Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Магистерская диссертация

Разработка метода восстановления фрагментов генома по парным чтениям с ошибками вставки и удаления

А. А. Сергушичев, гр. 6539

Научный руководитель: д.т.н., проф. А. А. Шалыто

Санкт-Петербург

2013

Содержание

Введение

Наследственная генетическая информация живых организмов (геном) закодирована в молекулах дезоксирибонуклеиновой кислоты (ДНК). Эта информация описывает существенную часть того, как организм будет развиваться и какими свойствами будет обладать. В связи с этим, в разных областях человеческой деятельности знание последовательности ДНК организмов оказывается полезным. Например, информация о геноме человека может быть для определения предрасположенностям к болезням, а информация о геномах бактерий может быть использована для получения новых видов бактерий для более эффективного производства биотоплива.

Длина молекул ДНК может составлять от нескольких миллионов до нескольких сотен миллионов нуклеотидов. Существующие технологии секвенирования не позволяют определить последовательность нуклеотидов в таких молекулах напрямую. В настоящее время возможно определить последовательность для молекул длиной в несколько тысяч нуклеотидов, но этот процесс можно проводить одновременно для большого числа молекул. На этом основан способ определения последовательности ДНК, используемый в настоящее время. Молекула ДНК разбивается на множество перекрывающихся фрагментов небольшой длины, которые могут быть прочитаны секвенатором. Затем по множеству известных последовательностей этих фрагментов с помощью различных алгоритмов делается попытка восстановить полный геном. Кроме того, существует технология получения парных чтений, когда прочитывается не весь фрагмент, а его концы. В этом случае для парных чтений известно примерное расстояние между ними.

Все секвенаторы обладают различными наборами параметров: длина и качество чтений, стоимость самого секвенатора и одного запуска, время работы и др. В настоящее время стали появляться секвенаторы, к недостаткам которых можно отнести большую долю ошибок вставки и

удаления, но большинство существующих программ для сборки генома не поддерживают такой тип ошибок. В том числе и разработанный в работе [1] метод восстановления фрагментов из парных чтений. В настоящей работе этот метод развивается так, чтоб он мог работать с данными с ошибками вставки и удаления.

1 Обзор

В настоящей главе приводится обзор задачи сборки генома и существующих методов ее решения, а также выполняется постановка задачи восстановления фрагментов из парных чтений с ошибками вставки и удаления.

1.1 Геном

В ходе различных экспериментов к середине двадцатого века было установлено, что вся наследственная информация об организме находится в молекулах дезоксирибонуклеиновой кислоты (ДНК). В 1953 году Уотсоном и Криком была предложена модель двухцепочечной спирали [2], что затем нашло подтверждение.

Молекула ДНК состоит из двух цепочек, являющихся последовательностями нуклеотидов: аденина (A), тимина (T), гуанина (G) и цитозина (C). Нуклеотиды в парах A-T и G-C комплементарны друг другу, и в этих цепочках на одинаковых позициях находятся комплементарные нуклеотиды. Два конца любой цепочки ДНК различаются с химической точки зрения, один из них называется 5', другой – 3′. Таким образом, у любой цепочки можно определить направления, обычно прямым считается направление от 5' к 3', что обусловлено тем, что большинство биологических процессов по обработке цепочки ДНК протекают именно в этом направлении. Две цепочки в одной спирали противонаправлены.

Наследственная информация расположена не в одной молекуле ДНК, а, вообще говоря, в нескольких, обычно расположенных в хромосомах. Кроме того, у некоторых организмов, к которым, например, относится большинство млекопитающих, набор хромосом удвоен, такие организмы называются диплоидными. В каждой паре находится по одной хромосоме от каждого родителя, причем эти хромосомы, за исключением пары

половых хромосом, отличаются в относительно небольшом числе нуклеотидов, такое отличие в одном нуклеотиде называется однонуклеотидным полиморфизмом. Также существуют организмы с утроенным, учетверенным и т.д. наборами хромосом.

Задачей сборки генома является определение последовательности ДНК в каждой из хромосом организма.

1.2 Существующие методы секвенирования

В данном разделе кратко описаны существующие методы секвенирования. В основе любого метода лежит технология, позволяющая физически узнавать, в какой последовательности находятся нуклеотиды в молекуле ДНК. С развитием этих технологий изменялись и методы секвенирования геномов.

1.2.1 Метод обрыва цепи

Фредериком Сенгером к 1977 году был разработан метод обрыва цепи [3]. Образец цепочки ДНК, которую необходимо секвенировать, делится на четыре группы, по одной на каждый нуклеотид. Затем в каждой группе происходит *de novo* синтез молекулы нуклеиновой кислоты из, например, радиоактивно помеченных нуклеотидов, причем этот синтез может завершиться только в нуклеотиде, соответствующем группе. После этой операции в каждой группе получаются молекулы разных весов, в зависимости от того, в каких позициях находится соответствующий группе нуклеотид в исходной цепочке. Далее в каждой группе с помощью электрофореза происходит разделение молекул по весам с точностью до одного нуклеотида, что позволяет узнать, в каких позициях этот нуклеотид встречается в исходной цепочке.

В настоящее время этот метод позволяет напрямую секвенировать последовательности из примерно 800–1000 нуклеотидов.

1.2.2 **Метод дробовика**

Для того чтобы секвенировать геномы больше тысячи нуклеотидов примерно одновременно было предложено два метода: метод обхода хромосомы (*chromosome walking*) [4] и метод дробовика (*shotgun sequencing*) [5].

Метод обхода хромосомы заключается в том, чтобы сначала секвенировать первую тысячу нуклеотидов, затем, зная последние нуклеотиды первого прочитанного фрагмента и зацепившись за его конец, секвенировть очередную тысячу нуклеотидов и т.д. Такой метод оказывается неэффективным для больших геномов.

Более удобным для применения к большим геномам оказывается метод дробовика, в котором сначала молекулы ДНК случайным образом разделяются на небольшие перекрывающиеся фрагменты, которые затем секвенируются с помощью, например, метода обрыва цепи. За счет большого числа итераций получается множество последовательностей, покрывающих весь геном. На основе анализа наложений этих чтений друг на друга можно попытаться восстановить всю исходную последовательность. Таким образом, задача секвенирования становится не только биохимической, но и вычислительной.

