Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Исенбаев Владислав Вольдемарович

Разработка системы секвенирования ДНК с использованием paired-end данных

Научный руководитель: доктор технических наук,
профессор А. А. Шалыто

Санкт-Петербург
2010

Оглавление

Введение

Актуальность темы. Многие задачи современной биологии, а также медицинские методы диагностики, требуют требуют знания нуклеотидных последовательностей дезоксирибонуклеиновой кислоты (ДНК) и рибонуклеиновой кислоты (РНК) живых организмов. Поэтому возникает необходимость в дешевых и быстрых методиках секвенирования (получения этой информации по физическому образцу). В настоящее время такие методики разработаны и активно применяются. Все они состоят из снятия информации с физического носителя — молекулы ДНК или РНК, и последующей обработки полученной информации на вычислительной технике. При этом в связи с большим объемом данных, а также с возможной неточностью данных, полученных на первом этапе, возникает необходимость в разработке эффективных, легко масштабируемых, точных и устойчивых к ошибкам алгоритмов секвенирования. При этом, ранее разработанные алгоритмы для решения обычной задачи секвенирования не подходят для новых методик считывания из-за нового формата данных (так называемые paired-end данные).

Объект исследования — задача секвенирования последовательности ДНК по данным, получаемым методиками считывания с paired-end информацией.

Исследование состоит из следующих частей:

- разработать эффективный алгоритм восстановления исходной последовательности по чтениям с paired-end информацией;
- разработать на основе полученного алгоритма методику устранения ошибок чтения, возникших в результате неточности измерений в первой фазе секвенирования;

- реализовать программное обеспечение ЭВМ, реализующее разработанные алгоритмы и проверить их эффективность экспериментально;

В ходе исследования требуется решить следующие **задачи**:

1. Разработать алгоритм для сведения задачи секвенирования с применением paired-end данных к хорошо изученной обычной задаче секвенирования.
2. Разработать алгоритм для выявления и устранения ошибочных чтений.
3. Оценить эффективность и масштабируемость полученных решений.
4. Реализовать алгоритмы в виде программы для ЭВМ.
5. Провести численные эксперименты, основанные на реальных данных.

Научная новизна. В бакалаврской работе разработан новый подход к решению задачи секвенирования с использованием paired-end информации, а также создан очень эффективный метод фильтрации ошибок чтения.

Кроме того, разработано программное обеспечение, реализующее созданные методики, и проведены численные эксперименты с ним.

Теоретическая и практическая значимость. Разработанные в данной работе методики эффективны и могут применяться в комплексе с аппаратурой севенирования нового поколения для получения дешевого и быстрого инструмента для исследований в различных областях биологии, а также в диагносических целях.

Структура работы. Работа состоит из введения, четырех глав и заключения.

В главе 1 рассмотрены основы биоинформатики, описаны исследуемые объекты и рассмотрены различные методики, применяемые на первой стадии процесса секвенирования. Описано, что такое paired-end данные, и сформулирована задача секвенирования по этим данным.

В главе 2 выполнен обзор существующих методик решения обычной задачи секвенирования (без информации о соответствующих парах) и задачи секвенирования с использованием paired-end информации.

В главе 3 описаны разработанные в данной работе алгоритмы.

В главе 4 описаны детали реализации, методики численных экспериментов и их результаты.

Глава 1. Обзор предметной области

1.1. Биоинформатика

Биоинформатика — наука на стыке двух дисциплин — информатики и биологии. В современной биологии возникают задачи, требующие обработки огромных объемов данных. Поэтому возникла необходимость автоматизации этого процесса с помощью вычислительной техники.

1.2. Молекулярная биология

1.2.1. Строение ДНК и РНК

ДНК и *РНК* — сложные полимеры, которые являются основными носителями информации в живых организмах. В большинстве случаев ДНК состоит из двух полимерных цепочек, состоящих из соеденных в последовательность нуклеотидов. Каждый нуклеотид состоит из азотистого основания, дезоксирибозы (сахара) и фосфатной группы. Азотистые основания, встречающиеся в ДНК, называются аденин (A), гуанин (G), тимин (T) и цитозин (C). Пары оснований A-G и T-C соответствуют друг другу (комплементарны) за счет водородных связей. Таким образом, две цепочки в ДНК должны быть комплеметарны друг другу (на соответствующих местах должны стоять комплементарные основания). Более того, цепочки ДНК ориентированы. Ориентации двух цепочек одной ДНК противоположны.

В эукариотических организмах (организмы, клетки которых содержат ядра) ДНК содержится в ядре клетки и является основным хранилищем генетической информации. За счет своей структуры, молекулу ДНК легко превратить в две идентичных копии. Это свойство используется в процессе деления клетки.

