Санкт-Петербургский государственный университет
Математико-механический факультет

Кафедра системного программирования

Разработка алгоритмов обработки графа де Брюина в задаче геномного ассемблирования

Дипломная работа студента 545 группы
Нурка Сергея Юрьевича

Научный руководитель _____Вяхкхи Н.И.

Рецензент _____Алексеев М.А.

"Допустить к защите"
заведующий кафедрой _____д.ф-м.н, проф. Терехов А.Н.

Санкт-Петербург
2011

Saint-Petersburg State University
Mathematics and Mechanics Faculty

Software Engineering Department

Development of de Bruijn graph processing algorithms for genome assembly problem

Graduate paper by
Sergey Nurk
545 group

Scientific advisor _____ N.I. Vyahhi

Reviewer _____ Assistant Professor M.A. Alekseyev

"Approved by"
Head of Department _____ Professor A.N. Terekhov

Saint-Petersburg
2011

Содержание

Введение **3**

1 Графы де Брюина и постановка задачи **8**
1.1 Основные понятия 8
1.2 Граф де Брюина 9
1.2.1 Классическое определение 9
1.2.2 Граф де Брюина для идеального множества ридов . . 10
1.2.3 Граф де Брюина для множества ридов 12
1.2.4 Сжатый граф де Брюина 14
1.3 Общая схема работы ассемблера, основанного на графах де Брюина 15
1.3.1 Предварительное исправление ошибок 15
1.3.2 Построение графа де Брюина 16
1.3.3 Коррекция графа 16
1.3.4 Разрешение повторов 17
1.3.5 Получение контигов и скэффолдов 18
1.3.6 Этап консенсуса 19
1.4 Постановка задачи 19
1.4.1 Разрабатываемый ассемблер 19
1.4.2 Цели работы 21

2 Анализ предметной области **23**
2.1 Классификация основных типов ошибочных участков графа де Брюина 23
2.2 Обзор существующих ассемблеров 24
2.2.1 Euler 24
2.2.2 Velvet 25
2.2.3 ALLPATHS 26
2.2.4 ABySS 27
2.2.5 SOAP *de novo* 27

3 Предлагаемое решение **28**
3.1 Построение графа де Брюина 28
3.2 Построение сжатого графа 28
3.3 Графовые операции 29
3.4 Сжатый граф, учитывающий сопряженность 30
3.5 Дополнительная информация 32
3.6 Покрытие 33
3.6.1 Первоначальный подсчет 33
3.6.2 Правила преобразования 34
3.7 Информация о межреберных расстояниях 34
3.7.1 Первоначальный подсчет 35
3.7.2 Правила преобразования 36
3.7.3 Кластеризация “на лету” 37
3.8 Алгоритмы коррекции графа 38
3.8.1 Удаление тупиков 39

3.8.2 Удаление пузырей 40
3.8.3 Удаление ошибочных соединений 41

4 Реализация и результаты эксперимента 42
4.1 Основные используемые структуры 42
4.2 Концепция “наблюдателей” 43
4.3 “Умные” итераторы 45
4.4 Экспериментальные данные 45

Заключение 48

Использованные источники 50

Введение

Что такое геном

Геном — это полная совокупность наследственной информации организма. Обычно она закодирована в молекулах дезоксирибонуклеиновой кислоты (ДНК). Она определяет полный жизненный цикл организма.

Молекула ДНК состоит из двух линейных длинных полимеров, называемых *стрендами*, состоящих из более простых молекул, называемых *нуклеотидами* или *основаниями*. Есть всего четыре различных оснований: аденин (A), цитозин (C), гуанин (G) и тимин (T). Два стренда расположены близко друг к другу и закручены в двойную спираль. Химическая структура стренда несимметрична, поэтому можно формально определить его направление. В молекуле ДНК стренды антипараллельны: нуклеотиды располагаются один напротив другого, а направления стрендов противоположны. Оказывается, что зная конкретный нуклеотид (тип основания) на одном стренде можно сказать какой будет на соответствующем месте другого стренда. Этот факт носит название *комплементарной парности оснований*. A находится только напротив T, а C — напротив G.

Подытоживая вышесказанное, если мы выпишем стренды ДНК как две длинные строки в четырех буквенном алфавите с учетом направления, то они окажутся реверс-комплементарными (в дальнейшем объекты, находящиеся в этом отношении будем называть *сопряженными*) друг другу: если прочитать одну из них от конца к началу, заменяя каждый нуклеотид комплементарным, получится вторая.

Внутри клетки молекулы ДНК организуются в более сложные структуры, называемые *хромосомами*.

Длиной генома будем считать суммарную длину всех хромосом.

Для простоты везде далее (если не сделано специальных оговорок) считается, что весь геном заключен в одной единственной хромосоме (молекуле ДНК).

Помимо здорового интереса, у проектов по выявлению конкретной последовательности ДНК есть огромное количество областей применения. В них, например, входят:

- персональная медицина (диагностика генетических заболеваний)

- поиск и производство новых антибиотиков
- выявление эволюционных закономерностей и степеней родства между организмами
- анализ генов и генная инженерия

Секвенирование ДНК и сборка из фрагментов

Технология, позволившая бы "прочитать"длинный (длиннее 10 тыс. оснований) участок генома нуклеотид за нуклеотидом остается неизвестной. Но были разработаны технологии, работающие в обход этого ограничения. Идея состоит в генерации огромного количества сравнительно коротких фрагментов из большого числа копий одного и того же генома и разгадывании получившейся гигантской головоломки из перекрывающихся элементов.

Таким образом, проблема восстановления последовательности нуклеотидов ДНК в подавляющем большинстве случаев сводится к генерации фрагментов (биологическая задача) и реконструкции по ним исходной последовательности (вычислительная задача).

Процесс генерации фрагментов называется *секвенированием*. Аппараты, осуществляющие этото процесс называют *секвенаторами*.

Процесс реконструкции по ним исходной последовательности называется *ассемблированием* или *сборкой*. Как мы увидим в дальнейшем, сборка является очень сложной алгоритмической задачей, многие трудности, возникающие при решении которой, так и не преодолены. Програмные комплексы, осуществляющие сборку называют *ассемблерами*.

Первую технологию секвенирования ДНК придумали в 1970-х, Уолтер Гилберт и Фред Сангер, получившие за это Нобелевскую премию. С помощью различных приемников этой технологии было секвенировано большое количество вирусных и бактериальных геномов. Но эти технологии были слишком медленными и дорогими. Проекты по секвенированию млекопитающих, таких как человек или мышь растягивались на годы и стоили миллионы долларов.

Начиная с 2000 года все большее распространения начали получать технологии, объединенные под общим названием технологий секвенирования

следующего поколения (NGS, Next Generation Sequencing technologies). Они позволили снизить стоимость и время секвенирования в сотни раз. Но при этом, если длина фрагментов при использовании классических технологий, могла достигать 1000 нуклеотидов, то наиболее популярные [1] из NGS технологий даже и сегодня, по прошествии десяти лет, предоставляют фрагменты длина которых всего $100 - 200$ нуклеотидов.

Переход на NGS технологии значительно усложнил задачу сборки и сделал существовавшие ранее подходы просто-напросто неприменимыми.

На самом деле, в области ассемблирования есть две различные задачи:

- сборка при наличии ранее восстановленного генома особи того же вида, что и исследуемая (reference genome)
- *de novo* сборка, когда таковой отсутствует

Везде далее в данной работе идет речь именно о подходах к *de novo* сборке.

Павел Певзнер с соавторами ([10]) предложил следующую метафору. Восстановление генома похоже на подрыв ящика, в котором находились экземпляры сегодняшнего номера газеты "Times" и последующую попытку восстановления содержания газеты по получившимся кусочкам. Некоторые участки конкретного экземпляра могли быть полностью уничножены взрывом, а кусочки разных газет будут перекрываться и содержать одни и те же участки текста.

Но хотя задача сборки генома по постановке и похожа на восстановление газеты по клочкам, на самом деле она значительно сложнее. И дело не только в том, что для достижения приемлемого результата при восстановлении, скажем, генома млекопитающего, приходится иметь дело с миллиардами фрагментов.

Во-первых, вспомним, что газета написана на некотором естественном языке, чьи правила могут подсказать нам взаимное расположение кусочков вне зависимости от того перекрываются они или нет. Правила "языка" ДНК все еще в основном не известны биологам, так что практически невозможно определить даже примерное взаимное расположение двух неперекрывающихся ридов.

[1] Наиболее популярными на сегодняшний день являются секвенаторы фирмы Illumina

Во-вторых, “алфавит” нуклеотидов содержит всего четыре буквы (A, T, G, C). Это сильно усложняет задачу, так как большое количество наблюдаемых нами наложений произошли совершенно случайно и не соответствуют одному участку исходной последовательности.

В-третьих, ДНК содержит большое количество последовательностей, повторенных в геноме большое количество раз, быть может, с небольшими изменениями. Такие участки называются *повторами*. Например, так называемый повтор Alu, последовательность длины около 300 нуклеотидов, встречается (с небольшими вариациями) в человеческом геноме миллион.

В-четвертых, даже самые современные методы секвенирования не идеальны и получающиеся фрагменты содержат большое количество ошибок, попадание которых в итоговую сборку крайне нежелательно.

И наконец, большинство современных технологий секвенирования имеют возможность генерации не только одиночных, но и парных фрагментов, причем расстояние в геноме между фрагментами, принадлежащими одной паре считается известным. Правильное использование этой дополнительной “парной” информации, как мы увидим в дальнейшем, может значительно повысить качество итоговой сборки, но является весьма непростой задачей, универсального метода для решения которой не существует.

Таким образом, задача сборки генома по коротким фрагментам является одной из наиболее важных и алгоритмически трудных задач в биоинформатике. Несмотря на то, что за последние 20 лет ей занимались сотни исследователей и были реализованы десятки ассемблеров, в ней и по сей день остается огромное количество нерешенных проблем.

О теоретической сложности задачи ассемблирования

Задача ассемблера состоит в нахождении строки ДНК, наиболее хорошо согласованной с данными ему на вход фрагментами. Далее в этом разделе предполагается, что данные сгенерированы без ошибок, что является очень серьезным упрощением решаемой задачи.

Как уже упоминалось, объемы входных данных для алгоритма, огромны. Ассемблерам, работающим с фрагментами, полученными по NGS технологиям приходится обрабатывать миллионы и миллиарды (если речь о сборке генома млекопитающего) фрагментов.

В таких условиях первостепенным является вопрос об асимптотике применяемого алгоритма. Квадратичный (от размера входных данных) алгоритм, даже с чрезвычайно малой константой уже оказывается более чем сомнительным. 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.

Обычно, говоря о формальной постановке, задачу ассемблирования переформулируют как известную задачу о *кратчайшей общей надстроке* (SSP, Shortest Superstring Problem) для множества входных фрагментов.