1.2.3 **Высокопроизводительные методы секвенирования**

В связи с ростом производительности вычислительной техники появилась возможность использовать методы, которые дают менее качественные результаты, зато больше и дешевле. При этом все также используется метод дробовика.

В статье [6] производится сравнение существующих секвенаторов. Некоторые их характеристике приведены в таблице 1.

Таблица 1 – Технические характеристики секвенаторов нового поколения

Платформа	Illumina	Ion Torrent	PacBio RS	Illumina	Illumina

	MiSeq	PGM		GAIIx	HiSeq 2000
Стоимость	$128000	$80000	$695000	$256000	$654000
Объем ДНК, прочитываемый за один проход	$1,5\text{-}2\cdot10^9$ Гбаз	20–1000 Мбаз	100 Мбаз	30 Гбаз	600 Гбаз
Стоимость за Гб*	$502	$1000	$2000	$148	$41
Длительность одного запуска	27 часов	2 часа	2 часа	10 дней	11 дней
Процент ошибок	0.80 %	1.71 %	12.86 %	0.76 %	0.26 %
Тип ошибок	Замена	Вставка/ удаление	Вставка/ удаление	Замена	Замена
Длина чтения	До 150 баз	~200 баз	~1500 баз	До 150 баз	До 150 баз
Парные чтения	Да	Да	Нет	Да	Да
Размер фрагмента	до 700 баз	до 250 баз	до 10 Кб	до 700 баз	до 700 баз

Стоит отметить, что стоимость секвенирования быстро падает (рис. 1) и обгоняет скорость роста производительности вычислительной техники, что влечет за собой необходимость разработки эффективных алгоритмов сборки генома.

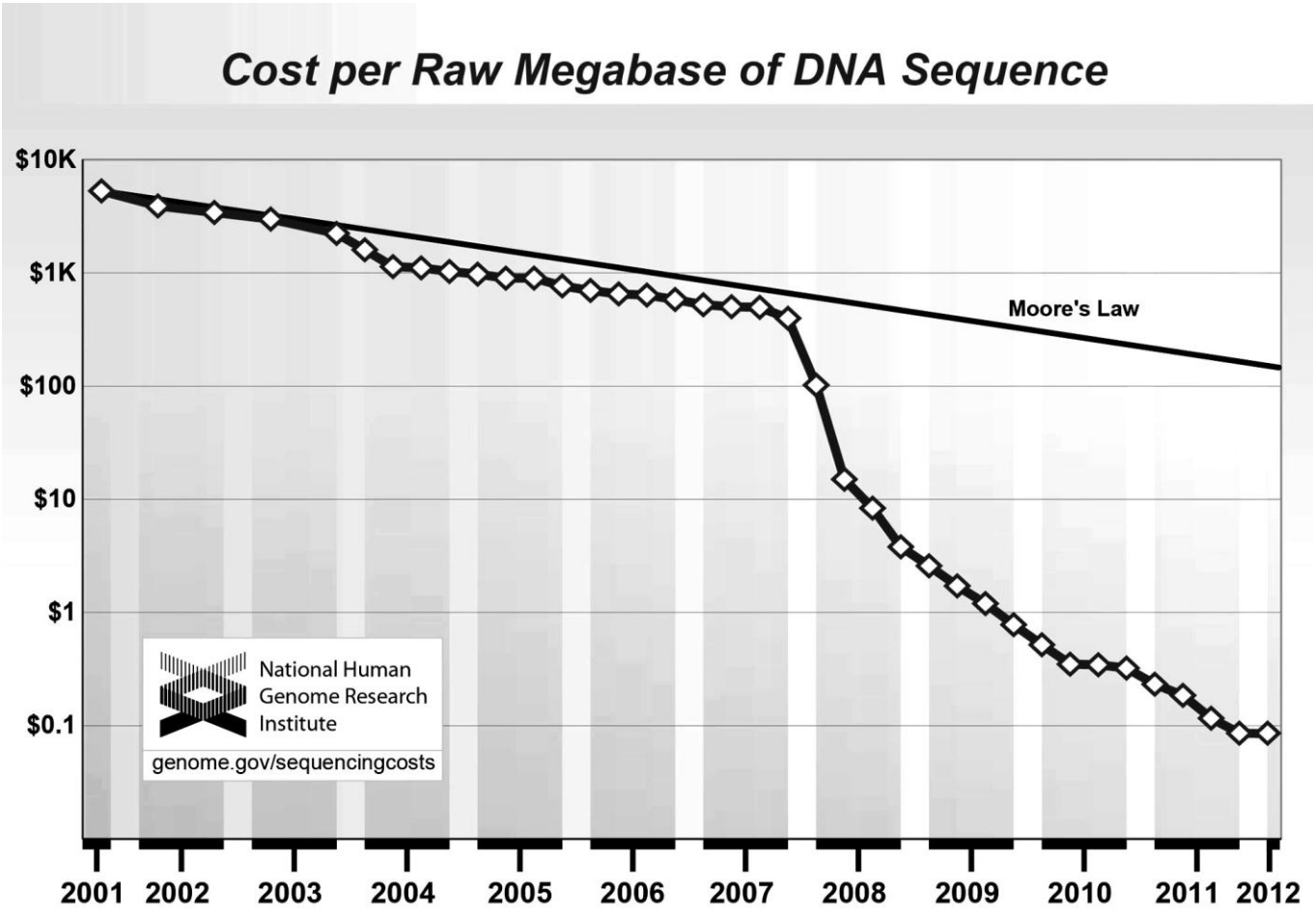

Рисунок 1 – Снижение стоимости секвенирования

1.3 Постановка задачи восстановления геномной последовательности

Таким образом, в процессе получения геномной последовательности возникает следующая задача. Имеется набор библиотек чтений (библиотекой называется набор чтений имеющих одинаковые характеристики), по ним необходимо получить как можно больше информации о последовательном расположении нуклеотидов в геноме. Обычно решением этой задачи является набор контигов, объединенных в скэффолды. *Контигом* называется непрерывная последовательность нуклеотидов. *Скэффолдом* называется последовательность контигов с оценкой на расстояния между ними. При этом предполагается, что такие (или близкие к ним) контиги и скэффолды действительно присутствуют в геноме.