РНК — одноцепочечная версия ДНК, у которой в нуклеотиды

вместо дезоксирибозы входит рибоза. Набор оснований такой же, как и в ДНК, за исключением того что вместо тимина (T) используется урацил (U). В эукариотических организмов РНК используется для переноса информации с ДНК к митохондриям (для воспроизведения закодированных белков), для транспортировки аминокислот, в процессах регулирования экспрессии генов, а также носит еще множество других функций.

1.2.2. Строение белков

Белки — высокомолекулярные органические вещества, состоящие из альфа-аминокислот, соединённых в цепочку пептидной связью. В живых организмах аминокислотный состав белков определяется генетическим кодом. При синтезе в большинстве случаев используется 20 стандартных аминокислот. Их комбинации дают большое разнообразие свойств молекул белков. Кроме того, аминокислоты в составе белка часто подвергаются посттрансляционным модификациям, которые могут возникать и до того, как белок начинает выполнять свою функцию, и во время его «работ» в клетке.

Функции белков в клетках живых организмов очень разнообразны. Так, белки-ферменты катализируют протекание биохимических реакций и играют важную роль в обмене веществ. Некоторые белки выполняют структурную или механическую функцию, образуя цитоскелет, поддерживающий форму клеток. Также белки играют важную роль в сигнальных системах клеток, при иммунном ответе и в клеточном цикле.

Аминокислотную последовательность белка называют первичной структурой. Вторичная, третичная и т.д. структуры относятся к пространственной конфигурации молекулы белка.

1.3. Основные направления биоинформатики

1.3.1. Анализ последовательностей

В молекулярной биологии многие объекты представляются в виде последовательности символов (например, нуклеотиды в ДНК и РНК, аминокислоты в белках и т.д.). Так как длина этих последовательностей могут достигать миллиарды символов, для их обработки необходимо применение высокопроизводительных систем и алгоритмов.

Инструменты анализа последовательностей могут относиться к следующим темам:

1. Сравнение последовательностей с целью обнаружения совпадений и различий в них (*англ.* sequence alignment).
2. Обнаружение консервативных мотивов, рамок считывания, интронов, регуляторных элементов и других интересных с биологической точки зрения конструкций в последовательностях.
3. Анализ мутаций с целью выявления генетических маркеров.
4. Аннотация генов (определение их функциональности).
5. Секвенирование — получение последовательности из физического представления.

1.3.2. Вычислительная эволюционная биология

Эволюционная биология исследует происхождение и появление видов, также как их развитие во времени. Информатика помогает эволюционным биологам в нескольких аспектах:

1. Изучать эволюцию большого числа организмов, измеряя изменения в их ДНК, а не только в строении или физиологии.
2. Сравнивать целые геномы, что позволяет изучать более комплексные эволюционные события, такие как дупликация генов, латеральный перенос генов, а также предсказывать бактериальные специализирующие факторы.

3. Строить компьютерные модели популяций, для того чтобы предсказать поведение системы во времени.
4. Отслеживать появление публикаций, содержащих информацию о большом числе видов.

1.3.3. Анализ экспрессии генов

Уровень экспрессии многих генов может быть определен измерением концентрации соответствующих мРНК (матричная РНК) в клетке с помощью различных техник. Все они дают сильно зашумленные данные, для обработки которых разрабатываются инструменты статистического анализа для выделения полезной информации. С помощью этих инструментов становится возможным исследование патологической экспрессии генов в раковых клетках.

1.3.4. Предсказание структуры белков

Первичная структура белка легко определяется из кодирующей его последовательности ДНК. Целью данного направления исследований является разработка методов определения пространственной конфигурации молекулы белка по этим данным. Эта информация необходима для изучения механизма работы данного белка, а также для изучения паталогических процессов при некоторых заболеваниях (так называемых прионных инфекциях).

1.3.5. Моделирование биологических систем

Практичиски любой уровень биологической организации поддается моделированию. В связи со сложностью происходящих в этих системах процессов, различные их модели являются важным инструментом для их изучения.

1.4. Секвенирование ДНК

1.4.1. Постановка задачи

Секвенирование биополимеров ДНК — определение ее первичной нуклеотидной последовательности. В результате получается линейное символьное описание, которое сжато характеризует атомную структуру молекулы. Обычно до начала секвенирования производят амплификацию (увеличение числа копий) участка ДНК, последовательность которого требуется определить, при помощи полимеразной цепной реакции (реакции удвоения ДНК под действием фермента полимеразы).

1.4.2. Метод Сенгера (обрыва цепи)

Метод основан на присоединении к однонитчатой секвенируемой молекуле ДНК прямого или обратного секвенирующего праймера и синтезе de novo молекулы нуклеиновой кислоты с применением, например, дидезоксинуклеозидтрифосфатов (ddNTP). При этом синтезируются молекулы разной длины с определённым дидезоксинуклеотидом на конце. После разделения синтезированных молекул ДНК электрофорезом возможно определение первичной последовательности.