Задача о кратчайшей общей надстроке [3] 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.

Если не делать никаких дополнительных предположений, то задача SSP является NP-полной [27]. Из этого следует, что не только ответ на вопрос о существовании, но и поиск самой кратчайшей строки невозможно осуществить за полиномиальное время, если только не выполняется $P = NP$.

Обычно современным ассемблерам приходится иметь дело с парными фрагментами. В работе [26] задача SSP была расширена на множество парных фрагментов, которые в итоговой строке должны находиться в фиксированном диапазоне расстояний друг от друга и показано, что в отсутствии дополнительных предположений, эта задача также NP-полна.

Получается, что найти точное решение задачи ассемблирования в такой постановке оказывается невозможно, так как на существование полиномиального алгоритма рассчитывать не приходится.

1 Графы де Брюина и постановка задачи

1.1 Основные понятия

Риды — последовательности, получаемые при секвенировании и содержащие информацию о фрагментах генома. Можно считать, что рид — это строка четырех буквенного алфавита *нуклеотидов*, соответствующая некоторому фрагменту стренда. При этом входные данные содержат "'вперемежку" риды, соответствующие фрагментам различных стрендов (и обычно даже различных хромосом).

О риде, полученном по некоторому участку стренда говорят как о риде, *покрывающем* данный участок.

Покрытие участка генома (в частности всего генома) — среднее количество ридов, покрывающих отдельно взятый нуклеотид на этом участке. Обычно предполагают, что геном покрыт ридами более-менее равномерно.

Все секвенаторы совершают ошибки, наиболее частая из которых — замена одного нуклеотида на другой (например, в риде находится T, а в соответствующей позиции генома — A). Для того чтобы помочь исследователям, в дальнейшем анализирующим эти данные, современные секвенаторы, вместе с каждым ридом, предоставляют *характеристику качества* прочтения каждого из его нуклеотидов. Это информация о степени уверенности секвенатора в правильности того или иного нуклеотида.

Наиболее распространенной среди всех характеристик качества, являются так называемые Phred значения (Phred value coeffitient) $Q = -10log_{10}P$, где P — предполагаемая самим секвенатором вероятность ошибки в конкретном нуклеотиде.

К примеру, если Phred значение для некоторой позиции равно 30, вероятность того, что на этом месте должен стоять другой нуклеотид, равна 1/1000.

В силу технологических особенностей, качество нуклеотидов значительно снижается ближе к концу рида. И если для современных секвенаторов фирмы Illumina, Phred значения в начале и середине рида составляют, в среднем $30 - 40$, то к концу рида они падают до значений порядка 5.

Расстоянием между двумя участками одного стренда будем называть разность их начальных позиций.

Как уже отмечалось, большинство популярных секвенаторов имеют возможность генерации *парных* ридов. Оба рида из пары соответствует участкам одного и того же стренда. Расстояние между этими участками будем называть *внутренним расстоянием* парного рида.

Каждый набор парных ридов, соответствующих одному запуску секвенатора, характеризуется длиной одиночных ридов и предположительным внутренним расстоянием d.

В силу различных технологических причин, истинное внутреннее расстояние отдельно взятого парного рида может отличаться от d.

Желаемое внутреннее расстояние d производимых парных ридов задается при конкретном запуске секвенатора. О результатах такого запуска говорят как о множестве ридов (*библиотеке ридов*) с предположительным внутренним расстоянием d. Ограничения на возможные значения d варьируются от секвенатора к секвенатору и обычно находятся в диапозоне от 100 до 10000 нуклеотидов.

k-мер — строка длины k в алфавите A,T,G,C. (k-mer$\in \{A, C, G, T\}^k$)

1.2 Граф де Брюина

Граф де Брюина — компактная структура, основанная на k-мерах, оказавшаяся крайне полезной для представления множества коротких ридов, обеспечивающих высокое покрытие графа.

Подход к сборке генома, основанный на использовании этих графов был предложен Павлом Певзнером с соавторами в 2000 году и с тех пор стал основным подходом к ассемблированию данных от NGS (Next Generation Sequencing) технологий.

Более того, на сегодняшний день, это по сути единственный подход, которым можно хотя бы пытаться собирать большие геномы, содержащие миллиарды оснований, по коротким ридам ≈ 200 нуклеотидов.

1.2.1 Классическое определение

В 1946 году, голландский математик Николас де Брюин занимался проблемой поиска минимальной циклической подстроки, содержащей все возможные двоичные последовательности длины l в качестве подстрок.

Он рассматривал особый класс графов, определяемый следующим образом: рассмотрим $n-$буквенный алфавит и некоторое фиксированное число l. Сформируем множество всех возможных строк длины $l-1$ над этим алфавитом. Де Брюин рассмотрел граф $B(n,l)$, в котором в качестве вершин стоят все элементы построенного множества, а две вершины v_1, v_2 соединены направленным ребром тогда и только тогда, когда существует слово длины l, чьим префиксом является строка из v_1, а суффиксом строка из v_2. Граф содержит n^{l-1} вершин. Нетрудно также заметить, что входящая и исходящая степень любой вершины этого графа равна n.

1.2.2 Граф де Брюина для идеального множества ридов

Построим подобную конструкцию по множеству ридов и выясним чему соответствует исходный геном в получившемся графе.

Но, для начала, упростим ситуацию, предположив, что множество наших ридов S *идеально*:

- Все риды имеют одну и ту же длину l.
- Все возможные подстроки генома длины l встречаются в множестве ридов.
- Все риды соответствуют одному стренду (далее в этом разделе слова геном и стренд являются синонимами).
- Риды были сгенерированы без ошибок.

Также предположим пока, что в восстанавливаемом геноме нет повторяющихся фрагментов длины l.

Граф де Брюина для такого множества ридов определим следующим образом.

Рассмотрим все различные префиксы и суффиксы ридов длины $l-1$. Построим граф D, в котором они являются вершинами и соединим две вершины v_1 и v_2 направленным ребром тогда и только тогда, когда существует рид с префиксом v_1 и суффиксом v_2. Каждый исходный рид соответствует ребру полученного графа.

Ясно, что получившийся граф D является подграфом $B(4, l)$, так как получился тем же методом построения просто по меньшему множеству строк.

Каждый путь в графе соответствует некоторой строке. Начинаем с $k-1$-мера, соответствующего первой вершине пути. Проходя по каждому ребру, приписываем последний нуклеотид соответствующего k-мера справа к имеющейся строке. В итоге получится строка длины $k-1+$ (количество ребер пути).

Нетрудно заметить, что в условии сделанных предположений, стренд соответствует некоторому *эйлеровому* пути в графе D. Эйлеров путь — путь, проходящий по каждому ребру ровно по одному разу. Также ясно, все эйлеровы пути в полученном соответствует строкам одинаковой длины, каждая из которых является решением задачи о кратчайшей надстроке (SSP, [3]) для множества S.

Замечание. Для нахождения эйлерова пути в графе есть простой линейный алгоритм. Получается, что нам удалось решить задачу SSP за полиномиальное время! Но сделать это нам удалось лишь за счет очень сильного дополнительного условия на множество входных строк: оно содержит все подстроки длины l исходной строки.

Попробуем для начала отбросить единственное предположение, которое мы сделали про сам геном: предположение об уникальности всех l-меров в геноме. Как уже отмечалось, в геноме присутствует большое количество повторяющихся участков, так что это условие заведомо не выполняется.

Если бы нам помимо множества ридов были бы известны кратности соответствующих l-меров в геноме, то мы бы просто добавить в граф ребра соответствующей кратности. Нетрудно заметить, что при этом исходный геном продолжал бы соответствовать некоторому эйлеровому пути в графе. На практике, получить информацию о точной кратности конкретного l-мера в геноме оказывается невозможно. Таким образом, ввести в граф правильную кратность ребер не получится, поэтому вообще не будем использовать кратные ребра. Это, конечно, еще не повод отказываться от выбранного подхода, просто нужно понимать, что в полученном графе геном больше не соответствует эйлеровому пути.

Если нет никакой дополнительной информации, то разумно вновь ис-

кать геном среди решений задачи SSP для множества S. Все они в точности соответствуют кратчайшим путям, проходящим по каждому из ребер графа не менее одного раза. Задача поиска таких путей называется *Задачей Китайского Почтальона* (Chinese Postman Problem). Все ее решения являются кандидатами в геном, но у нас не хватает инфорамции, чтобы отдать предпочтение кому-либо из них.

Есть один нюанс. В общем случае, задача Китайского почтальона тоже NP-полна! Если бы мы рассматривали парные риды и пытались отбросить из рассмотрения пути, не согласованные с их внутренними расстояниями, то это только усложнило бы задачу. Получилось, что даже максимально упростив ситуацию, мы вновь пришли к труднорешаемой задаче.

Но на самом деле все не так плохо и полученный граф может принести нам пользу. Предположим, что мы нашли некоторый путь, который гарантированно присутствует в любом решении задачи китайского почтальона (например, таким является любой путь, внутренние вершины которого имеют исходящую степень 1). Тогда строка, соответствующая этому пути гарантированно присутствует в искомом геноме.

1.2.3 Граф де Брюина для множества ридов

К сожалению, описанный выше способ построения графа по идеальному множеству ридов на практике не приведет ни к чему хорошему. Что естественно, ведь множество настоящих ридов обычно не удовлетворяет ни одному из условий "идеальности", введенных в прошлом разделе.

Начнем с того, что риды могут иметь имеют неодинаковую длину. Но даже если бы это было так, то невыполнение второго условия приведет к тому, что граф "распадется" на большое количество компонент связности. Действительно, отсутствие в множестве ридов некоторого l-мера s, присутствующего в геноме, приведет к "разрыву" в графе де Брюина пути, соответствующего любому фрагменту генома, содержащему s.

Неожиданное решение этих проблем в том, чтобы извлечь из ридов, на чью малую длину мы постоянно сетовали, еще более короткие подстроки длины k, по которым уже построить граф де Брюина.

Итак, опишем построение финального варианта графа более формально. Пусть дано множество ридов $S = s_1, \ldots, s_n$. Через S_k обозначим мно-

жество всех k-меров, встречающихся в качестве подстрок по крайней мере одного рида из множества S. Выберем некоторой k и определим граф де Брюина $D(S_k)$ следующим образом: все элементы множества S_{k-1} являются вершинами графа, $k-1$-мер v соединен с $k-1$-мером w направленным ребром, если S_k содержит k-мер, у которого первые $k-1$ нуклеотид совпадают с v, а последние $k-1$ — с w. Таким образом каждый k-мер из S_k соответствует ребру графа $D(S_k)$.