Часто процесс построения решения разбит на три этапа. Сначала в исходных данных исправляются или удаляются ошибки. Затем из чтений, возможно с использованием парной информации из библиотек с небольшим размером фрагмента, строятся контиги. В конце по парным библиотекам с большим расстоянием между чтениями полученные контиги упорядочиваются в скэффолды.

1.3.1 **Задача о наименьшей общей надстроке**

Одной из первых формальных постановок задачи сборки генома является поиск наименьшей общей надстроки последовательностей фрагментов генома. Пусть $S=\{s_i\}$ – множество прочитанных последовательностей, задача состоит в отыскании такой минимальной строки s^*, что все s_i являются подстрокой этой строки. В такой формулировке задача сборки генома является NP-трудной. Кроме того, эта формулировка обладает недостатком, что длинные повторяющиеся участки генома присутствуют в наименьшей общей надстроке только один раз.

1.3.2 **Граф перекрытий**

Проблемы учета повторов в постановке задачи как поиска наименьшей общей подстроки могут быть частично решены, если использовать графы. В графовой модели возможно несколько постановок задачи сборки генома. В статье [7] было предложено построить *граф перекрытий* – граф, вершинами которого являются чтения, а ребрами – перекрытия между ними, размером перекрытия не меньше константы τ. После построения граф упрощается, путем удаления транзитивных перекрытий. Транзитивным называется такое перекрытие $u \to w$, что существует вершина v и ребра $u \to v$ и $v \to w$. Такие ребра удаляются, так как последовательность, образуемую путем $u \to w$, можно получить через путь $u \to v \to w$. Для удаления транзитивных ребер предлагается алгоритм, со временем работы равным $O(\sum_{v \in V} \mathrm{tr.deg}(w)\deg(v))$, где $\deg(v)$ –

степень вершины в исходном графе, а $\mathrm{tr.deg}(v)$ – степень вершины в графе с удаленными транзитивными ребрами.

После удаления транзитивных ребер вершины можно разбить на две группы: внутренние, имеющие входящую и исходящую степени равные единице, и узловые – все оставшиеся. Также можно определить составные ребра – пути, соединяющие две узловые вершины через последовательность внутренних. Далее для каждого составного ребра можно определить вероятность того, что оно встречается в геноме ровно один раз. Тогда каждое ребро можно отнести к одной из трех групп:

а) *Точные ребра* – ребра, которые с большой вероятностью встречаются в геноме ровно один раз. Ребра относятся к этой группе, если посчитанная вероятность меньше единицы на небольшую константу.

б) *Обязательные ребра* – ребра, которые встречаются в ответе не меньше одного раза. Составные ребра, которые не были отнесены к первой группе, но содержат внутренние вершины, должны быть посещены хотя бы один раз, так как все вершины должны быть посещены хотя бы один раз.

в) *Необязательные ребра* – ребра, на которые нет никаких ограничений. К этой группе относятся все остальные ребра, то есть ребра, не имеющие внутренних вершин.

Затем, чтобы определить, сколько раз должно быть посещено каждое ребро, в полученном графе решается задача о потоке минимального стоимости, удовлетворяющего ограничениям. Для точных, обязательных и необязательных ребер нижние и верхние границы для потока соответственно равны $[1,1]$, $[1,+\infty)$ и $[0,+\infty)$. После этого из графа удаляются все ребра, которые не должны быть посещены ни разу. Упрощенный таким образом граф является результатом сборки. Задача поиска потока решается за полиномиальное время, следовательно, за полиномиальное время работает и весь предлагаемый метод.

Заметим, что этот метод может давать несвязный граф, даже если исходный геном представляет собой одну циклическую хромосому. Это, в частности, демонстрируется теоремой, доказанной в [8], о том, что поиск обхода удовлетворяющего ограничениям в графе с таким образом классифицированными ребрами является *NP*-трудной задачей.

1.3.3 **Граф де Брёйна**

В связи с развитием высокопроизводительного секвенирования, при котором получается большое покрытие относительно короткими чтениями, стали развиваться подходы на основе графа де Брёйна.

Графом де Брёйна степени k над алфавитом Σ для множества строк $S=\{s_1, ..., s_n\}$ называется ориентированный граф, в котором вершинами являются все строки из Σ^k, которые являются подстрокой какой-либо строки s_i из S. Ребрами являются строки e из Σ^{k+1}, которые являются подстрокой какой-либо строки из S, причем ребро e соединяет вершины $e[1..k]$ и $e[2..k+1]$.

В модели графа де Брёйна задачу сборки генома можно свести к задаче китайского почтальона (Chinese Postman Problem), в которой требуется найти цикл, проходящий по всем ребрам хотя бы по одному разу, – китайский обход, минимального веса. Действительно, каждой ребро присутствует в чтениях, поэтому должно присутствовать и в пути, соответствующему геному. Эта задача может быть решена за полиномиальное время с помощью сведения к задаче о потоке минимальной стоимости и поиску Эйлерова цикла [9]. Если граф является не связным, то задача не имеет решения. Для связного графа задача решается в два этапа. Сначала для того, чтобы определить, сколько раз должно быть посещено каждое ребро, ставится задача о поиске потока минимальной стоимости, при цене за пропускание единицы потока по любому ребру равной единице. При этом по каждому ребру должен проходить поток объема не меньше единицы, так как в исходной задаче

цикл должен пройти по каждому ребру хотя бы раз. Далее возникает задача поиска цикла, проходящего по всем ребрам заданное число раз (разное для каждого ребра), которая эквивалентна задаче о поиске Эйлерова цикла в мультиграфе, где каждое ребро заменено на несколько ребер, число которых равно тому, сколько раз по этому ребру должен был пройти цикл в исходном графе.

Правильность этого алгоритма следует из того, что, во-первых, в полученном мультиграфе всегда существует Эйлеров цикл, так как необходимые для этого условия: связность и балансировка входящих и исходящих ребер, – следуют соответственно из того, что поток по каждому ребру не меньше единицы, и того, что для каждой вершины выполняется условие балансировки потока. Во-вторых, будет найден именно минимальный обход, потому что из построения длина обхода равна стоимости соответствующего потока.