В современной интерпретации метод обрыва цепи позволяет секвенировать за один этап последовательность ДНК длиной около 800-1000 нуклеотидов [1].

1.4.3. Методы нового поколения

Современные методы секвенирования ориентированы на снижение цены и увеличение пропускной способности. При этом уменьшается длина считываемого фрагмента. Часть методов позволяет прочитать короткие куски начала и конца длинного фрагмента — получить дополнительную информацию о “спаренности” полученных чтений (так называемые paired-end данные) [2].

1.5. Метод дробовика

Все методы секвенирования позволяют обрабатывать только сравнительно небольшие фрагменты ДНК. В случае если требуется секвенировать более длинную цепочку, используются следующую методику.

Поступают следующим образом: делают большую случайную выборку коротких фрагментов заданной ДНК и секвенируют каждый отдельно. После этого различными способами по полученным данным восстанавливается исходная последовательность.

Эта методика называется методом дробовика (*англ.* shotgun sequencing) [3] [4].

1.6. Секвенирование методом double-barrel shotgun sequencing

1.6.1. Описание метода

Некоторые новые методы секвенирования позволяют читать длинный кусок с двух сторон — для каждой прочитанной строки будет известна строка, находящаяся от нее на некотором фиксированном расстоянии. Метод, который использует эту дополнительную информацию (paired-end data), называют double-barrel shotgun sequencing [5].

1.6.2. Описание алгоритмической задачи

Более формально, задача состоит в следующем: заданы пары строк $(U_i, V_i), |U_i| = |V_i| = k$. Требуется найти такую строку s, что каждая пара входит в нее, причем вхождения находятся примерно на расстоянии n друг от друга (параметры n и k зависят от используемого метода).

1.6.3. Теоретические проблемы

Искомая в задаче строка определена неоднозначно: если X представима в виде $X = PTATBTS$, и $|T| > n + k$, то строки X и $Y = PTBTATS$ никак не различить по исходным данным. На практике алгоритмы получают набор контигов — строки, которые гарантированно входили в исходную ДНК. В дальнейшем из них можно собрать исходную последовательность с помощью дополнительных данных, либо использовать их напрямую (например, для генетического анализа) [5].

1.7. Выводы по главе 1

Рассмотрены основы биоинформатики, описаны исследуемые объекты и различные методики, применяемые на первой стадии процесса секвенирования. Объяснено понятие paired-end данных, и сформулирована задача секвенирования ДНК по этим данным.

Глава 2. Описание известных подходов к задаче секвенирования

2.1. Формулировки задачи секвенирования методом дробовика

Задача секвенирования формулируется следующим образом: имеется набор строк (чтений), полученных каким-либо способом чтения ДНК. Требуется получить как можно больше сведений об изучаемой последовательности (в идеальном случае — восстановить полную последовательность нуклеотидов).

В варианте задачи, изучаемом в данной работе, известна также информация о парах — известны попарные соответствия между чтениями, прочитанными с двух концов одного фрагмента.

Неформальную формулировку задачи можно формализовать несколькими способами.

2.1.1. Формулировка в виде задачи о наименьшей надстроке

Дано множество строк x_i. Требуется найти минимальную по длине строку s такую, что все x_i входят в нее в качестве подстроки.

Данная задача является $\mathcal{NP}$-полной. Следовательно маловероятно что у нее существует достаточно эффективное решение [6].

2.1.2. Формулировка задачи как поиск гамильтонова пути

Дано множество строк x_i. Рассмотрим следующий ориентированный граф G: каждая его вершина v_i соответствует чтению x_i, каждое ребро (v_i, v_j) — перекрытию между строками x_i и x_j. В графе G требуется найти гамильтонов путь — путь, проходящий через каждую вершину ровно один раз.

Как и задача о наименьшей надстроке, данная задача $\mathcal{NP}$-полна [6] [7].

2.1.3. Формулировка задачи как поиск эйлерова пути

Дано множество строк x_i. Рассмотрим множество всех строк y_i длиной k, входящих в x_i, как подстроки. Рассмотрим следующий ориентированный граф G (рис.2.1): его вершины соответствуют всем возможным строкам длины $k-1$, каждое ребро соответствует строке y_i и идет из вершины, соответствующей префиксу y_i, в вершину, соответствующую суффиксу. Заметим, что этот граф — подграф графа де Брейна порядка k, содержащего все возможные ребра данного вида. В этом графе требуется найти эйлеров путь — путь, проходящий через каждое ребро ровно один раз.