Если строить граф таким образом, то связность нарушается только в том случае, если некоторый k-мер генома вовсе не встречается в ридах. Выбрав достаточно маленькое k, можно снизить вероятность этого события практически до нуля. Но при этом не стоит забывать, что чем меньше k, тем больше путей, соответствующих различным участкам генома будут пересекаться в графе из-за наличия в них одинакового $k-1$-мера. Обычно выбирают $20 < k < 40$ нуклеотидов, конкретное значение прежде всего должно зависеть от длины ридов, а также кратности и равномерности покрытия.

Попробуем отказаться от предполоения о том, что риды принадлежат одному и тому же стренду. На практике, в большинстве случаев, в процессе получения ридов, секвенатор действительно "перемешивает" данные, полученные с различных стрендов. В силу сопряженной структуры ДНК, можно предварительно дополнить S сопряженными (реверс-комплементарными) ридами. Тогда построенный граф де Брюина будет представлять из себя как бы "склейку" графов, соответствующих каждому из двух стрендов. Теперь путь, соответствующий стренду не должен проходить по всем ребрам.

Опианный граф де Брюина обладает очень важным свойством, делающим его столь полезным для сборки геномов из коротких ридов, обеспечивающих высокое покрытие генома: его компактность. Действительно, количество вершин и ребер графа линейно относительно длины генома и не зависит от количества ридов.

Теперь обсудим последнее из сделанных ранее предположений: безошибочность ридов. На самом деле в ридах присутствует большое количество ошибок различных типов. Из-за них, в графе появляется большое количество вершин и ребер, не соответствующих никакому из участков генома. Их наличие, с одной стороны усложняет дальнейшую работу с графом,

запутывая его структуру, а с другой — может привести к появлению соответствующих ошибочных участков в итоговой сборке. Так как размер графа де Брюина зависит от количества ошибок в ридах, то оказывается, что он все-таки зависит от количества самих ридов. Но при использовании процедур исправления ошибок, влияние этой зависимости на размер графа не оказывается фатальным.

Теперь достаточно трудно формально описать свойства, которым должен удовлетворять путь, соответствующий некоторому стренду. И тем не менее, не все так плохо. Раньше ведь мы знали ограничения и все-равно не имели полиномиальных алгоритмов поиска путей им удовлетворяющим.

Представим себе, что нам удалось удалить все ошибочные ребра из графа. Тогда, к примеру, как и раньше любой путь без ответвлений будет гарантированно соответствовать участку некоторого стренда.

1.2.4 Сжатый граф де Брюина

Альтернативным, но во многих смыслах более удобным способом представления той же информации, что и граф де Брюина, является *сжатый граф де Брюина*.

Сжатый граф де Брюина — это граф, получаемый из графа де Брюина в результате применения следующего преобразования сжатия неразветвленных участков:

while В графе есть вершина v, входящая и исходящая степени которой равны 1 **do**

Заменим ребра, инцидентные v одним ребром, помеченным последовательностью, считанной по пути из удаляемых ребер

Удалим v из графа

end while

Множество вершин полученного графа, является подмножеством вершин исходного графа ($k-1$- меров). Ребра соответствуют последовательностям нуклеотидов различной длины (не короче k). При этом последовательности на ребрах согласованы с последовательностями в вершинах (последовательность в ребре начинается и заканчивается с $k-1$-меров из соответствующих вершин).

Последовательность, соответствующая пути в сжатом графе определя-

ется также как и в несжатом, с той лишь разницей, что, проходя по ребру, мы добавляем в конец все нуклеотиды ребра, начиная с k-го.

Длиной ребра e ($length(e)$) будем называть количество k-меров, содержащихся в этом ребре. То есть $length(e) =$ (количество нуклеотидов в ребре) $-(k-1)$. При таком определении, до сжатия графа все ребра имеют единичную длину.

В дальнейшем, если не сделано специальных оговорок под графом де Брюина будет пониматься именно его сжатая версия.

1.3 Общая схема работы ассемблера, основанного на графах де Брюина

Общая схема работы современных ассемблеров, основанных на графах де Брюина:

- Предварительное исправление ошибок
- Построение графа де Брюина
- Коррекция графа
- Разрешение повторов
- Получение контигов и скэффолдов
- Этап консенсуса

Вкратце опишем что происходит на каждом из этих этапов.

1.3.1 Предварительное исправление ошибок

Как уже неоднократно отмечалось, в ридах присутствует большое количество ошибок. К примеру, для типичных данных, полученных секвенатором фирмы Illumina ошибки присутствуют в среднем в кажом третьем риде (если точнее, в 32% ридов). Если ничего не предпринять, то мы получим граф, содержащий большое количество ошибочных элементов. К примеру, в проекте по секвенированию *N.meningitidis* ([9]) несжатый граф, построенный по эмулированным данным состоял из 4,039,248 вершин, в то

время как граф, построенный по реальным данным (для 20-меров), содержал 9,474,411 вершину.

Все современные ассемблеры производят этап предварительной коррекции ошибок.

Обычно, в начале, у ридов обрезаются концы низкого качества (на основе Phred коэффициентов).

Затем все подходы используют следующую идеею: каждая подстрока генома фиксированной длины (обычно, порядка 20-30 нуклеотидов), должна встречаться в ридах большое количество раз. Задача состоит в том, чтобы внести в риды в некотором смысле наименьшее (с учетом весовых PHRED коэффициентов) количество исправлений таким образом, чтобы все присутствующие в них подстроки, встречались бы в них достаточное (порог зависит от покрытия) количество раз.

Эти техники устраняют только ошибки вида “замена одного нуклеотида на другой”. Но учитывая, для современных секвенаторов такие ошибки составляют абсолютное большинство, то процедуры устранения ошибок способны устранить до 98% всех ошибок, находящихся в ридах.

Так, в уже упомянутом проекте по секвенированию *N.meningitidis* удалось снизить количество вершин до 4,081,857.

1.3.2 Построение графа де Брюина

Почти все популярные ассемблеры в явном виде строят некоторый сжатый граф де Брюина.

1.3.3 Коррекция графа

Большая часть ошибок устраняется на этапе предварительного исправления (этап 1), но даже относительно небольшая часть ошибок, оставшаяся во входных данных приводит к тому, что количество ребер сжатого графа возрастает в десятки раз.

Для того, чтобы сделать результат сборки более качественным, предпринимаются попытки устранения из графа участков, соответствующих ошибкам входных данных.

1.3.4 Разрешение повторов

Как уже было отмечено, в числе всех трудностей, значительно усложняющих задачу ассемблирования, особое значение играет наличие в геноме так называемых повторов. Это идентичные (или имеющие малое редакционное расстояние) участки генома, повторяющиеся 2 и более раз. Повторы имеют свою классификацию, которая не будет нас интересовать. Более 50% длины генома млекопитающих занимают повторы различных типов. Это обусловлено биологическими причинами: не до конца изученными преобразованиями, за счет которых происходит эволюция генома.

В контексте графа де Брюина, построенного по k-мерам, *повторами*, будем считать, все последовательности нуклеотидов длины $\geq k-1$, присутствующие в геноме более одного раза (быть может, на разных стрендах). Ясно, что каждый повтор в геноме "запутывает" графа де Брюина, склеивая участки графа, соответствующие различным участкам генома. Эти участки графа будем

При разрешении повтора, происходит "расклеивание" некоторых ребер графа так, что из графа исчезают пути, которые не соответствуют никаким участкам генома.

Рассмотрим простой пример: представим себе два участка генома, содержащих повтор. Наличие повтора приводит к тому, что пути в сжатом графе, соответствующие этим участкам "накладываются" друг на друга (см. рис.1а). Через ребро, соответствующее самому повтору при этом проходит уже 4 различных пути, лишь 2 из которых соответствуют исходным участкам.

Рис. 1: Иллюстрация разрешения повтора

Разрешение повторов — это этап, на котором происходит попытка обнаружения "истинных" путей в графе, проходящих через повторы, или, если немного переформулировать, связывание входов в повтор с выходами

из него. Затем происходит разделение повтора, при котором создается по копии повтора для каждого истинного пути. (см. рис.1 б)

За счет чего можно отличить истинные пути?

Вернемся к нашему примеру. Рассмотрим риды, проходящие через участок графа с рис. 1а. Предположим, что есть риды, проходящие одновременно по обоим красным ребрам и есть риды, проходящие по обоим зеленым ребрам, в то время как ридов, идущих через красное и зеленое ребро не существует. Тогда мы можем предположить, что "перекрестные" пути являются побочными.

Ясно, что подобные рассуждения могут принести плоды только в случае, если длина повтора меньше длины ридов (в противном случае по ним нельзя сделать вывод о связях между входами и выходами из повтора). Чем длиннее риды, тем больше повторов мы сможем разрешить. К сожалению, нам приходится работать с достаточно короткими ридами.

На помощь приходит то, что мы имеем дело с парными ридами. Вновь рассмотрим ситуацию на рис. 1а, но в этот раз будем считать, что длина повтора превышает длину рида. При этом, если есть парные риды, ложащиеся обоими концами на красные ребра и есть — ложащиеся на зеленые, но нет парных ридов, одним концом ложащихся на зеленое, а другим — на красное ребро, то мы вновь можем разрешить этот повтор

Общий итог такой: при использовании парных ридов у нас появляется возможность разрешить повторы, длина которых не превышает длину их внутреннего расстояния.

Важно понимать, что в ходе разрешения повторов, граф перестает быть графом де Брюина, так как один и тот же k-мер теперь может присутствовать более графе более одного раза.

1.3.5 Получение контигов и скэффолдов

Контиги и скэффолды — конечный результат сборки.

Контигами (contigs) называют последовательности нуклеотидов, выдаваемые ассемблером в качестве участков генома.

Наиболее простая и популярная стратегия получения контигов по графу заключается в том, чтобы взять в качестве контигов последовательности, соответствующие отдельным ребрам графа.

При этом про структуру графа обычно полностью забывают и больше ее не используют.

Скэффолдами (scaffolds) называют линейные структуры из объединения контигов.

Процесс получения скэффолдов включает в себя:

- установление кратности контигов
- установление порядка контигов
- вычисление длин зазоров между контигами

Установление кратности контигов происходит, в основном по информации о покрытии соответствующих участков графа.

Ключевую роль при получении скэффолдов играет информация о расстояниях, полученная из парных ридов.

1.3.6 Этап консенсуса

Это финальный этап сборки, на котором происходит прикладывание исходных ридов к полученным контигам и исправление ошибок, которые могли быть внесены на предыдущих этапах.

1.4 Постановка задачи

1.4.1 Разрабатываемый ассемблер

В Проблемной лаборатории вычислительной биологии АУ РАН [2], под руководством Павла Певзнера [3] и Максима Алексеева [4], с февраля 2011 года, ведется разработка нового ассемблера, использующего подход, основанный на графе де Брюина.