Заметим, что часто может возникнуть ситуация, когда в графе существуют несколько минимальных китайских обходов. В этом случае можно поставить задачу поиска путей, которые являются подпутями любого минимального китайского обхода. Эту задачу в статье [10] предлагается решать в два этапа, соответствующих этапам решения задачи китайского почтальона. Определять сколько раз должно быть посещено ребро можно не только с помощью поиска потока минимальной стоимости, но и с помощью поиска минимального паросочетания между вершинами с входящей степенью большей исходящей, составляющими одну долю, и вершинами с входящей степенью меньшей исходящей, составляющими другую долю. Стоимость ребра между вершинами в двудольном графе равна длине минимального пути в исходном графе. Выбор паросочетания минимальной стоимости эквивалентен поиску путей, вдоль которых необходимо пропустить поток для балансировки, в задаче поиска потока минимальной стоимости. Известен полиномиальный алгоритм, позволяющий найти паросочетание, являющееся

подпаросочетанием любого минимального паросочетания. Затем строится мультиграф, в котором ищутся пути, являющиеся подпутем любого Эйлерова пути, что также можно сделать за полиномиальное время.

Сведение к задаче китайского почтальона не учитывает, что (k+1)-меры были получены из более длинных чтений. Чтобы учесть эту информацию ставится задача поиска кратчайшего надпути в графе де Брёйна [11]. Пусть дано множество чтений $R=\{r_1, ..., r_n\}$, не содержащих ошибок. Построим над этим множеством строк граф де Брёйна для некоторого k. Каждому чтению r_i длиной не меньше k+1 соответствует путь p_i в графе де Брёйна длины $|r_i| - k$. Будем считать, что длина всех чтений не меньше k+1. Тогда получается множество путей $P=\{p_1, ..., p_n\}$. Задача поиска кратчайшего надпути состоит в том, чтобы найти в этом графе путь P наименьшей длины, проходящий через каждое ребро не меньше одного раза и содержащий все пути из P как подпути. В статье [8] доказывается, что эта задача является *NP*-трудной, путем сведения к ней задачи о наименьшей общей подстроке.

1.4 Задача сборки генома из парных чтений

Для задачи сборки генома с учетом парных чтений существует несколько подходов. В этом разделе приведены некоторые из них.

1.4.1 **Восстановление фрагментов**

В работах [12,13] рассматриваются методы, позволяющие восстановить фрагмент из которого были получены парные чтения, тем самым сведя задачу к задаче сборки генома из одиночных чтений. Будем называть последовательности фрагментов, полученных при таком подходе, *квазиконтигами*. Метод, предложенный в [12], работает только если парные чтения перекрываются.

В работе [13] рассматривается сборщик GapFiller, который является локальным сборщиком, основанным на методе постепенного наращивания квазиконтигов. В качестве первого приближения квазиконтига

используется чтение, затем на каждой итерации производятся попытки увеличения квазиконтига. Для этого из всех чтений выбираются те, которые лучше всего перекрываются с концом текущей версии квазиконтига. Внутри этой группы производится выбор консенсусной последовательности. В зависимости от того, насколько часто нуклеотид из консенсусной последовательности встречается в соответствующих позициях чтений, некоторые из чтений обрезаются и удаляются. После этого консенсус считается еще раз и квазиконтиг наращивается.

Остановка цикла наращивания квазиконтига происходит в одном из четырех случаев:

а) Число чтений, перекрывающихся с концом квазиконтига, меньше заранее заданного порога m. В этом случае квазиконтиг не может быть продолжен.

б) Число чтений, оставшихся после удаления, становится меньше m. Это обычно означает, что квазиконтиг нельзя продолжить из-за повтора.

в) Было найдено чтение, парное к тому, которое было использовано в качестве первого приближения квазиконтига. Это означает, что скорее всего квазиконтиг был получен правильно.

г) Длина квазиконтига превышает заранее заданный порог $L_{\max}$, что означает, что чтение, парное к исходному, не было найдено, и квазиконтиг был продолжен неправильно.

Для того, чтобы быстро искать чтения, перекрывающиеся с суффиксом текущего квазиконтига применяется хеш-таблица. Перед запуском алгоритма выбирается параметр b – длина префикса, по которому будут индексироваться чтения. Все чтения добавляются в хеш-таблицу, отображающую префикс чтения в список чтений с таким префиксом.

1.4.2 **Парные графы де Брёйна**

Большинство сборщиков используют обычный граф де Брёйна и использует парную информацию после построения этого графа. В подходе, описанном в [14], вместо обычного графа де Брёйна используется специально разработанный *парный граф де Брёйна*.

В модели, используемой в этой статье, геном преставляет собой циклическую строку, а все чтения имеют одинаковую длину l. Распространение рассматриваемого подхода на случай нескольких хромосом и чтений разной длины не представляет труда. Соответственно, парные чтения – пары подстрок длины l, прочитанных с некоторых позиций i и j, находящихся на расстоянии $d = j - i$ друг от друга (для начала полагается, что все чтения в парах находятся на одинаковых расстояниях друг от друга).

Как и в случае с обычным графом де Брёйна, чтения преобразуются в набор из l-k пар (k+1)-меров. В каждой паре k-меры находятся на расстоянии d друг от друга. Такая пара, состоящая из (k+1)-меров a и b, обозначается ($a|b$), и называется (k+1, d)-мером.

Для удобства вводятся следующие обозначения:

а) $prefix(a)$ – строка, состоящая из первых k символов (k+1)-мера a;

б) $suffix(a)$ – строка, состоящая из последних k символов (k+1)-мера a;

в) $prefix(a|b) = (prefix(a)|prefix(b))$;

г) $suffix(a|b) = (suffix(a)|suffix(b))$.

Построение парного графа де Брёйна для набора (k+1, d)-меров C производится следующим образом. Пусть G_0 – граф из $2|C|$ вершин. Для каждого (k+1, d)-мера ($a|b$) в граф добавляются вершины u и v, а также ребро из u в v. Ребро помечается элементом ($a|b$), вершина u имеет метку $prefix(a|b)$, вершина v – $suffix(a|b)$. Все вершины, имеющие одинаковые

метки, склеиваются в одну, при этом переносятся как входящие, так и исходящие ребра.

