n_α экземплярах, то ожидается, что в пути f это ребро должно встречаться n_α раз.

Это утверждение можно переформулировать проще: будем требовать, чтобы каждое «клонированное» ребро в пути f содержалось ровно один раз.

Тогда искомый путь f есть не что иное, как эйлеров путь в графе G_c с клонированными ребрами [3].

Применяя алгоритм поиска эйлерова пути в данном графе (основанный на обходе в глубину), получим последовательность ребер f, или одну из таких последовательностей, удовлетворяющих всем имеющимся условиям, если их несколько.

Раскрывая в данной последовательности каждое ребро как строку, соответствующую ему в графе G_b, получем последовательность символов G, которая удовлетворяет входным данным.

Глава 4. Экспериментальная часть

В данной главе содержатся результаты применения предлагаемого алгоритма для секвенирования ДНК на основе биоинформатических данных, полученных в результате эксперимента SRX00429 [14].

Эксперимент состоял в чтении цепей генома бактерии Escherichia coli K-12 с использованием инструмента Illumina Genome Analyzer.

4.1. Статистические данные

Входные данные содержат 10 408 224 пар цепей. Каждая цепь имеют длину 36 нуклеотидов, сдвиг позиций парных цепей относительно друг друга распределен нормально с математическим ожиданием $\mu = 179.38$ и стандартным отклонением $\sigma = 10.88$.

График распределения качественных характеристик нуклеотидов в биоинформатических данных рассматриваемого эксперимента представлен на рис. 4.1. На графике не отмечено значение качественной характеристики 0.9999, так как доля нуклеотидов с таким значением качества составляет 0.781, что много больше остальных ординат на графике.

Характеристики 0.602, 0.683 и особенно 0.9 (что соответствует качеству 4, 5 и 10 по шкале Phred [11]) встречаются чаще, чем разумно ожидать согласно наблюдаемому распределению. По всей видимости, это связано с конкретными типами технических погрешностей, возникающих при чтении нуклеотидов.

Также наблюдаемое распределение вероятностей позволяет сделать предположение, что у нуклеотидов, имеющих качественную характеристику 0.9999 (качество 40 по шкале Phred), данная характеристика является оценкой снизу, истинное же распределение качественной характеристики нуклеотидов распространяется на значения, большие 0.9999. Таким образом, у 78.1% нуклеотидов истинная качественная характеристика может оказаться заметно выше, чем

Рис. 4.1. Распределение качественных характеристик нуклеотидов во входных данных

приведенная во входных данных.

Даже без учета последнего соображения (формально его учесть не представляется возможным) средняя качественная характеристика нуклеотида равняется 0.985. Следовательно, вероятность наугад взятого нуклеотида быть ошибочно считанным составляет 0.015, а с учетом «массового» округления качественной характеристики до 0.9999, имеем $\epsilon \leq 0.015$.

4.2. Результаты применения алгоритма

Для успешной сборки генома организма-эукариота, в связи с огромным размером графа возможных продолжений G_c, необходимо выполнить построение отдельных его частей и вычисление индуктивных функций на вершинах этих частей в параллельной или распределенной

системе (см. разд. 3.1.2). В связи с этим, приведем результаты численных экспериментов, поставленных с использованием лишь части входных данных, опубликованных в результате эксперимента SRX00429.

Каждый эксперимент проводится следующим образом: выбирается подстрока H последовательности G, такая что $|H| \ll |G|$. Затем просматриваются все пары цепей из входных данных, и выбираются лишь те пары, в которых обе цепи либо являются подстроками H, либо отличаются от подстрок H только в ошибочно считанных нуклеотидах (в соответствии с качественными характеристиками этих нуклеотидов во входных данных).

Выбранные таким образом пары цепей становятся входными данными для программы, реализующей предлагаемый алгоритм. Следует отметить, что в связи с небольшой длиной строки H в поставленном таким образом эксперименте невозможно исследовать эффекты, связанные с наличием в G повторов, длина которых больше чем μ (среднее расстояние между парами цепей). Однако, в геномах организмов-эукариотов такие ситуации встречаются повсеместно [26].

В условиях такого эксперимента удается исследовать эффективность предлагаемых в настоящей работе методов обнаружения и исправления ошибок во входных данных.

Рис. 4.2. Зависимость доли исправленных нуклеотидов от среднего покрытия

На рис. 4.2 представлена зависимость доли нуклеотидов, которые удалось исправить на (предположительно) правильные в процессе рассмотрения ребер с незначительным весом в графе возможных продолжений (см. разд. 3.2), от среднего покрытия. Среднее покрытие — это математическое ожидание числа цепей, в которые входит наугад взятый нуклеотид, оно равняется $2nL/|H|$. (Для уменьшения значения среднего покрытия из входных данных исключалось соответствующее число пар цепей).

Также посредством данного типа экспериментов удается исследовать эффективность алгоритма наращивания подстроки (см. разд. 3.3), примененного после обнаружения и исправления ошибок. В качестве начального текущего состояния перебираются цепи, имеющие максимальную характеристику качества по всем нуклеотидам. Для каждой такой цепи запускается алгоритм наращивания, и в качестве итогового значения берется наибольшая длина подстроки, которую удается нарастить таким образом.

Результаты данного эксперимента для пяти строк H длины 50 000 представлены на рис. 4.3.

Рис. 4.3. Зависимость наибольшей длины полученной подстроки H от среднего покрытия

Таким образом, при имеющемся в реальных биоинформатических данных среднем покрытии, представляется возможным в алгоритме наращивания подстрок получать сколь угодно длинные подстроки — вплоть до всей искомой последовательности G.