В силу достаточно жестких требований к производительности и потребляемой памяти, разработку ассемблера было решено вести на языке C++.

Этот проект нацелен сразу на два направления, еще не охваченные существующими продуктами.

[2] bioinf.spbau.ru

[3] Professor, University of California at San Diego

[4] Assistant professor, University of South Carolina

- *Работа с данными секвенирования, полученными по единственной клетке.*

  Значительная часть видов бактерий не поддается клонированию в условиях лаборатории. Раньше секвенировать такие бактерии было невозможно, но сравнительно недавно появились технологии, позволяющие получать достаточное для сборки количество ридов по нескольким или даже всего лишь одной клетке (single cell sequencing technologies). К сожалению, использование этих технологий приводит к получению ридов, крайне неравномерно покрывающих геном (разброс кратности покрытия может составлять десятки тысяч). С такими входными данными не справляется ни один из существующих ассемблеров.

  Для того, чтобы ассемблировать риды, полученные по single cell технологиям секвенирования, необходимо разработать принципиально новые подходы к обработке графа де Брюина.

- *Сборке больших геномов с использованием "бесконтекстного" подхода.*

  На данный момент, наиболее узкое место всех современных ассемблеров, нацеленных на сборку больших геномов (прежде всего, геномов млекопитающих) — это размер потребляемой оперативной памяти. К примеру, для *de novo* сборки человеческого генома по коротким ридам ассемблеру SOAP *de novo*, требуется супер-компьютер с 512Gb оперативной памяти.

  Одним из наиболее перспективных подходов к сборке больших геномов, является так называемый "бесконтекстный" подход. Основная его идея в том, чтобы попробовать отказаться от хранения и использования конкретных последовательностей нуклеотидов при построении и анализе графа де Брюина (формально, используемый там граф не является графом де Брюина, но очень похож на него по топологической структуре).

  Ни один из существующих ассемблеров данный подход не использует. По нашим предварительным оценкам, его использование позволит снизить требования к потребляемой оперативной памяти в де-

сятки раз и позволит перенести сборку больших геномов с суперкомпьютеров с сотнями гигабайт оперативной памяти (который, кстати говоря, есть не в любой исследовательской лаборатории) на мощные сервера.

Являясь сотрудником упомянутой лаборатории, я принимаю активное участие в разработке нового ассемблера.

Было решено разбить проект на три фазы.

- Реализовать хороший ассемблер для бактериальных геномов, использующий известные и не раз опробованные методы, а также новый подход к разрешению повторов.

- Реализовать бесконтекстный подход и приспособить ассемблер к сборке генома млекопитающих.

- Детально исследовать проблемы, возникающие при сборке по данным от single cell sequencing и разработать методы их преодоления.

Пока проект находится в первой фазе, и наш ближайший планы — разработать просто хороший ассемблер для бактериальных геномов. Но при этом было принято решения сразу же проектировать ассемблер максимально гибко, чтобы при переходе к следующей фазе проекта не пришлось бы переделывать все с нуля.

1.4.2 Цели работы

В рамках проекта, я занимался разработкой библиотеки для построения и обработки сжатого графа де Брюина.

Результатом работы должна была стать библиотека, послужащая базой для для нового ассемблера.

Основными целями моей работы были:

- *Разработка удобной, расширяемой библиотеки для работы с графом де Брюина.*

  Библиотека должна предоставлять удобные средства для разработки новых алгоритмов обработки и анализа графа.

- *Разработка алгоритмов обнаружения и устранения ошибочных участков графа.*

  В рамках нашего проекта, важным требованием к этим алгоритмам, является наличие режима работы, не использующего конкретные последовательности нуклеотидов. Это нужно для того, чтобы их можно было применить к графу, реализующему “бесконтекстный” подход.

2 Анализ предметной области

2.1 Классификация основных типов ошибочных участков графа де Брюина

Как уже не раз упоминалось, во входных данных, присутствует большое количество ошибок.

Основные типы ошибок:

- *Замена* отдельного нуклеотида на другой.
- *Вставка/удаление* нуклеотида.
- *Химерные риды* — риды, составленные из различных участков генома, разделенных неизвестным расстоянием.

В зависимости от характера ошибки в риде, она может привести к образованию ошибочных участков различного вида ([25, 9]).

- Если ошибка в риде произошла в первом/последнем $k-1$ нуклеотиде, то в графе появится короткое ответвление, с одного из концов "отделенное от графа". Участок графа такого вида называют *тупиком* (tip).

  Если ошибка одна, то длина соответствующего тупика не превосходит k. Но так как ошибок в одном риде может быть несколько, то длина тупика может достигать длины рида.

- Если ошибка произошла где-то в середине рида, то в графе появится короткий участок, представляющий альтернативный (приблизительно той же длины) путь между двумя вершинами графа (короткий ненаправленный цикл). Участок графа такого вида называют *пузырем* (bulge, bubble). Одиночная ошибка, приводит к образованию ошибочного пути длины k, но так как ошибок в одном риде может быть несколько, то длина пузыря может достигать длины рида.

- Химерному риду будет соответствовать короткий (длины порядка k) путь, соединяющий два "далеких" друг от друга участка графа

(участки, скорее всего, ранее не связанные коротким путем).Такие пути называют *ошибочными соединениями* (erroneous connections).

Важное замечание. Только из того, что участок имеет вид, похожий на тупик или пузырь, еще не означает, что он соответствует ошибке! Настоящие ребра, являющиеся тупиками, могут появляться из-за разрыва в покрытии генома. Пузырь может соответствовать неидеальному повтору (их в геноме великое множество).

Итак, одного вида окрестности недостаточно для того, чтобы принять решение об ошибочности участка. Обычно, при решении об удалении того или иного участка, ассемблеры пользуются:

- величиной среднего покрытия ребер
- последовательностями нуклеотидов ребер

2.2 Обзор существующих ассемблеров

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

Далее приведен обзор нескольких (наиболее популярных) ассемблеров, основанных на графах де Брюина. Примеры ассемблеров, использующих другие подходы не приводятся, из-за того, что как было отмечено ранее, они не могут работать на интересующих нас входных данных.

В связи с решаемой в данной работе задачей, особое внимание уделяется способам представления графа и устранения ошибочных участков, используемых в этих ассемблерах.

2.2.1 Euler

Первая версия разработана в 2000 году под руководством Павла Певзнера. Это первый ассемблер, использовавший подход, основанный на графах де Брюина. [19, 20] В то время произвел настоящую революцию и доказал возможность качественной сборки по коротким ридам.

За 10 лет было разработано большое количество версий этого ассемблера (Euler, Euler-DB, Euler+, Euler-SR), но в последние несколько лет, про-

ект развивается не слишком активно. Текущая версия ассемблера, Euler-SR, вышла в 2008 году ([9]).

Алгоритмы коррекции работают на сжатом графе де Брюина.

Все короткие ребра, один конец которых "отсоединен" от графа, удаляются как тупики.

Пузыри удаляются сложным методом, который в начале строит максимальное ветвление (maximum branching) графа и только затем устраняет короткие ненаправленные циклы, внимательно следя за тем, чтобы не удалить направленные циклы.

Все ребра с небольшим покрытием удаляются как ошибочные.

2.2.2 Velvet

Разработан Дэниэлем Дзербино ([5]).

Изначально являлся набором инструментов для построения и упрощения графа де Брюина, построенного по ультра-коротким ридам ([25]). Позднее были добавлен модуль разрешения повторов с использованием парных или длинных одиночных ридов ([24]).

Использует оригинальное представление сжатого графа, в котором однозначные участки графа сжимаются не в длинные ребра, а длинные вершины.

Velvet также характерен тем, что он, в отличие, скажем, от Euler, в процессе коррекции графа уделяет много внимания поддержанию согласованности сопряженных элементов.

Тупик удаляется только если он имеют длину $< 2k$ (достаточно распространенным среди ассемблеров пороговое значение), а также через место его "крепления" к графу проходит более покрытый путь.

Алгоритм удаления пузырей был назван Tour Bus. Схема его работы следующая. От произвольной вершины графа запускается алгоритм Дейкстры. В качестве весов при работе алгоритма рассматривается длина, деленная на значение покрытия. Таким образом алгоритм будет отдавать предпочтение путям с большим покрытием. Если алгоритм приходит в ранее посещенную вершину a, то он делает шаг на одну вершину и оказывается в b. После чего происходит поиск общего предка вершин a и b по отношению к рассмотренным путям. Если предок c был найден не слишком далеко, то

алгоритм анализирует похожесть последовательностей нуклеотидов, соответствующих путям $c \to a$ и $c \to b$. И если они оказываются похожи, то он запускает сложную процедуру слияния путей, не нарушающую согласованность сопряженных элементов графа, отдавая предпочтение тому пути, который был найден первым (более покрытому). Слияния не происходит, если оба пути имеют большое покрытие.

Сложность алгоритма Tour Bus составляет $O(N * logN)$, где N - количество вершин графа.

Все короткие участки с небольшим покрытием удаляются как ошибочные. Важно, что этот шаг происходит после устранения тупиков и пузырей, когда большая часть таких участков, соответствовавшая реальным участкам генома, была "включена" в длинные вершины, в которых информация о покрытии усредняется.

2.2.3 ALLPATHS

Этот ассемблер является единственным из перечисленных здесь, который в явном виде не строит граф де Брюина целиком ([6, 16]).

Вместо этого он некоторым образом "локализует" риды, принадлежащие одному и тому же региону, после чего анализирует эти регионы отдельно друг от друга (при этом может делать это параллельно). Затем происходит этап "склеивания" локальных графов в единый граф. И только после этого происходит коррекция итогового графа.

На этапе корректирования не удаляет пузыри, удаляются только тупики.

Ошибочные соединения отдельно не удаляются, но из-за особенностей алгоритма, в итоговую сборку и так не попадут.

ALLPATHS, активно использует информацию о парных ридах. Более того, необходимым условием для использования этого ассемблера является наличие нескольких библиотек парных ридов со значительно отличающимся внутренним растоянием (в оригинальной статье использовались одновременно библиотеки с внутренним расстояниями 50 тыс, 6 тыс и 5 нуклеотидов).

2.2.4 ABySS

Ассемблер, использующий распределенное хранение несжатого графа де Брюина, при котором информация о различных k-мерах распределена по различным физическим машинах ([23, 1]). Позволяет осуществлять сборку больших геномов, используя сравнительно более дешевые компьютеры, объединенные в кластер (при этом из-за необходимости обмена сообщениями, скорость сборки в десятки раз ниже, чем, скажем, у SOAP *de novo*).

Один из немногих ассемблеров, которому удалось осуществить *de novo* сборку человеческого генома по NGS данным. Для этого использовался 21 компьютер, всего лишь с 16GB каждый.