На рис. 2 изображены обычный и парный графы де Брёйна, построенные по одной и той же строке. В парном графе де Брёйна больше вершин, что является преимуществом перед обычным графом де Брёйна, так как это означает, что меньше вершин, соответствующих разным местам генома, склеились в одну, что позволит лучше обработать повторы.

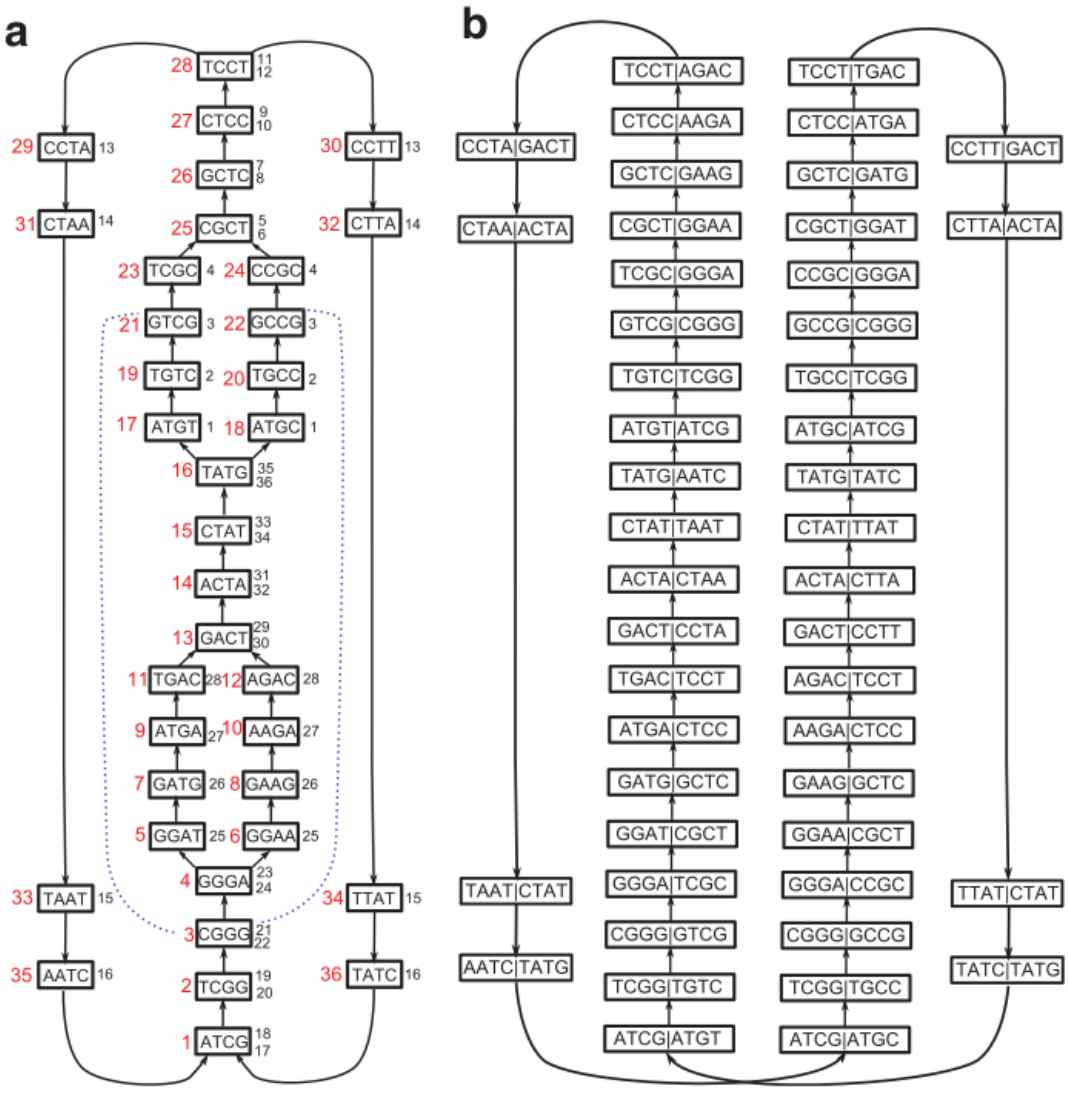

Рисунок 2 – Обычный и парный графы де Брёйна, построенные по одной строке

Если в случае с обычным графом де Брёйна путь в графе соответствовал подстроке генома, то в случае с парным графом путь

соответствует двум подстрокам. Можно использовать, например, левую из них так же, как в случае с обычным графом.

К сожалению, чтения, предоставляемые секвенаторами, располагаются не на фиксированном расстоянии от своих пар. Обычно известны нижняя и верхняя границы для этих расстояний. Таким образом, расстояния между парами чтений находятся в промежутке $[d - \Delta, d + \Delta]$. Значения d и Δ зависят от библиотеки.

Как и раньше, чтения преобразуются в набор (k+1, d, Δ)-меров, после чего требуемый *примерный парный граф де Брёйна* (*approximate paired de Bruijn Graph)* для набора (k+1, d, Δ)-меров C строится в два шага:

а) Пусть G_0 – граф из $2|C|$ вершин. Для каждого (k+1, d, Δ)-мера ($a|b$) в граф добавляется вершины u и v, а также ребро из u в v. Ребро помечается элементом ($a|b$), вершина u имеет метку *prefix*($a|b$), вершина v – *suffix*($a|b$).

б) Склеить все пары вершин вида ($\alpha|\beta$) и ($\alpha|\beta'$), для которых в обычном графе де Брёйна, построенному по C, есть путь из вершины β в вершину β` (или наоборот) длины не более 2Δ.

Смысл склеивания состоит в том, чтобы объединить в одну вершину те вершины, которые могут соответствовать одному месту в геноме.

Отличие этого графа от графа для пар чтений с одинаковыми расстояниями состоит в том, что вершины при таком склеивании не могут сохранять свои метки. Левая часть метки у всех склеиваемых вершин одна и та же, однако правые могут различаться. Поэтому в рассматриваемом алгоритме правые метки вершин после построения графа не используются.

1.4.3 **Pathset-графы**

Эффективность использования парных графов де Брёйна снижается по мере увеличения длины промежутка, в котором находятся расстояния между чтениями. В [15] описывается другая структура – *граф путемножеств* (*Pathset Graph*). Вместо обычных графов де Брёйна в этой

работе используются *сжатые* графы де Брёйна, полученные из обычных заменой каждого максимального неветвящегося пути ребром с длиной, равной числу ребер в первоначальном пути.