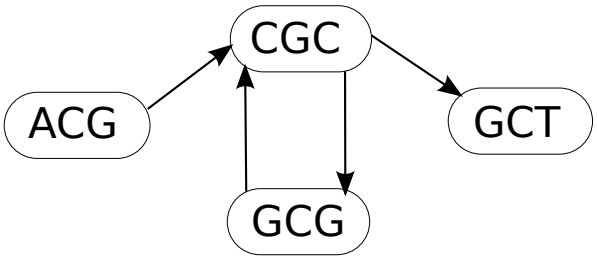

Рис. 2.1. Пример графа де Брейна. Эйлеровому пути соответствует строка ACGCGT.

Данная задача имеет эффективные полиномиальные решения [7].

2.2. Алгоритмы решения задачи секвенирования методом дробовика

Рассмотрев различные виды формализации задачи приведем соответствующие алгоритмы решения [8].

2.2.1. Алгоритмы на основе поиска эйлерова пути

Строится граф де Брейна, соответствующий формулировке задачи. После этого в нем ищется эйлеров путь одним из известных эффективных алгоритмов.

Данный метод используют следующие системы секвенирования: *EULER*, *Velvet*, *ABySS* [9] [10] [11] [12].

2.2.2. Алгоритмы на основе жадного решения задачи о наименьшей надстроке

В связи с тем, что задача о наименьшей надстроке $\mathcal{NP}$-полна, на данный момент не известно эффективных методов ее решения. Тем не менее, существует простой жадный алгоритм, дающий неплохие результаты на практике [13].

2.2.3. Алгоритм overlap-layout-consensus

Алгоритм overlap-layout-consensus использует различные эвристические методы решения задачи о гамильтоновом пути.

Данный метод используют следующие системы: *Newbler*, *Celera*, *Arachne*.

2.3. Обработка ошибок чтения

В связи с тем, что исходные данные получают из эксперимента, в них обязательно присутствуют ошибки. Для получения корректных результатов применяются различные методики их исправления.

Большинство существующих систем используют различные эвристические подходы для удаления ошибочных чтений. Обычно это различные методики анализа графа де Брейна, либо графа наложений [14].

2.4. Структура систем секвенирования

Большинство существующих систем секвенирования построены по следующей схеме:

1. Фильтрация ошибочных чтений.

2. Решение задачи секвенирования методом дробовика для отдельных чтений (на выходе контиги — длинные фрагменты исходных последовательностей).
3. Синтез контигов в более длинные последовательности за счет использования paired-end данных (*англ.* scaffolding).

2.5. Использование paired-end данных

Большинство методов дает на выходе набор контигов. После этого контиги объединяются в более длинные последовательности, используя данные о парах. Этот процесс называется scaffolding. Для этого строится граф с вершинами, соответсвующими контигам, и ребрами между ними с весами, равными числу пар, которые эти контиги объединяют. Затем к этому графу применяется алгоритм, сходный с алгоритмом overlap-layout-consensus [14].

2.6. Выводы по главе 2

Выполнен обзор существующих методик решения обычной задачи секвенирования и задачи секвенирования с применением paired-end информации. Известные методы секвенирования на основе paired-end информации недостаточно полно используют имеющиеся разработки по задаче обычного секвенирования.

Все известные методы устранения ошибок являются эвристическими, в то время как требуется разработать алгоритм, который не зависит от эвристик.

Глава 3. Новый подход к решению задачи с использованием paired-end данных

3.1. Граф де Брейна

Графом де Брейна k-ого порядка называют ориентированный граф, в котором вершинам соответствуют строки длины k, а ребрам — строки длины $k+1$, причем ребра идут из вершины, соответствующей префиксу строки ребра, в вершину, соответствующую суффиксу [15].

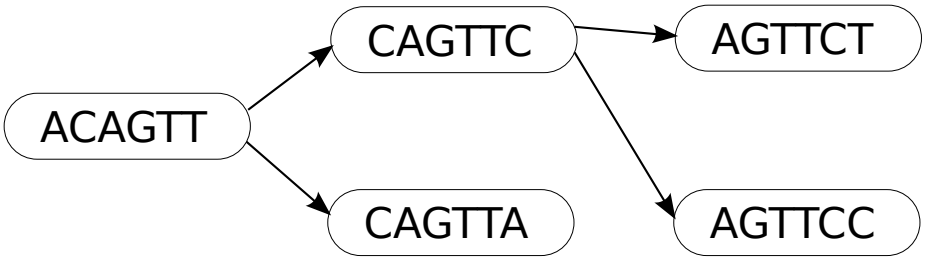

Рис. 3.1. Пример графа де Брейна

Из определения следует, что для любого ребра (u, v) суффикс u длины $k-1$ совпадает с префиксом v длины $k-1$. При этом любому пути в графе соответствует строка, составленная из перекрывающихся строк в вершинах. Например, в графе на рис. 3.1 пути из вершины ACAGT в вершину AGTTCC соответствует строка ACAGTTCC.