Все алгоритмы коррекции работают на распределенном несжатом графе и распараллелены. Сжатый граф также строится, но уже после всех алгоритмов коррекции.

2.2.5 SOAP *de novo*

Пожалуй, наиболее "продвинутый" из всех ассемблеров. Активно разрабатывается в Китае большой группой исследователей ([4]).

Нацелен в первую очередь на сборку больших геномов ([15]).

Второй (помимо ABySS) ассемблер, способный за разумное время осуществлять *de novo* сборку генома млекопитающих. Значительно выигрывает у ABySS в скорости, но требует для этого супер-компьютер с сотнями гигабайт оперативной памяти.

Недавно сего помощью была осуществлена *de novo* сборка панды — первого млекопитающего, собранного исключительно по коротким ридам.

По словам самих авторов, алгоритмы коррекции очень похожи на используемые в Velvet.

3 Предлагаемое решение

3.1 Построение графа де Брюина

Из-за сопряженной структуры ДНК, каждый парный рид корректнее представлять себе как два сопряженных парных рида. Для того чтобы учесть эту специфику, явно добавим сопряженные парные риды к множеству входных пар.

Построение несжатого графа де Брюина достаточно прямолинейно.

Заранее выберем **четную** константу k, она останется неизменной до самого конца.

Проходом по всем ридам, извлекаем все k-меры, присутствующие в каждом из них и складываем их в эффективную по структуру хранения.

Для каждого $k-1$-мера (вершины) и k-мера (ребра) в графе найдется сопряженный. Это в частности означает, что для каждого пути в графе найдется сопряженный ему путь.

Замечание: Так как выбранное k является четным, то никакой $(k-1)$-мер не может оказаться самосопряженным. Это означает, что все вершины разбиваются на сопряженные пары. При этом самосопряженные ребра могут присутствовать в графе. Ясно, что ребро является самосопряженным т. и т.т., когда оно соединяет пару сопряженных вершин.

3.2 Построение сжатого графа

Простым будем называть путь в графе, входящая и исходящая степень всех внутренние вершин которого равны 1.

Операцию замены простого пути ребром, содержащим последовательность ранее соответствовавшую этому пути, будем называть *конкатенацией* ребер этого пути.

Замечание. Так как в качестве длины ребра e мы рассматриваем количество k-меров, принадлежащих ему, то она оказывается аддитивна относительно операции конкатенации. $e = concat(p_1, p_2, \ldots, p_n) \Rightarrow length(e) = \sum_{i=1}^{n} length(p_i)$

Помимо самого графа, нам будет необходим *индекс* k-меров в этом графе. Это структура, позволяющая для каждого k-мера получить иденти-

фикатор ребра, в котором находится данный k-мер и его позицию относительно начала ребра (количество нуклеотидов, которые нужно пропустить, чтобы его считать).

При условии, что последовательности нуклеотидов хранятся в простых линейных структурах (например, массивах), строить сжатый граф напрямую по псевдокоду алгоритма может оказаться крайне неэффективно. Действительно, ведь если объединение двух последовательностей происходит за время O(их длины), то построение длинного ребра сжатого графа может занять квадратичное от его длины время.

Предположим, что у нас имеется несжатый граф, тогда по следующему алгоритму можно эффективно построить сжатый граф вместе с заполнением индекса.

Алгоритм 1 Алгоритм сжатия графа

Вход: . D_k — граф де Брюина
Выход: C_k — сжатый граф де Брюина, I — индекс по k-мерам.

for s — k-мер, присутствующий в D_k **do**
  if s отсутствует в индексе **then**
    Пройти от него влево по несжатому графу пока не встретим развилку (вершину у которой входящая или исходящая степень > 1) или тупик (вершину исходящей степени 0)
    От найденной позиции пройти направо до ближайшей развилки или тупика, считывая пройденные ребра в последовательность нуклеотидов.
    Найти (с помощью индекса) в уже поенной части C_k, соответствующие началу и концу последовательности, по необходимости создав их.
    Добавить в C_k ребро между этими вершинами, соответствующее последовательности и внести позиции соответствующих k-меров в I.
  end if
end for

3.3 Графовые операции

Определим операции преобразования графа, которые наиболее естественным образом позволят нам выразить алгоритмы упрощения графа.

Их условно можно разделить на операции низкого и высокого уровней.

К операциям низкого уровня относятся операции добавления/удаления ребра/вершины.

При каждом добавлении/удалении ребра необходимо преобразование индекса.

Операции высокого уровня это:

- *Конкатенация* ребер простого пути. *concat(P)*

  Описывалась ранее. Можно было ввести конкатенация двух ребер и выразить рассматриваемую операцию через нее, но в некоторых случаях, реализация последней может оказаться существенно эффективнее, так как сжатие длинного пути P по парам ребер может занять $O(length(P)^2)$ времени ($length(P)$ —суммарная длина всех ребер, входящих в путь).

- *Разделение* ребра e в заданном отношении α. Ребро e удаляется из графа. Появляются ребра $e_1, e_2 : e = concat(e_1, e_2) \& l_{e_1}/l_{e_2} = \alpha$ и вершина их соединяющая.

- *Проекция* ребра e_1 на ребро e_2. Семантика данной операции заключается в переносе всей информации с ребра e_1 на ребро e_2. Ребро e_1 удаляется из графа. Если при этом появляется висячая вершина, то она также удаляется из графа.

Операции высокого уровня реализованы через операции низкого уровня.

3.4 Сжатый граф, учитывающий сопряженность

По аналогии с несжатым графом, в сжатом графе, после его построения, каждому элементу соответствует сопряженный.

Это естественное ограничение (свойство) пришло из сопряженной структуры ДНК (и соответствующей обработки входных данных). Но мы собираемся преобразовывать граф, при этом это ограничение, вообще говоря может оказаться нарушенным.

Замечание: сопряженные вершины сжатого графа могут быть соединены как одним самосопряженным, так и двумя сопряженными (но не самосопряженными) ребрами.

Некоторые ассемблеры явно не поддерживают согласованность сопряженных путей при выполнении преобразований графа, надеясь на то, что все преобразования пройдут симметрично для участков графа, соответствующих разным стрендам. Но если надежда не оправдывается (а это запросто может произойти, так как зачастую одни и те же последовательности присутствуют на обоих стрендах), то полученный граф окажется в противоречивом состоянии, что может привести к ошибкам в итоговой сборке.

Мы будем использовать структуру сжатого графа, явно учитывающую сопряженность.

Можно представлять себе, что в графе все вершины и ребра "сдвоенные". Верхняя и нижняя половины вершин соответствуют сопряженным $(k-1)$ мерам (самосопряженных $(k-1)$-меров у нас не бывает из-за четности k). Если $e : v_1 \to v_2$, то $e' : v_2' \to v_1'$.

Вершину, сопряженную к вершине v будем обозначать $conj(v)$ (от англ. conjugate). Аналогично и для ребер.

Будем считать, что у нас есть способ эффективно переходить с элемента графа на его сопряженный и реализуем все операции, предоставляемые графом таким образом, чтобы они сразу же влияли бы и на сопряженные элементы.

При добавлении/удалении вершины/ребра, будем сразу добавлять/удалять из графа сопряженный элемент. Единственная тонкость заключается в обработке самосопряженных ребер. При добавлении их нужно добавлять в единственном экземпляре, а при удалении не пытаться удалить их дважды.

За счет такой модификации операция разделения ребра автоматически начинает корректно обрабатывать сопряженные элементы.

Операция проекции также будет работать корректно, с тем лишь естественным ограничением, что запрещается проецировать ребро на сопряженное.

В общем случае, при конкатенации простого пути ребер, будет автоматически корректно сжиматься и комплементарный путь.

Единственная опасность заключается в следующем: если последняя вершина v_{last} в пути является сопряженной некоторой его внутренней вершине v $(conj(v) = v_{last})$, то действуя по общей схеме, мы бы добавили ребро

между v_{first} и v_{last}, а затем, удалив v, удалили бы и v_{last} вместе с только что добавленным ребром. Если рассмотреть ситуацию подробнее, то можно доказать , что она возможна только когда рассматриваемый простой путь содержит самосопряженное ребро.

Итак, случай присутствия в пути самосопряженного ребра, является особым. Пусть $P = \{e_i\}_{i=1}^{n}$, e_k — самосопряженное, и $k > n/2$ (случай $k < n/2$ разбирается аналогично). Ясно, что для $i = 1 \ldots k-1$ $conj(e_{k-i}) = e_{k+i}$. Нужно дополнить путь недостающими сопряженными ребрами, после чего обработать его как обычно (добавить в граф новое самосопряженное ребро $e = concat(e_1, \ldots e_{k-1}, e_k, conj(e_{k-1}), \ldots, conj(e_1))$, удалить из графа все внутренние вершины пути вместе с инцидентными им ребрами (при этом удалятся и их сопряженные).

3.5 Дополнительная информация

Помимо самих последовательностей, с ребрами полезно ассоциировать еще несколько видов информации: уже упоминавшуюся информацию о среднем покрытии ребра и информацию о межреберных расстояниях (см. далее). Различные ассемблеры используют различные политики работы с этими видами информации. Одна из проблем заключается в том, что чтобы их вычислить, нужно знать какому участку графа соответствует каждый из исходных ридов (процесс выяснения этих участков называется *прикладыванием* ридов). Но в процессе преобразований из графа исчезают ребра и многие риды перестают точно прикладываться к графу. Таким образом, прикладывание ридов к преобразованному графу может оказаться непростой задачей, а информация, полученная в ходе этого процесса может оказаться не полной.

В данной работе предлагается следующая стратегия для работы с любым типом дополнительной информации.

- первоначальный подсчет на графе до преобразований
- изменение информации при преобразованиях графа в соответствии с семантикой операций, описанных ранее

Замечание. Преобразование дополнительной информации, соответствующей сопряженным элементам, нужно производить одновременно и сим-

метрично, для сохранения ее непротиворечивости. Везде далее считается, что так и происходит (о том как это реализовано см. раздел 4.2).

Преимущества такого подхода:

- прикладывание ридов осуществляется очевидным образом с помощью индекса k-меров
- практически не происходит потери информации в процессе преобразований
- появляется возможность использовать дополнительную информацию в процессе преобразований графа для разрешения спорных ситуаций

3.6 Покрытие

Информация о среднем покрытии нужна будет нам в первую очередь для удаления из графа ошибочных ребер. Также она важна для определения приблизительной кратности контигов при скэффолдинге.

3.6.1 Первоначальный подсчет

Обычно под покрытием некоторого участка генома подразумевают среднее по всем нуклеотидам количество ридов, покрывающих этот нуклеотид.