Как и в работе с парными графами де Брёйна, первый шаг алгоритма – преобразование парных чтений в последовательность пар k-меров. Для упрощения делается предположение о том, что геном определяет *геномный обход*, проходящий по всем ребрам в графе де Брёйна. Поскольку некоторые ребра могут встречаться в этом пути несколько раз, необходимо различать ребро e и его экземпляр в геномном пути $\tilde{e}$. Для любых двух экземпляров ребер $\tilde{e}_1$ и $\tilde{e}_2$ можно рассчитать *геномное расстояние* $d_w(\tilde{e}_1, \tilde{e}_2)$. Тройка $(e_1, e_2, , d_w(\tilde{e}_1, \tilde{e}_2))$ называется *геномной реберной парой.* Для реберной пары (e_1, e_2) определяется *парная реберная гистограмма* h, где $h(x)$ – число (k, d, Δ)-меров, поддерживающих геномное расстояние x между e_1 и e_2.

Поскольку размеры фрагментов имеют нормальное распределение, гистограмма представляет собой выдержку из нормального распределения или, если рассматриваемая реберная пара встречается в геномном пути несколько раз, смесь нескольких выдержек из нормальных распределений. Для удобства эта гистограмма сглаживается, после чего выбирается пороговое значение и от диаграммы оставляются только интервалы, значения на которых лежат выше выбранного порога. Эти интервалы, называемые *парными реберными интервалами*, не пересекаются и отвечают одной или нескольким реберным парам с близкими расстояниями. Интервал называется *корректным*, если существует геномная реберная пара (e_1, e_2, D), причем $a \leq D \leq b$. Пороговое значение должно выбираться таким образом, чтобы максимизировать число корректных интервалов, при этом разделяя геномные реберные пары с достаточно различными расстояниями.

После преобразования гистограммы во множество интервалов каждый интервал $(e_1, e_2, [a, b])$ преобразуется во множество всех путей,

начинающихся в e_1, заканчивающихся в e_2 и имеющих длину не меньше a и не больше b. Такие множества называются *путемножествами* (*pathsets*).

Затем происходит разделение путемножеств. Два ребра в путемножестве называются *независимыми*, если ни один путь в этом путемножестве не содержит оба этих ребра сразу. Ребро e называется *необходимым* для путемножества, если в нем существует геномный путь, содержащий e. Множество необходимых ребер называется *независимым*, если любые два ребра в этом множестве являются независимыми. Легко убедиться, что независимое множество ребер $A = \{a_1, \dots, a_t\}$ задает разбиение путемножества на t путемножеств: PS_{a_1}, …, $PS_{a_{t-1}}$, $PS_{\overline{a_1},\dots,\overline{a_{t-1}}}$. Каждое из них содержит необходимое ребро из A и потому является корректным.

В изображенном на рис. 3 путемножстве PS ребра e_2 и e_3 (аналогично, e_5 и e_6) являются независимыми. Это позволяет разделить PS на два путемножества: $PS_{e_2} = \{e_1e_2e_4e_5e_7, e_1e_2e_4e_6e_7\}$ и $PS_{e_3} = \{e_1e_3e_4e_5e_7, e_1e_3e_4e_6e_7\}$

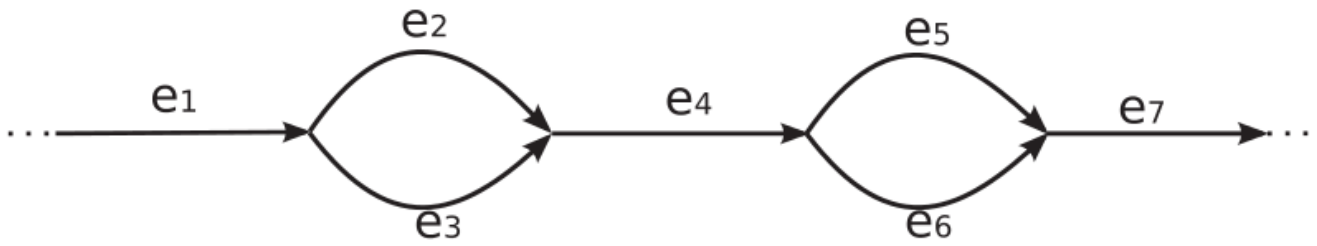

Рисунок 3 – Путемножество, построенное по ребреной паре (e_1, e_2, d)

После разделения путемножеств строится *граф путемножств* (*pathset graph*). Вершины в нем отвечают путемножествам, а неветвящиеся пути – контигам.

1.5 Постановка задачи

В данной работе рассматривается процесс сборки генома состоящий из четырех этапов:

а) исправление ошибок;

б) сборка квазиконтигов из парных чтений;

в) сборка контигов из квазиконтигов;

г) построение *скэффолдов*.

В настоящей работе рассматривается шаг сборки квазиконтигов. Квазиконтигами называются последовательности, полученные путем заполнения промежутка в парных чтениях. В работе [1] был предложен метод сборки квазиконтигов из парных чтений, полученных с помощью секвенаторов компании *Illumina*, в которых практически отсутствуют ошибки вставки и удаления. В связи с тем, что в настоящее время появляются новые технологии секвенирования, в том числе и с ошибками вставки и удаления, возникает задача учета этого типа ошибок. В данной работе были поставленны следующие задачи:

а) Разработать алгоритм сборки квазиконтигов из парных чтений, содержащих ошибки вставки и удаления.

б) Реализовать алгоритм на одном из языков программирования.

в) Провести экспериментальное исследование этого алгоритма на чтениях, полученных секвенатором *IonTorrent*.

1.6 Выводы по главе 1

В данной главе были рассмотрены существующие технологии секвенирования. Было показано, что среди современных секвенаторов присутствуют секвенаторы с достаточно большой долей ошибок вставки и удаления.

Были рассмотрены различные подходы к постановке задачи сборки генома и способы ее решения. Большинство из подходов не учитывают наличие ошибок вставки и удаления.

Была выполнена постановка задачи настоящей работы.

2 Предлагаемый метод

В настоящей главе приводится описание предлагаемого метода восстановления фрагментов из парных чтений с ошибками вставки и удаления.