3.2. Алгоритм восстановления цельных фрагментов по известным концам и длине

Будем сводить задачу с paired-end данными к обычной задаче. Для этого требуется определить по известным парам чтений длинные фрагменты, из которых они были получены.

Формализуем эту задачу следующим образом: по набору пар чтений (u_i, v_i) (каждое чтение длиной k) и длине исходного фрагмента m

восстановить набор исходных фрагментов (рис. 3.2).

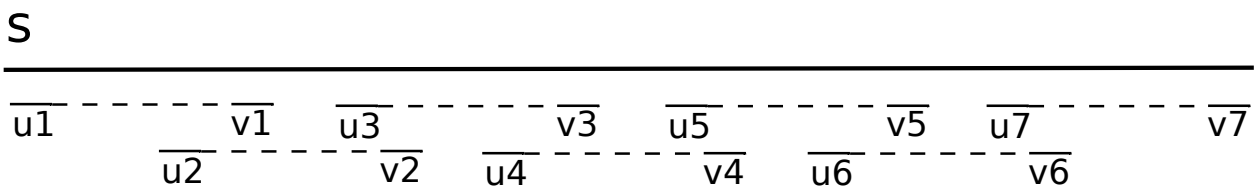

Рис. 3.2. Исходные данные

3.2.1. Переборный алгоритм (алгоритм 1)

Построим по строкам u_i, v_i граф де Брейна G (граф де Брейна, в котором вершины соответствуют данным строкам и проведены все возможные ребра). Каждой паре строк (u_i, v_i) соответствует фрагмент исходной последовательности x_i длиной m. Рассмотрим все подстроки x_i длиной k: $y_i^j = x_i[j\,..\,j+k-1]$, причем $y_i^j = u_i$ и $y_i^{m-k} = v_i$. Предположим, что покрытие достаточно для того чтобы каждая подстрока исходной строки была прочитана хотя бы раз без ошибок. Тогда все y_i^j будут встречаться в качестве вершин G. Более того, между y_i^j и y_i^{j+1} в G будет ребро, соответствующее строке $x_i[j\,..\,j+k]$. Таким образом, в G есть путь из $y_i^1 = u_i$ в $y_i^{m-k} = v_i$ длиной $m-k$, соответсвующий строке x_i, и наша задача сводится к поиску в графе G пути между двумя вершинами заданной длины.

Рассмотрим следующий алгоритм:

Данный алгоритм находит все строки, соответствующие путям в графе G из вершины u в вершину v и имеющие длину от minDepth до maxDepth. Процедура Recursive-Walk реализует рекурсивный обход графа. В процессе работы поддерживаются следующие инварианты:

- *depth* содержит длину рассматриваемого пути;
- *pathString* содержит прочтение рассматриваемого пути.

Когда текущая длина пути превышает maxDepth, поиск обрывается.

Получим эмпирическую оценку сложности алгоритма. Будем исходить из предположения, что граф G случаен, и средняя

Алгоритм 1 WALK

```
WALK(G, u, v, foundPaths)
1  return RECURSIVE-WALK(G, G.vertices[u], G.vertices[v], 0, u, foundPaths)

RECURSIVE-WALK(G, u, v, depth, pathString, foundPaths)
 1  if depth > maxDepth
 2      // отсечение перебора при превышении максимальной длины
 3      return
 4  if u == v and depth ⩾ minDepth
 5      // найден путь
 6      foundPaths.add(pathString)
 7  for (u, u') ∈ G.edges
 8      // перебираем очередное ребро в пути
 9      newPathString = pathString + (u, u').symbol
10      // аттрибут symbol ребра (u, v) содержит символ c
11      // такой, что (u + c)[2..u.length] == v
12      RECURSIVE-WALK(G, u', v, depth + 1, newPathString, foundPaths)
```

степень вершины в нем d. Тогда вызовов RECURSIVE-WALK с $depth == 0$ будет $1 = d^0$, с $depth == 1$ их будет d^1, и т.д. $depth ==$ maxDepth их будет d^{maxDepth}. Таким образом, всего вызовов будет $\sum_{i=0}^{\text{maxDepth}} d^i = \frac{d^{\text{maxDepth}+1}-1}{d-1} = O(d^{\text{maxDepth}})$.

Разработанный алгоритм, как и следующий соответствует решению поставленной задачи, и позволяет сводить задачу с paired-end данными к обычной задаче секвенирования.