Для нас будет удобно немного модифицированное понятие покрытия: мы будем рассматривать среднее покрытие k-меров ребра, то есть среднее по всем k-мерам количество ридов, покрывающих этот k-мер ($coverage(e)$).

Модель подсчета такова: разбить рид на множество k-меров, учесть каждый из них в покрытии соответствующего ребра.

Непосредственно подсчет происходит путем прикладывания одиночных ридов к сжатому графу.

Для каждого ребра e, по которому прошел рид r, увеличим его покрытие на количество k-меров в перекрытии r и e, в конце добавляем нормировку на $1/length(e)$.

Замечание: так как множество ридов было дополнено сопряженными, то покрытие сопряженных ребер автоматически окажется одинаковым.

3.6.2 Правила преобразования

При конкатенации простого пути покрытие нового ребра получается как взвешенная сумма покрытий склеиваемых ребер. Если $e = concat(e_1, e_2)$, то $coverage(e) = (coverage(e_1) * length(e_1) + coverage(e_2) * length(e_2))/length(concat$

При разделении ребра, покрытия обоих итоговых ребер будем считать равными исходному покрытию (здесь происходит потеря некоторой информации).

При проекции ребра e_1 на e_2, покрытие e_2 увеличивается на покрытие e_1.

3.7 Информация о межреберных расстояниях

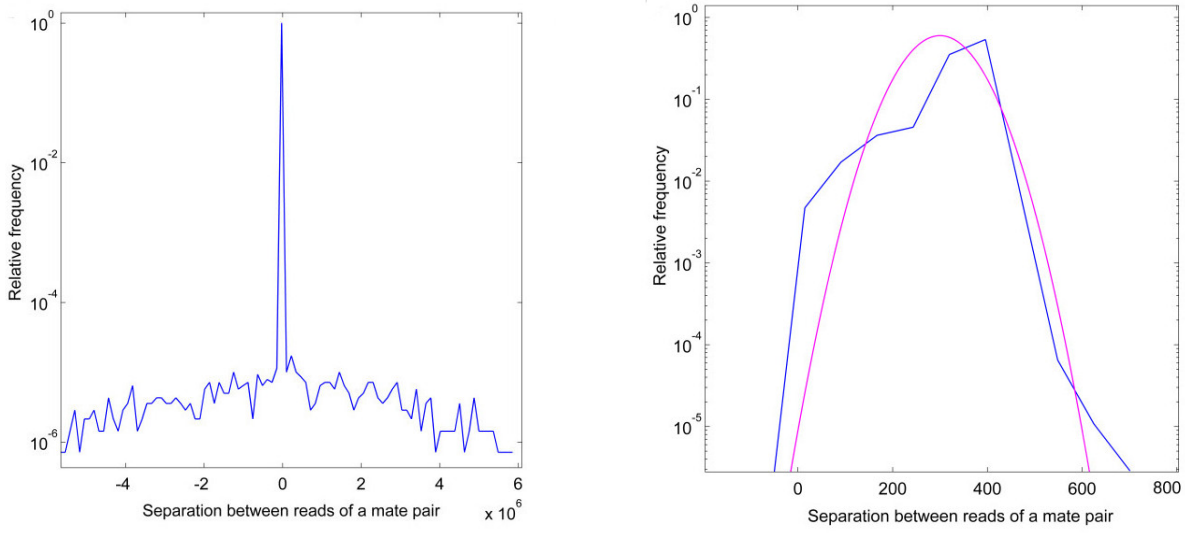
Как уже отмечалось, информация, извлекаемая из факта парности ридов, необходима для качественной сборки. Она активно используется как при разрешении повторов, так и на этапе скэффолдинга.

Но с получением информации есть одна проблема: на самом деле, единичный парный рид не дает нам представления об истинном расстоянии между его концами. Когда говорят, что дано множество парных ридов с внутренним расстоянием l, предполагается, что l — матожидание истинного распределения внутренних расстояний (см. рис. 2)

Чтобы преодолеть эту проблему, целесообразно “перенести” информацию о парных ридах на ребра сжатого графа и пересчитать ее как расстояние между началами соответствующих ребер. Ребра сжатого графа, являются в среднем достаточно длинными, и анализ информации по всем прошедшим по ним ридам может дать хорошее приближение приближение истинных расстояний.

Итак, информация о парных ридах трансформируется в еще один тип важной дополнительной информации, ассоциированной с графом — информацию о предположительных расстояниях между некоторыми парами ребер в итоговой сборке.

Мы будем представлять ее в виде некоторой структуры, хранящей отображение из некоторого множества троек (a, b, d'), где a, b—ребра, d' — возможно отрицательное, предположительное расстояние между их началами, в значение некоторого условного веса истинности этой информации. При



а) б)

Рис. 2: (а) Гистограмма реальных внутренних расстояний парных ридов для данных по P. syringae от платформы Illumina из [12]. Похожа на смесь двух распределений: распределения около предположительного внутреннего расстояния библиотеки и “широкого” фона. (б) Участок той же гистограммы около пика. Распределение, соответствующее (б) имеет большое стандартное отклонение, порядка 20% от среднего значения.

этом, вместе с тройкой (a, b, d'), там также содержится и тройка $(b, a, -d')$ с тем же весом. Эти записи добавляются и удаляются из структуры парами. За счет такого представления информации, нам достаточно рассматривать только тройки, начинающиеся с e, чтобы получить полную информацию о ее парных ребрах.

Так как мы заботимся о непротиворечивости сопряженной информации, то в структуре также должны содержаться и соответствующие записи о сопряженных парах ребер (это свойство будет достигнуто автоматически, за счет симметричности производимых для сопряженных элементов операций).

3.7.1 Первоначальный подсчет

В чем измерять веса? На первый взгляд, самым естественным вариантом весовой функции истинности некоторого расстояния, является количество парных ридов, согласованных с этим расстоянием. К сожалению, способ корректного поддержания такой весовой функции при преобразованиях, нетривиален.

Предлагается следующая модель подсчета: в начале из каждого парного рида, по одной из двух возможных стратегий, извлекается множество пар k-меров, после чего информация о расстоянии с каждой пары переносится на соответствующие ребра с весом 1 (весовая функция — количество пар k-меров, поддерживающих некоторое расстояние).

Считаем, что риды имеют внутреннее расстояние d. Стратегии получения множества пар k-меров из парного рида.

- "Скользящие" пары k-меров на расстоянии d. Минус в том, что если раньше была информация об участках генома на расстоянии $d+l$, то осталась — на $d+k$. (сможем разрешить меньше повторов)
- Все возможные пары k-меров на соответствующих расстояниях.

Если пара k-меров $(a|b)$ на расстоянии d' находятся в ребрах e_a и e_b, начиная с позиций p_a и p_b соответственно, то эта пара приносит информацию о том, что e_a и e_b находятся на расстоянии $d' + p_b - p_a$ с весом 1.

Непосредственно подсчет происходит путем прикладывания парных ридов к сжатому графу. При этом в случае обеих стратегий нет необходимости явно учитывать по-отдельности каждую извлеченную пару k-меров, так как количество пар, дающих вклад в расстояние между парами ребер и сами расстояния могут быть явно вычислены исходя из длин ребер и степеней их пересечений с ридами.

3.7.2 Правила преобразования

При конкатенации ребер простого пути $\{e_i\}$ в ребро e, нужно преобразовать всю информацию, ранее затрагивавшую хотя бы одно из ребер e_i в корректную информацию, включающую новое ребро.

Не будем описывать процесс формально, а приведем поясняющий пример. Допустим, ранее в структуре находилась информация (e_2, e_a, d') с весом w. Теперь она (вместе с информацией $(e_a, e_2, -d')$) должна исчезнуть из структуры, а информация $(e, e_a, d' - length(e_1))$ добавиться в структуру с весом w.

Весовые функции специально выбирались так, чтобы обладать свойством аддитивности относительно конкатенации.

Интересно, если немного задуматься, то станет ясно, что самая простая стратегия, которая, рассматривая ребра пути одно за другим "переносит" парную информацию с него на ребро e , корректно обрабатывает практически все нетривиальные случаи: присутствие информации вида $(e_i, e_j, .)$ или $(e_i, conj(e_j), .)$

Отдельно нужно рассмотреть лишь случай наличия информации вида $(e_i, e_i, .)$.

При разделении ребра e на e_1, e_2 в отношении α, информация разделяется между ребрами в отношении α. То есть, вместо информации (e, e_a, d) с весом w, добавляется (e_1, e_a, d) с весом $\alpha * w$ и $(e_2, e_a, d - length(e_1))$ с весом $(1 - \alpha) * w$. Как и в случае с покрытием, здесь, естественно, происходит некоторая потеря информации.

При проекции ребра e_1 на ребро e_2, вся парная информация ребра e_1 без изменений добавляется к e_2

3.7.3 Кластеризация "на лету"

Как уже отмечалось ранее, из-за значительного разброса истинных внутренних расстояний, чтобы получить представление об истинном расстоянии между двумя ребрами, необходима интеллектуальная обработка имеющихся данных. Задача достаточно непростая и не решается банальным усреднением всех значений с учетом весов. Во-первых, не известно сколько различных расстояний нужно получить, ведь два ребра могут встречаться в правильной сборке несколько раз на различных расстояниях. Это приводит к необходимости выделения кластеров среди имеющихся расстояний и поиску наиболее вероятного расстояния внутри каждого кластера в отдельности. Во-вторых, не трудно представить себе ситуацию, в которой, при условии безошибочных расстояний, два ребра вообще не могли быть связаны парными ридами. В этом случае внутренние расстояния всех парных риды, которые связывают эти ребра отличаются от предполагаемых, что может привести нас к совершенно неправильной оценке расстояний между этими ребрами. Таким образом, проблема оценки истинных расстояний между ребрами является сложной задачей, дальнейшее обсуждение которой выходит за рамки данной работы.

Но есть момент, который мы не можем обойти стороной: все из-за того

же разброса внутреннего расстояния, для сохранения всей парной информации "в лоб" потребовалось бы много места. Чтобы сэкономить память, предлагается применить простую стратегию кластеризации. Выберем константу ϵ. При добавлении в структуру информации о тройке $(e_a, e_b, d_{a,b})$ с весом w, проверяем не находится ли в ней информация о $(e_a, e_b, d'_{a,b})$, где $|d - d'| < \epsilon$ расстоянии и, если такая была, добавляем новую информацию к уже имевшейся, корректируя значение расстояния и вес. Новое расстояние определяется как взвешенное среднее исходных расстояний, а вес — их суммарному весу.

3.8 Алгоритмы коррекции графа

Были разработаны собственные методы обнаружения и устранения ошибочных участков графа, которые несмотря на свою внешнюю простоту, на практике работают не хуже, чем сложные алгоритмы, используемые в Velvet, SOAP *de novo* или Euler-SR (речь про их алгоритмы удаления пузырей).