2.1 Общее описание алгоритма

Предлагаемый метод сборка квазиконтигов использует взвешенный граф де Брёйна построенный на чтениях. Вершины этого графа являются k-меры, ребрами – p-меры, $p=k+1$.

Для каждой пары чтений в графе де Брёйна находится множество вершин и ребер, через которые проходит хотя бы один путь, соединяющий эти два чтения, подходящий по длине под априорные границы на расстояние между чтениями. В получившемся подграфе выделяется подграф с большим весом так, что все возможные пути, соединяющие парные чтения совпадают с каким-нибудь путем по оставшимся ребрам с точностью до небольшого числа ошибок замены, вставки или удаления. Если после этого граф состоит из единственного пути, то последовательность нуклеотидов на этом пути выводится как квазиконтиг.

2.2 Представление графа де Брёйна

Для того, чтобы потребление памяти предлагаемого метода было не очень большим, необходимо иметь компактное представление используемого подграфа графа де Брюина. Для его хранения достаточно хранить только множество его ребер, что можно эффективно делать, используя, например, хеш-таблицу с открытой адресацией. Преимуществами такого подхода хранения перед другими являются его простота, быстродействие и возможность балансировки между используемой памятью и скоростью. Более эффективными с точки зрения потребляемой памяти являются rank/select словари [16], которые

позволяют сделать ее использование близким к энтропии, но из-за этого увеличивается время доступа.

Важным является и то, что каждый *(k+1)*-мер входит в граф вместе с обратно-комплементарным. Тогда вместо пары *(k+1)*-меров s и s^{rc} можно хранить только один, определяемый по некоторому правилу — например, таким правилом может быть выбор лексикографически минимального *(k+1)*-мера. В этом случаем необходимый объем памяти уменьшается примерно в два раза для четных k и ровно в два раза — для нечетных (только для четных k бывают обратно комплементарные себе *(k+1)*-меры).

В хеш-таблице значению хеш-функции h_x для p-мера сопоставляется сколько раз он встречался в исходных данных, то есть его вес. В качестве хеш-функции h_x от p-мера a используется минимум из значений полиномиальной хеш-функции h_p с простым основанием z по модулю $M=2^{64}$ от a и от a^{rc}.

Такое представление позволяет, во-первых, по заданному p-меру e, соответствующему ребру графа де Брёйна, получать его вес. Для этого достаточно просто вычислить значение хеш-функции $h_x(e)$ и проверить, присутствует ли оно в хеш-таблице. Во-вторых, по заданному k-меру v, соответствующему вершине графа, возможно получить все ее входящие и исходящие ребра. Для этого достаточно перебрать по какому из четрыех нуклеотидов будет выполнен переход в графе и проверить, есть ли в графе ребро с соответствующей пометкой. Использование полиномиального хеширования позволяет добавлять и удалять символы в конец или начало строки за время $O(1)$, что, совместно с амортизированной оценкой $O(1)$ на время доступа к хеш-таблице, влечет такую же амортизированную оценку на время перехода по ребру.

Таким образом, этот подход позволяет использовать любые значения k при небольшом использовании памяти. Единственным ограничением является то, что для достаточно небольшого числа коллизий число различных $(k+1)$-меров не должно быть больше 2^{32}.

2.3 Построение графа де Брёйна

Из-за наличия ошибок в чтениях возникает большое число ошибочных *p*-меров, встречающихся в чтениях только один раз. При сборке квазиконтигов эти *p*-меры практически не несут полезной информации, но занимают много места, поэтому в предлагаемом методе в качестве ребер графа де Брёйна используются только *p*-меры, встречающиеся не меньше *h* раз (на правктике обычно используются два, но при большом покрытии генома чтениями может быть увеличено).

Для того чтобы было возможно построить граф де Брёйна в случае, когда оперативной памяти хватает только для хранения ребер графа, но не хватает для хранения весов всех *p*-меров, применяется подсчет весов в несколько проходов по чтениям. Все *p*-меры разделяются на несколько групп в зависимости от префикса так, чтобы внутри одной группы было возможно хранить в памяти веса всех *p*-меров. За один проход по чтениям выполняется подсчет весов *p*-меров в одной группе, после чего эти веса записываются в файл. После выполнения всех проходов выполняется проход по файлам с весами и в хеш-таблицу с ребрами графа добавляются только *p*-меры с весами не меньше *h*.

2.4 Оценка «правдоподопности» пути

Пусть s – строка из нуклеотидов. Будем обозначать строку, обратно-комплементарную s как s^{rc}.

Рассмотрим парные чтения r_1 и r_2 длины l_r, s_1 и s_2 – строки их нуклеотидов ($s_{i,j}$ – j-й нуклеотид i-го чтения), q_1 и q_2 – векторы вероятностей ошибок ($q_{i,j}$ – вероятность неправильного прочтения в j-й позиции i-го чтения. Пусть a_1 и a_2 – их первые k-меры, а p – путь длины $(l-k)$, соединяющий в рассматриваемом графе a_1 и a_2^{rc}. Этому пути естественным образом соответствует строка s длины l, полученная конкатенацией a_1 и $(l-k)$ символов, последовательно стоящих на ребрах

этого пути. В дальнейшем, будем это преобразование пути в строку будет обозначать *s(p)*.

Необходимо оценить «правдоподобность» утверждения, что s является последовательностью нуклеотидов в том фрагменте, который породил чтения r_1 и r_2. Для этого сэмулируем безошибочные чтения $\tilde{s}_1$ и $\tilde{s}_2$ – в качестве $\tilde{s}_1$ возьмем первые l_r символов s, а в качестве $\tilde{s}_2$ – обратно-комплементированные последние l_r символов. Посчитаем вероятность того, что при чтении были допущены именно такие ошибки, обозначим это событие как $R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2}$. Событие, произошедшее в отдельной позиции j чтения i, обозначим $R'_{i,j}$. Если исходить из того, что ошибки в каждой позиции возникают независимо (что в действительности не совсем так, но для оценки этого достаточно), то вероятность $P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2})$ будет находиться по формуле:

$$P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2}) = \prod_{i=1}^{2} \prod_{j=1}^{|\tilde{s}_1|} P(R'_{i,j}), \qquad (2.1)$$

где

$$P(R'_{i,j}) = \begin{cases} q_{i,j}, & если\ \tilde{s}_{i,j} \neq s_{i,j}, \\ (1 - q_{i,j}) / 3, & иначе. \end{cases}$$

Если вероятность (2.1) для некоторого пути p много меньше вероятности для наиболее вероятного на данный момент пути, то путь p можно отбросить как неправдоподобный.