3.2.2. Meet-in-the-Middle алгоритм (алгоритм 2)

Для ускорения алгоритма 1 можно применить подход Meet-in-the-Middle.

Рассмотрим алгоритм 2. Он работает следующем образом: для всех вершин графа находятся пути из начальной вершины длиной не более $\lceil \frac{\text{maxDepth}}{2} \rceil$ и пути в конечную вершину длиной не более $\lfloor \frac{\text{maxDepth}}{2} \rfloor$. После этого в каждой посещенной вершине конкатенируются все возможные префиксы и суффиксы пути требуемой длины.

Цикл в строках 5-7 процедуры WALK-MITM можно реализовать за $O(\text{maxDist} \cdot |foundPaths|)$ (каждый конечный путь рассматривается максимум maxDist раз). Вызовы функций не сильно отличаются

от рекурсивного обходчика в алгоритме 1 и выполняются за $O(d^{\lceil \frac{\text{maxDist}}{2} \rceil})$. Таким образом, в целом алгоритм выполняется за $O(d^{\lceil \frac{\text{maxDist}}{2} \rceil} + \text{maxDist} \cdot |foundPaths|)$.

Алгоритм 2 Walk-MitM

```
Walk-MitM(G, u, v, foundPaths)
1  visited = Empty-Set
2  Recursive-Walk-Forward(G, G.vertices[u], 0, u, visited)
3  Recursive-Walk-Backward(G, G.vertices[v], 0, "")
4  for node ∈ visited
5       for prefix ∈ node.forwardPaths, suffix ∈ node.backwardPaths,
6       minDepth ⩽ |prefix| + |suffix| ⩽ maxDepth
7            foundPaths.add(prefix + suffix)

Recursive-Walk-Forward(G, u, depth, pathString, visited)
1  if 2 · depth > maxDepth + 1
2       return
3  visited.add(u)
4  u.forwardPaths.add(pathString)
5  for (u, u') ∈ G.edges
6       newPathString = pathString + (u, u').symbol
7       Recursive-Walk-Forward(G, u', depth + 1, newPathString, foundPaths)

Recursive-Walk-Backward(G, u, v, depth, pathString)
1  if 2 · depth > maxDepth
2       return
3  u.backwardPaths.add(pathString)
4  for (u', u) ∈ G.edges
5       newPathString = pathString + (u', u).symbol
6       Recursive-Walk-Backward(G, u', depth + 1, newPathString, foundPaths)
```

3.2.3. Выявление ошибок чтения (алгоритм 3)

Модифицировав данный алгоритм, можно получить эффективный метод устранения ошибочных чтений.

Рассмотрим исходную последовательность ДНК: в ней каждая подстрока длины k (рис. 3.3) перекрывается как минимум $m - k$ фрагментами (кроме близких к краям).

Посчитаем для каждой вершины графа, сколько раз через нее проходят пути, найденные алгоритмом. После этого, будем считать ошибочными все чтения, соответствующие вершинам с числом путей, не превосходящим заданный порог.

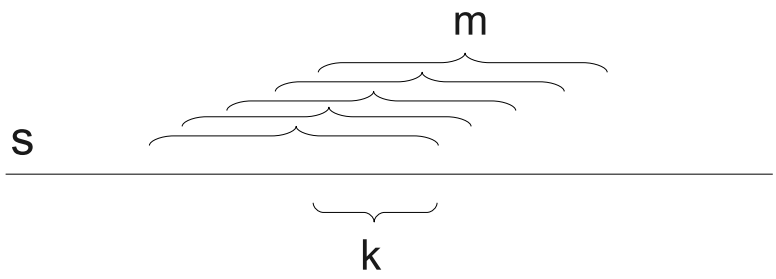

Рис. 3.3. Перекрытие фрагментов

Для подсчета путей, проходящих через вершины, модифицируем алгоритм 1, получив алгоритм 3.

Алгоритм 3 WALK2

```
WALK2(G, u, v, foundPaths)
1  return RECURSIVE-WALK2(G, G.vertices[u], G.vertices[v], 0, u, foundPaths)

RECURSIVE-WALK2(G, u, v, depth, pathString, foundPaths)
 1  if depth > maxDepth
 2      return 0
 3  paths = 0
 4  if u == v and depth ⩾ minDepth
 5      foundPaths.add(pathString)
 6      paths = paths + 1
 7  for (u, u') ∈ G.edges
 8      newPathString = pathString + (u, u').symbol
 9      paths = paths + RECURSIVE-WALK2(G, u', v, depth + 1, newPathString, foundPaths)
10  u.paths = u.paths + paths
11  return paths
```

Таким образом, разработан алгоритм, отличающийся от известных тем, что он не является эвристическим.