За счет постоянного хранения и поддержания информации о межреберных расстояниях, она может использоваться как один из факторов для определения является ли ребро ошибочным, с целью повышения точности алгоритмов.

При этом, разработанные алгоритмы показывают хорошие результаты даже если вовсе запретить им использовать последовательности нуклеотидов (возможность работы в условиях "бесконтекстного" подхода было одним из требований, предъявлявшихся к новым алгоритмам).

Замечание 1: приведенные ниже алгоритмы запускаются один за другим в несколько итераций, так в результате работы каждого из них могут образоваться новые элементы, за удаление которых отвечают остальные. Также как и в Velvet, устранение ошибочных соединений запускается после удаления тупиков и пузырей, чтобы дать возможность правильным, но коротким и плохо покрытым ребрам "слиться" с более хорошо покрытыми соседями.

Замечание 2: практически для всех констант, фигурирующих ниже есть значения по умолчанию, определяемые с учетом конкретных характеристик графа.

3.8.1 Удаление тупиков

Как нетрудно видеть, в зависимости от того, в начале или в конце рида случилась ошибка, тупики бывают двух типов: "входящие" в граф и "выходящие" из него соответственно. Также ясно, что сопряженным участком каждому "входящему" тупику будет "выходящий" тупик и наоборот. Таким образом, можно искать и удалять, скажем, только "выходящие" тупики (ведь вместе с каждым элементом из графа удаляется его сопряженный).

Алгоритм 2 Алгоритм удаления "тупиков"

Вход: . C_k — сжатый граф де Брюина, l_{max} — максимальная длина "тупика"

Выход: C'_k — сжатый граф де Брюина, не содержащий "тупиков"

for e — ребро графа C_k, не имеющее продолжения, и $length(e) < l_{max}$ **do**
  if Есть альтернативное ребро e', такое что $start(e) = start(e')$ (из всех таких ребер выбирается наиболее покрытое ребро) **then**
    if Выполняются некоторые условия на e, e' (о них дальше) **then**
      Удаляем e из графа
      Пытаемся сжать окрестность его начала и включаем новое ребро (если появилось) в обход (так как новое ребро вполне может оказаться тупиком)
    end if
  end if
end for

Замечание: чтобы в процессе работы было удалено как можно меньше нуклеотидов, ребра необходимо перебирать в порядке возрастания их длин. Чтобы добиться того же эффекта, к примеру, ассемблеру ABySS приходится производить удаление тупиков в несколько итераций, каждый раз увеличивая пороговое значение для длины рассматриваемых тупиков.

Опишем условия на e, e', при которых ребро e **не будет** удалено:

- $coverage(e) > max\_cov$, где max_cov — константа, задаваемая пользователем
- $coverage(e) > r * coverage(e')$, где r — константа, задаваемая пользователем

- множества парных к e, e' ребер не имеют пересечения

Первые два условия похожи на те, которые использует Velvet, но он не дает возможности задания конкретных констант вручную.

Третье условие реализовано, но является опциональным.

3.8.2 Удаление пузырей

Простым назовем пузырь, состоящий из одного ребра. Заметим, что последовательным удалением простых пузырей и сжатием однозначных участков, можно удалить достаточно сложные структуры "наползающих" друг на друга пузырей.

Алгоритм 3 Алгоритм удаления "пузырей"

Вход: . C_k — сжатый граф де Брюина, l_{max} — максимальная длина "пузыря", δ — максимальная разница длин между ребром и альтернативным путем

Выход: C'_k — сжатый граф де Брюина, не содержащий "пузырей"

for e — ребро графа C_k и $length(e) < l_max$ **do**
  if Есть альтернативный путь $P : start(P) = start(e) \wedge end(P) = end(e)$, такой что $|length(P) - length(e)| < \delta$ (из всех путей, удовлетворяющих этому условию выбирается наиболее покрытый в среднем) **then**
    if Выполняются некоторые условия на e, P (о них дальше) **then**
      Разбиваем e на ребра, длины которых пропорциональны длинам ребер в P
      Одно за другим, проецируем новые ребра на соответствующие ребра P (используя описанную ранее соответствующую операцию графа)
      Пытаемся сжать окрестности концов e и включаем новые ребра (если появились) в обход, так как они могут оказаться новыми простыми пузырями.
    end if
  end if
end for

Опишем условия на e, P, при которых ребро e **не будет** удалено:

- $coverage(e) > max\_cov$, где max_cov — константа, задаваемая пользователем

- $coverage(e) > r * coverage(P)$, где r — константа, задаваемая пользователем, а $coverage(P)$ – среднее покрытие пути P

- последовательности нуклеотидов ребра e и пути P имеют редакционное расстояние $> max_red$, где max_red — константа, задаваемая пользователем

- множества ребер, парных к e и ребрам из P не имеют пересечения

Как и в случае с удалением тупиков, первые три условия похожи на те, которые использует Velvet, но он не дает возможности конфигурации конкретных констант вручную.

Здесь третье и четвертое условия являются опциональными.

Замечание. Есть еще одно условие, при котором ребро e не будет удалено: оно не будет удалено, если P содержит $conj(e)$ или P содержит e_i и e_j, такие что $e_i = e_j$ или $e_i = conj(e_j)$. Наличие этого условия на структуру пути P обусловлено с одной стороны нашей стратегией поддержания корректного состояния сопряженных элементов, а с другой — нежеланием возиться с большим количеством хитрых случаев, которые разбирает Velvet при “склеивании” параллельных путей.

3.8.3 Удаление ошибочных соединений

Удаление ошибочных соединений происходит так же, как и у всех. Ошибочными будем признавать все ребра e:

- $length(e) < l_{max}$

- $coverage(e) < max\_cov$

, где l_{max} и max_cov — задаваемые пользователем константы

Будем удалять из графа все такие ребра. Как и раньше, удаление одного ошибочного соединения может привести к образованию новых ошибочных соединений, которые будут включены в обход и тут же удалены.

4 Реализация и результаты эксперимента

4.1 Основные используемые структуры

- *Последовательности.* Прежде всего, стоит упомянуть о реализации последовательностей нуклеотидов. В проекте их две. Обе тратят 2 бита для хранения одного нуклеотида. Одна из них имеет фиксированную длину времени компиляции, что позволяет не хранить эту длину вместе с последовательностью и тратить меньше места, скажем, на хранение k-меров в различных индексах. Другая — полноценная реализация неизменной (immutable) последовательности произвольной длины. Реализация использует подсчет ссылок за счет этого удается просто реализовать, взятие подстроки за O(1), и, что более важно, использование одной и той же памяти для хранения сопряженных последовательностей. Это легко сделать, так как мы добавляем сопряженные элементы одновременно.

- *Несжатый граф де Брюина.* В качестве внутренней структуры хранения несжатого графа, используется некоторая реализация множества, в которую помещаются k-меры. Для вершины графа, все исходящие ребра получаются следующим естественным образом: просто перебираем 4 возможных продолжения справа $k-1$-мера, соответствующего вершине и проверяем наличие соответствующего элемента в множестве.

  По причине, объясненной чуть дальше, оказывается более эффективным хранить k-меры не в множестве, а как ключи в некотором отображении. Были испробованы различные варианты реализации этого отображения, такие как: красно-черные деревья из стандартной библиотеки, хэш таблица из стандартной библиотеки, реализация sparsehash от google, собственная реализация cuckoo хэш таблицы с открытой адресацией. По итогам тестов на занимаемую память и время доступа к элементу, остановились на хэш таблице с открытой адресацией.

- *Сжатый граф де Брюина* Сжатый граф хранит множество вершин, в них указатели на исходящие ребра и сопряженную вершину. Каждое

ребро хранит последовательность нуклеотидов и ссылку на сопряженное ребро (как уже отмечалось, за счет подсчета ссылок в реализации последовательностей, физически хранится одна из сопряженных последовательностей, но графу об этом заботиться не нужно).

- *Индекс k-меров* Индекс k-меров хранит отображение из k-мера в пару из указателя на ребро сжатого графа и отступа от начала этого ребра. При сборке больших геномов, на хранение множества k-меров уходит много места. А при построении сжатого графа в памяти одновременно должны присутствовать и несжатый граф и строящийся индекс k-меров. Была предложена интересная оптимизация потребляемой памяти: можно переиспользовать структуру хранения несжатого графа для хранения индекса. Именно поэтому, мы изначально храним k-меры не в множестве, а как ключи в отображении в пару из нулевого указателя и нулевого смещения, которые позже заменятся осмысленными значениями.

- *Оптимизация несжатого графа и индекса* Помимо сохранения непротиворечивого состояния графа, стратегия постоянного поддержания сопряженности позволяет в 2 раза снизить затраты по памяти на хранение, в начале сжатого графа, а затем и индекса k-меров. *Каноническим* будем называть тот из двух сопряженных k-меров, который меньше лексикографически. Так вот, так как нам гарантируется присутствие сопряженного ребра в графе и более того, ребра хранят ссылки на свои сопряженные, то в индексе можно физически хранить информацию лишь о расположении канонического k-мера каждой пары, а информацию о втором восстанавливать по необходимости.

4.2 Концепция “наблюдателей”

Для того, чтобы предоставить возможность ассоциировать с элементами графа произвольную информацию, было решено применить паттерн *наблюдатель*. Граф сообщает о каждой производящейся над ним операции как высокого (конкатенация, разделение, проекция) так и низкого (добавление/удаление ребра/вершины) уровней всем подписавшимся на это наблюдателям. Эта концепция позволяет своевременно обновлять структуры,

хранящие информацию, ассоциированную с элементами графа.

Наиболее простым примером структуры, отлично подходящей на роль слушателя, является уже не раз упоминавшийся индекс k-меров. Действительно, после первоначального построения, он прописывается к графу в качестве слушателя и реагирует на операции добавления/удаления ребра, внесением и удалением из себя самого информации о соответствующих k-мерах.

Также в отдельных структурах хранится информация о покрытии ребер и межреберных расстояниях.

Их заполнение происходит по построенному сжатому графу, индексу k-меров в нем и множеству входных ридов. После заполнения, структуры прописываются наблюдателями в граф и осуществляют пересчет хранящейся в них информации при преобразованиях графа по правилам, описанным ранее.

Таким образом, вся логика по работе с дополнительной информацией оказывается с одной стороны вынесена из графа, а с другой — отделена от алгоритмов обработки, что значительно упрощает их реализацию!

Есть несколько тонких моментов.

Первый из них связан с тем, что мы стремимся к поддержанию согласованности не только сопряженных элементов графа, но и ассоциированной с ними информации. Теперь вспомним, что при вызове некоторой операции, симметричные действия произойдут для сопряженных элементов неявно. Поэтому при вызове любой операции, граф должен явно сообщить всем наблюдателям не только об этой операции, но и об аналогичной операции над сопряженными элементами. Помимо этого, нужно как всегда быть осторожным с самосопряженными ребрами, иначе можно, скажем дважды подряд вызвать обработчики добавления одного и того же ребра.