Помимо описанных выше результатов, экспериментальные исследования предлагаемого алгоритма показали, что на персональном компьютере (с оперативной памятью 2 Гб) получается восстановить участки ДНК длиной в сто тысяч нуклеотидов. Поэтому для того, чтобы восстановить геном, содержащий миллионы или сотни миллионов нуклеотидов, требуется запускать параллельную версию данного алгоритма (см. разд. 3.1.2).

Для того, чтобы расшифровать с использованием предлагаемого алгоритма человеческий геном, содержащий 3 миллиарда пар нуклеотидов, требуется (по предварительным оценкам) примерно два года суммарного времени вычислений. Соответственно, при увеличении числа компьютеров до ста, время вычислний составит неделю, а при тысяче компьютеров — семнадцать часов. Все это становится возможным благодаря тому, что части графа возможных продолжений строятся независимо друг от друга. Если же оперативная память каждого вычислительного узла будет увеличена в k раз, то суммарное время работы станет в k раз меньше.

Заключение

Описанный в работе алгоритм позволяет секвенировать ДНК организмов-эукариотов на основе данных биологических экспериментов, представленных в виде пар последовательностей нуклеотидов. В работе предложены методы борьбы с влиянием ошибок в биологических данных и алгоритмы устранения таковых ошибок, что позволяет уменьшить необходимое число чтений, и следовательно, уменьшить время секвенирования ДНК произвольного организма-эукариота.

Источники

[1] *Беллман Р.* Динамическое программирование. М.: Изд-во иностранной литературы, 1960.

[2] *Дэн Г.* Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. СПб.: Невский диалект; БХВ-Петербург, 2003.

[3] *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО: БИНОМ, 2004.

[4] *Хопкрофт Д. Э., Мотвани Р., Ульман Д. Д.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.

[5] *Шень А.* Программирование: теоремы и задачи. М.: МЦНМО, 2004.

[6] *Anderson S.* Shotgun dna sequencing using cloned dnase i-generated fragments // *Nucleic Acids Research*. 1981. Vol. 9.

[7] *Batzoglou S., Jaffe D. B., Stanley K.* Arachne: A whole-genome shotgun assembler // *Genome Research*. 2002. Pp. 177–189.

[8] *Birol I., Jackman S. D., Nielsen C. B.* De novo transcriptome assembly with ABySS // *Bioinformatics*. 2009. Vol. 25, no. 21.

[9] *Denisov G., Walenz B., Halpern A. L.* Consensus generation and variant detection by celera assembler // *Bioinformatics*. 2008. Vol. 24, no. 8.

[10] *De Bruijn N. G.* A combinatorial problem // *Koninklijke Nederlandse Akademie v. Wetenschappen*. 1946. Vol. 49. Pp. 758–764.

[11] *Ewing B., Hillier L., Wendl M., Green P.* Basecalling of automated sequencer traces using phred. accuracy assessment // *Genome Research*. 1998. Pp. 175–185.

[12] *Fraser C. M., Casjens S., Huang W. M.* Genomic sequence of a lyme disease spirochaete borrelia burgdorferi // *Nature*. 1997. no. 390. Pp. 580—586.

[13] Illumina's Genome Analyzer IIe. http://www.illumina.com/systems/genome_analyzer.ilmn.

[14] Illumina sequencing of Escherichia coli str. K-12 substr. MG1655 genomic paired-end library. http://www.ncbi.nlm.nih.gov/sites/entrez?db=sra&report=full&term=SRX000429.

[15] Illumina Sequencing Technology. http://www.illumina.com/technology/sequencing_technology.ilmn.

[16] *Marcelo A. Nóbrega Yiwen Zhu I. P.-F. V. A., Rubin E. M.* Megabase deletions of gene deserts result in viable mice // *Nature*. 2004. no. 431. Pp. 988–993.

[17] *Int. Human Genome Seq. Consortium. 2001.* Initial sequencing and analysis of the human genome // *Nature*. no. 409. Pp. 860—-921.

[18] *Mouse Genome Seq. Consortium. 2002.* Initial sequencing and comparative analysis of the mouse genome // *Nature*. no. 420. Pp. 520—-562.

[19] Newbler documentation. http://doc.bioperl.org/bioperl-run/lib/Bio/Tools/Run/Newbler.html.

[20] *Ohno S.* So much 'junk dna' in our genome // *Evolution of genetic systems*. 1972. Vol. 23. Pp. 366–370.

[21] *Pevzner P. A., Tang H., Waterman M. S.* An eulerian path approach to dna fragment assembly // *Proceedings of National Academy of Science, USA*. 2001. Vol. 98.

[22] The Shortest Superstring Problem. http://www.cs.ust.hk/ golin/COMP670J/Lecture4.ps.

[23] *Ukkonen E.* On-line construction of suffix trees // *Algorithmica*. Vol. 14. Pp. 249–260. http://www.springerlink.com/content/KQ55005QU6479276.

[24] *Watson J., Crick F.* A structure for deoxyribose nucleic acid // *Nature*. 1953. Vol. 171. P. 737.

[25] *Whitelaw C. A., Barbazuk W. B., Pertea G.* Enrichment of gene-coding sequences in maize by genome filtration // *Science*. 2003. no. 302. Pp. 2118—-2120.

[26] *Zerbino D. R., Birney E.* Velvet: Algorithms for de novo short read assembly using de bruijn graphs // *Genome Research*. 2008. Vol. 18. Pp. 821–829. http://genome.cshlp.org/content/18/5/821.