3.2.4. Оценка производительности

Предположим, что исходная последовательность является случайной последовательностью оснований длиной n. Тогда в графе де Брейна в среднем будет $n - 1 + 4n\frac{n}{2^{n-1}}$, следовательно средняя степень вершины будет $d \approx 1 + \frac{n}{2^{k-3}}$. Среднее время, затраченное на поиск всех путей между парой вершин будет $O(d^{m-k})$.

При $m = 200, k = 36$ алгоритм остается эффективным при $n \approx 10^8$.

Если в алгоритме использовать метод meet-in-the-middle вместо

простого перебора, оценку можно улучшить до $O(d^{\frac{n-m}{2}})$.

При $m = 200, k = 36$ такой алгоритм можно эффективно применять для $n \approx 10^9$.

3.3. Общая схема решения

Общая схема решения состоит в следующем:

1. Построить граф де Брейна.
2. Провести первую итерацию алгоритма восстановления фрагментов.
3. Воспользовавшись полученной информацией о частоте использования вершин, удалить ошибочные вершины из графа.
4. Провести вторую итерацию алгоритма.
5. Воспользоваться одним из общих алгоритмов секвенирования, не использующий информацию о парах.

3.4. Масштабируемость решения

Алгоритмы 1, 2 и 3 обрабатывают все пары чтений полностью независимо. Поэтому метод хорошо поддается масштабированию на распределенные системы.

Рассмотрим схему такой распределенной системы:

1. Построить граф де Брейна на центральном узле и раздать его всем остальным узлах (либо хранить его в некоторой распределенной базе данных).
2. Равномерно распределить пары по узлам.
3. Каждый узел выполняет алгоритм 3 на прикрепленных к нему парах.
4. Полученные веса вершин суммируются по всем узлам.
5. Из графа удаляются ошибочные вершины, и граф снова раздается узлам.

6. Узлы выполняют алгоритм 1 или алгоритм 2 на прикрепленных к нему парах.
7. Восстановленные фрагменты собираются и передаются алгоритму решения обычной задачи секвенирования.

3.5. Выводы по главе 3

Разработаны три алгоритма, первые два из которых предназначины для сведения задачи секвенирования с paired-end данными к обычной задаче секвенирования, а третий — фильтрует ошибки в исходных данных. Первый и второй алгоритмы отличаются сложностью реализации и временем работы, при этом первый — проще, второй — эффективней.

Глава 4. Реализация алгоритма и экспериментальные данные

4.1. Реализация

В рамках данной работы создана реализация метода на языке программирования *Java*.

4.1.1. Реализация графа де Брейна

Граф де Брейна реализован на основе хеш-таблицы из стандартной библиотеки языка *Java*:

```
public class Graph {

    // отображение из строки ДНК в информацию о вершине
    private HashMap<IDna, GraphNode> nodeMap;

    // конструктор
    public Graph() {
        ...
    }

    // получение информации о вершине по строке
    public GraphNode getNode(IDna data) {
        ...
    }

    // вставка новой вершины в граф
    public void addNode(IDna dna) {
        ...
    }

    // получение размера графа
    public int size() {
        ....
    }

    // получение множества всех вершин
    public Iterable<GraphNode> nodes() {
        ...
    }
}
```

```
public class GraphNode {
    // массив исходящих ребер
    public GraphNode[] edges;
    // массив входящих ребер
    public GraphNode[] revEdges;
    // количество проходящих через вершину путей
    public int paths;
    // вершина, обратно комплементарная данной
    public GraphNode dual;
}
```

4.1.2. Реализация алгоритмов 1 и 3

```
private static int recursiveWalk(GraphNode from, GraphNode to, int depth,
                        int minDepth, int maxDepth, boolean ignoreBad) {
    if (depth > maxDepth || ignoreBad &&
        from.paths + from.dual.paths < minPaths) {
        return 0;
    }
    int paths = 0;
    if (from == to && depth >= minDepth) {
        paths = 1;
        length = depth;
        if (foundPaths != null) {
            foundPaths.add(dnaBuilder.snapshot());
        }
    }
    for (int i = 0; i < 4; i++) {
        if (from.edges[i] != null) {
            dnaBuilder.append((byte) i);
            paths += dfs(from.edges[i], to, depth + 1, minDepth,
                         maxDepth, ignoreBad);
            dnaBuilder.removeLast();
        }
    }
    from.paths += paths;
    return paths;
}
```

Если параметр *ignoreBad* равен TRUE, процедура работает как алгоритм 1 и игнорирует вершины, помеченные как ошибочные, иначе она работает как алгоритм 3.

4.2. Экспериментальные данные

Программа тестировалась на наборе синтетических данных, сконструированных из реальных последовательностей ДНК бактерий.

Так сделано для того, чтобы была возможность оценить эффективность устранения ошибок и алгоритма в целом.