Второй тонкий момент, связан с последовательностью вызовов обработчиков при операциях высокого уровня. Ясно, что, скажем, операция конкатенации использует операции добавления и удаления ребер. При этом не вызывать обработчики этих операций нельзя, так как тогда, скажем, сломается работа индекса. Теперь представим себе следующую ситуацию. Происходит конкатенация двух ребер. Пересчет и запись нового покрытия в результирующее ребро происходит в методе HandleConcat, наблюдателя,

отвечающего за покрытие. При этом новое ребро уже должно было быть добавлено в граф. Но ведь обработчики добавления в граф нового ребра корректнее было бы вызывать после того, как HandleConcat установит корректное состояние (в нашем примере покрытие) нового ребра. Общее правило такое: при реализации операций высокого уровня, обработчики этих операций должны быть вызваны прежде, чем обработчики входящих в них операций низкого уровня.

4.3 “Умные” итераторы

Все описанные нами алгоритмы коррекции сжатого графа пользуются тем, что есть способ обойти все вершины/ребра графа в некотором порядке, при этом удаляя из графа элементы, а также добавляя в граф новые элементы, которые попадут в обход. Такую возможность предоставляют “умные” итераторы. В их реализации также помогла примененная концепция наблюдателей.

Не умаляя общности будем рассматривать итератор по ребрам. Реализован он следующим образом: в каждый момент времени итератор хранит упорядоченное (в соответствии с некоторым линейным порядком, заданным при создании итератора) множество еще не пройденных ребер. При создании итератора, все ребра графа помещаются в это множество, а сам итератор добавляется слушателем к графу и реагирует на операции добавления/удаления в граф ребер добавлением/удалением элементов из этого множества.

Даже в том случае, если изменений в графе не происходит, данная стратегия усложняет процесс обхода графа до O(E*log E), где E — количество ребер графа (из-за использования упорядоченного множества). Но взамен предоставляет удобный и мощный инструмент для обработки сжатого графа.

4.4 Экспериментальные данные

Эксперимент проводился на данных по бактерии *E.coli* (Escherichia coli, кишечная палочка), полученных при её секвенировании на аппарате компании Illumina.

E.coli давно уже является стандартной бактерией для тестирования ассемблеров. Не в последнюю очередь это обусловлено тем, что ее геном был большое количество раз собран по очень качественным и длинным ридам и известен целиком с высочайшей степенью точности. В дальнейшем будем называть этот геном эталонным.

Если и не самой показательной, то уж точно наиболее популярной из всех метрик качества сборки является так называемая N50. Применительно к нашей ситуации, ее можно определить так: это максимальное число N, такое что если рассматривать ребра графа, длина которых $\geq N$, то они покроют более 50% эталонного генома.

Входные данные представляют собой 14 млн парных ридов длины 100 с небольшим внутренним расстоянием 120 нуклеотидов, обеспечивающих $600X$ кратное среднее покрытие генома.

Вначале, риды были пропущены через популярный инструмент для коррекции ошибок, Quake ([14, 2]). Затем были изъяты из рассмотрения риды, содержавшие хотя бы один неизвестный нуклеотид, а также риды, парный к которым уже был отфильтрован. В результате, 30% всех парных ридов было отброшено, но оставшиеся все еще составили $400X$ кратное покрытие.

Для эксперимента было выбрано значение $k = 36$.

Результаты сравнивались с результатами запуска ассемблера Velvet с параметрами по-умолчанию на тех же данных. Velvet был выбран из-за того, что он также, как и наш ассемблер заботится о согласованности сопряженных элементов, а также реализует алгоритмы удаления тех же типов ошибочных участков графа. Причем алгоритмы, во многом более сложные чем наши.

Построение несжатого графа заняло 20 минут и потребовало всего 350Mb оперативной памяти. Приблизительно столько же заняли бы одни различные k-меры, встречающиеся в ридах.

После этого, сжатие графа заняло всего 140 секунд, а на хранение всей структуры сжатого графа потребовалось менее 10Mb. Velvet построил граф приблизительно за то же время, при этом использовав 1.6Gb оперативной памяти!

Подсчет покрытия занял 10 минут. И подсчет межреберных расстояний занял еще приблизительно 15 минут.

На хранение информации о межреберных расстояниях при отключенной кластеризации ушло 700Mb оперативной памяти. При включенной кластеризации “на лету” с радиусом окрестности равным всего лишь 5, на хранение информации о межреберных расстояниях ушло менее 200Mb.

Построенный по ридам сжатый граф содержал 34641 ребер. “Прикладыванием” эталонного генома (и его комплементарного) к полученному графу было выяснено, что

- в графе содержится 99.98% k-меров, находящихся в эталонном геноме
- 47% ребер полученного графа (более 16 тыс.) не принадлежат к эталонному геному и являются ошибочными
- N50 составляет всего лишь 1.6kB

Алгоритмы упрощения были запущены в режиме, не использующем последовательности на ребрах. Работа всех алгоритмов упрощения графа заняла менее 2 минут, что является более чем приемлемым результатом. В результате в графе осталось всего 3605 ребер (почти в 10 раз меньше изначального количества), при этом

- в графе все еще содержится 99.98% k-меров, находящихся в эталонном геноме
- только 1.9% всех ребер (68 штук) не принадлежит эталонному графу и являются ошибочными
- N50 составляет $>$ 24kB

Процент k-меров эталонного генома, находящихся в графе изменился только лишь в 3-м знаке после запятой (было удалено только одно ребро, принадлежащее эталонному геному), что является показателем высокой точности работы наших алгоритмов, даже в режиме, предназначенном для бесконтекстного подхода и, не использующем последовательности нуклеотидов. Из более чем 16 тыс. ошибочных ребер в графе осталось 68, то есть было удалено 99% ошибочных ребер. И наконец, полученное значение N50 полностью согласуется с тем, которое было получено Velvet.

Заключение

Результаты работы

Была разработана библиотека для работы с графом де Брюина, которая является основой будущего ассемблера, разрабатываемого в Лаборатории алгоритмической биологии АУ РАН.

Все задачи, перечисленные в разделе 1.4, были успешно выполнены.

Основные результаты:

- Реализованы эффективные способы представления и построения несжатого и сжатого графов де Брюина.
- Разработаны новые способы поддержания и использования информации о покрытии ребер и межреберных расстояниях в процессе изменений графа.
- Разработаны алгоритмы устранения основных видов ошибочных участков графа, допускающие тонкую настройку.
- Спроектированная архитектура позволяет легко расширять функциональность, а также реализовывать новые алгоритмы обработки графа.

Дальнейшие исследования

Планируется усовершенствование алгоритмов устранения ошибочных участков в сжатом графе с целью повышения их полноты и точности при работе с короткими фрагментами, обеспечивающими очень неравномерне покрытие генома (highly non-uniform coverage).

Естественным и крайне важным продолжением данной работы является разработка методов для как можно более точного приближения истинных межреберных расстояний по информации, извлеченной из парных ридов.

На основе описанной реализации сжатого графа де Брюина в лаборатории активно ведется разработка сразу нескольких различных модулей для разрешения повторов в графе де Брюина.

Также на основе описанной библиотеки, планируется начать реализацию подхода, который позволит эффективно и точно находить длинные пути, гарантированно присутствующие в геноме (именно они и должны становиться результатом качественной сборки).

Список литературы

[1] Abyss, http://www.bcgsc.ca/platform/bioinfo/software/abyss.

[2] Quake, http://www.cbcb.umd.edu/software/quake.

[3] The shortest superstring problem, http://www.cs.ust.hk/ golin/comp670j.

[4] Soap de novo, http://soap.genomics.org.cn/soapdenovo.html.

[5] Velvet, http://www.ebi.ac.uk/ zerbino/velvet/.

[6] J. Butler, I. MacCallum, M. Kleber, I.A. Shlyakhter, M.K Belmonte, E.S Lander, C. Nusbaum, and D.B Jaffe. Allpaths: de novo assembly of whole-genome shotgun microreads. *Genome Research*, 18(5):810–20, 2008.

[7] M. Chaisson, P. Pevzner, and H. Tang. Fragment assembly with short reads. *Bioinformatics*, 20:2067–2074, 2004.

[8] M.J. Chaisson, D. Brinza, and P.A. Pevzner. De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Research*, 19:336–346, 2009.

[9] M.J. Chaisson and P.A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Research*, 18:324–330, 2008.

[10] P. Compeau and P. Pevzner. Genome reconstruction: A puzzle with a billion pieces. 2010.

[11] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[12] A. Dayarian, T.P. Michael, and A.M. Sengupta. Sopra: Scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics*, 11, 2010.

[13] N. Jones and P.Pevzner. *An Introduction to Bioinformatics Algorithms*. The MIT Press, 2004.

[14] D.R. Kelley, M.C. Schatz, and S.L. Salzberg. Quake: quality-aware detection and correction of sequencing errors 2010 11(11):r116. *Genome Biology*, 11, 2010.

[15] Ruiqiang Li, Hongmei Zhu, and et al. Jue Ruan. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20:265–272, 2010.

[16] I. MacCallum, D. Przybylski, S. Gnerre1, and J. Burton. Allpaths 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome Biology*, 10, 2009.

[17] E. Myers. The fragment assembly string graph. *Bioinformatics*, 21:1179–1185, 2005.

[18] K. Paszkiewicz and D.J. Studholme. De novo assembly of short sequence reads. *Briefings in bioinformatics*, 11:457–472, 2010.

[19] P.A. Pevzner and H. Tang. Fragment assembly with double-barreled data. *Bioinformatics*, 17:225–33, 2001.

[20] P.A. Pevzner, H. Tang, and M.S. Waterman. An eulerian path approach to dna fragment assembly. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 98, page 9748–9753, 2001.

[21] M.C. Schatz, A.L. Delcher, and S.L. Salzberg. Assembly of large genomes using second-generation sequencing. *Genome research*, 20(9):1165–73, 2010.

[22] J. Shendure and H. Ji. Next-generation dna sequencing. *Nat Biotechnol*, 26:1135–1145, 2008.

[23] J.T. Simpson, K. Wong, and S.D. Jackman. Abyss: A parallel assembler for short read sequence data. *Genome research*, 2009.

[24] D. Zerbino, G. McEwen, E. Margulies, and E. Birney. Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *NCBI*, 4, 2009.

[25] Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18:821–829, 2008.

[26] М.Э. Дворкин. Алгоритм секвенирования ДНК на основе парных цепей. Master's thesis, СПбГУ ИТМО, 2010.

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