Для создания тестовых примеров использовались следующие методы симулирования физического этапа секвенирования:

- Из образца ДНК выбирались все возможные фрагменты длиной m. В качестве данных выбирались пары (префикс, суффикс), полученные из этих фрагментов. Таким образом генерировались идеальные данные без ошибок.
- Выбиралась случайная длина фрагмента по нормальному распределению с матожиданием m и заданной дисперсией, случайным образом выбирался фрагмент полученной длины, выбиралась пара (префикс, суффикс) и в нее вносились случайным образом ошибки в соответствии со статистикой ошибок в реальных данных. Таким способом генерировались зашумленные данные, максимально приближенные к реальным.

На симулированных данных без ошибок для генома бактерии *Escherichia Coli* алгоритм уникально восстанавливает фрагменты для 98% процентов пар. Этими фрагментами обеспечивается 99.5% покрытия исходной последовательности. Если же брать все фрагменты, найденные алгоритмом, 0.1% из них «ошибочен» (не встречается в исходной последовательности)

На симулированных сильно зашумленных данных (60% чтений с хотя бы одной ошибкой) для генома бактерии *Haemophilus Influenzae* алгоритм возвращает 0.01% ошибочных фрагментов и обеспечивает 99.5% покрытия исходной последовательности, если брать только фрагменты, уникальные для пары, и 0.07% ошибочных фрагментов и обеспечение 99.9% покрытия при взятии всех фрагментов.

4.3. Выводы по главе 4

Описаны детали реализации, методики численных экспериментов и их результаты.

Заключение

Приведем результаты работы, выносимые на защиту:

- разработано два алгоритма сведения задачи секвенирования с использованием paired-end данных к обычной задаче секвенирования, первый из которых является простым в реализачии, а второй — более эффективен;
- разработан алгоритм устранения ошибок чтения, не использующий эвристики;
- проведен теоретический анализ эффективности созданных алгоритмов;
- разработанные алгоритмы реализованы ввиде программы для ЭВМ;
- проведены численные эксперименты на синтетических данных, построенных из реальных ДНК живых организмов и проведен анализ полученных результатов.

Таким образом, поставленные задачи выполнены, и цель достигнута.

Источники

[1] *Sanger F., Niclein S., Coulson A. R.* Dna sequencing with chain-terminating inhibitors // *Proc Natl Acad Sci USA*. 1977. Vol. 74. Pp. 5463–5467.

[2] *Schuster S. C.* Next-generation sequencing transforms today's biology // *Nature Methods*. 2008. Vol. 5. P. 16–18.

[3] *Staden R.* A strategy of dna sequencing employing computer programs // *Nucleic Acids Research*. 1979. Vol. 6, no. 7. P. 2601–10.

[4] *Anderson S.* Shotgun dna sequencing using cloned dnase i-generated fragments // *Nucleic Acids Research*. 1981. Vol. 9, no. 13. P. 3015–27.

[5] *Roach J., Boysen C., Wang K., Hood L.* Pairwise end sequencing: a unified approach to genomic mapping and sequencing // *Genomics*. 1995. Vol. 26. P. 345–353.

[6] *Garey M., Johnson D. S.* Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1979.

[7] *Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.* Introduction to Algorithms. 3rd edition. MIT Press, 2009.

[8] *Kececioglu J. D., Myers E. W.* Combinatorial algorithms for dna sequence assembly // *Algorithmica*. 1995. Vol. 13, no. 1-2. Pp. 7–51.

[9] *Simpson J. T., Wong K., Jackman S. D., Schein J. E., Jones S. J., Birol I.* Abyss: A parallel assembler for short read sequence data // *Genome Research*. 2009. Vol. 19.

[10] *Birol I., Jackman S. D., Nielsen C. B., Qian J. Q., Varhol R., Stazyk G., Morin R. D., Zhao Y., Hirst M., Schein J. E., Horsman D. E., Connors J. M., Gascoyne R. D., Marra M. A., Jones S. J.* De novo transcriptome assembly with abyss // *Bioinformatics*. 2009. Vol. 25.

[11] *Zerbino D., Birney E.* Velvet: algorithms for de novo short read assembly using de bruijn graphs // *Genome Research*. 2008. Vol. 18.

[12] *Pevzner P. A., Tang H., Waterman M. S.* An eulerian path approach to dna fragment assembly // *Proc Natl Acad Sci USA*. 2001. Vol. 98, no. 17. Pp. 9748–9753.

[13] *Kasahara M., Morishita S.* Large-scale genome sequence processing. Imperial College Press, 2006.

[14] *Zerbino D., McEwen G. K., Margulies E. H., Birney E.* Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler // *PLoS One*. 2009. Vol. 4.

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