2.5 Алгоритм поиска путей

Для начала введем некоторые обозначения. Пусть путь p_1 длины l_1 ведет из вершины v_1 в вершину v_2, а путь p_2 длины l_2 ведет из вершины v_2 в вершину v_3. Будем обозначать конкатенацию этих путей, то есть путь длины $l_1 + l_2$, соединяющий вершины v_1 и v_3, проходящий сначала по пути p_1, а затем – по p_2, как $p_1 \cdot p_2$. Рассмотрим множества путей P_1 и P_2. С помощью $P_1 \cdot P_2$ будем обозначать все пути, которые можно получить конкатенацией путей $p1$ и p_2 из P_1 и P_2 соответственно, то есть

$P_1 \cdot P_2 = \{ p_1 \cdot p_2 \,|\, p_1 \in P_1, p_2 \in P_2 \}$. Заметим, что конкатенировать можно только те пути p_1 и p_2, у которых последняя вершина p_1 совпадает с первой вершиной p_2. Если P_1 состоит из одного пути $v_1 \to v_2$, а множество P_2 состоит из путей $v_2 \to v_3$ и $v_4 \to v_5$, то множество $P_1 \cdot P_2$ будет состоять из одного пути $v_1 \to v_3$.

Теперь рассмотрим задачу поиска путей, соединяющих две заданные вершины v_{start} и v_{end}, длины которых лежат в промежутке $[l_{\min}; l_{\max}]$. Будем обозначать множество всех путей из v_{start} в v_{end} длины l как P^l, тогда искомое множество всех путей из v_{start} в v_{end} будет получаться объединением множеств P^l:

$$P = \bigcup_{l=l_{\min}}^{l_{\max}} P^l .$$

Для поиска путей будем применять двунаправленный поиск, в котором происходит одновременный поиск путей, ведущих из первой вершины, и путей, ведущих во вторую вершину. Это позволяет сократить время работы с $O(d^{l_{\max}})$ до $O(d^{l_{\max}/2})$, где d – средняя степень вершины графа. Обозначим множество всех путей длины l_1, ведущих из первой вершины, за $P_1^{l_1}$, а множество всех путей длины l_2, ведущих во вторую вершину за $P_2^{l_2}$. Применение этого приема к решаемой задаче обусловлено тем, что верно равенство $P^l = P_1^{l_1} \cdot P_2^{l_2}$ для всех $l_1 \in \{0, 1, \cdots, l\}$ и $l_2 = l - L_1$. Действительно, любой путь p длины l из v_1 в v_2 можно разбить на два более коротких пути p_1 и p_2 длиной l_1 и l_2 соответственно, $p = p_1 \cdot p_2$, $l = l_1 + l_2$, верно и обратное – любые два пути p_1 из $P_1^{l_1}$ и p_2 из $P_2^{l_2}$, если их можно конкатенировать, после конкатенации образуют путь из v_1 в v_2 длины $l = l_1 + l_2$.

Для реализации такого подхода удобно запустить одновременно два обхода в ширину: из первой вершины по прямым ребрам и из второй – по обратным. Тогда на каждом шаге l можно поддерживать следующий инвариант: для первой вершины будем хранить множество $P_1^{l_1}$ всех исходящих из нее путей длины l_1, а для второй – множество $P_2^{l_2}$ всех входящих путей длины l_2, причем $l_1 + l_2 = l$. Таким образом, на l-ом шаге

мы можем получить все пути длины l из v_1 в v_2 путем конкатенацией путей из множеств $P_1^{l_1}$ и $P_2^{l_2}$:

$$P^l = P_1^{l_1} \cdot P_2^{l_2}.$$

На начальном шаге этого алгоритма l, l_1 и l_2 равны нулю, а P_1^0 и P_2^0 содержат по одному пути нулевой длины (эту пути состоят, соответственно, из вершин v_{start} и v_{end}).

Если E – это множество всех ребер графа, то шаг в первом обходе осуществляется по формуле:

$$P_1^{l_1+1} = P_1^{l_1} \cdot E,$$

а шаг во втором обходе по формуле:

$$P_2^{l_2+1} = E \cdot P_2^{l_2}.$$

Для перехода к следующей итерации необходимо выбрать, в каком из обходов делать шаг. Самым простым является поочередной переход, например, когда номер итерации l четный, делать шаг в первом обходе, а когда нечетный – во втором. Для более эффективного использования памяти и времени лучше производить увеличение в том обходе, в котором на данный момент $P_i^{l_i}$ в каком-то смысле меньше. Сравнение может происходить, например, по числу путей или, если использовать для хранения путей структуру, описанную в разд. 2.6, по числу различных концевых вершин.

2.6 Хранение путей

Самым простым способом хранения путей является множество, состоящего из строки $s(p)$ для каждого пути p. К сожалению, часто у одного пути возникает несколько продолжений, из-за чего строки приходится копировать, что требует $O(l)$ времени, где l – это длина строки (если бы продолжений было бы не больше одного, то при соответствующей реализации строк на добавление одного символа

требовалось бы амортизировано *O(1)* времени). Этого можно избежать, если для хранения путей использовать граф.

Рассмотрим множество $P_1^{l_1}$ всех путей из v_1 длиной l_1. Построим граф, в котором есть вершины вида *(v, l)* тогда и только тогда, когда в исходном графе де Брёйна существует путь из v_1 в v длиной l. Направленным ребром в этом графе будет соединены вершины *(u, l)* и *(v, l + 1)* тогда и только тогда, когда в исходном графе де Брёйна есть ребро $u \to v$. Будем называть множество всех вершин вида *(v, l)* слоем с номером l. Отметим, что для любого пути из v_1 в v длины l_1 есть соответствующий путь из $(v_1, 0)$ в (v, l_1). Верно и обратное, любому пути из $(v_1, 0)$ в (v, l_1) естественным образом соответствует путь в исходном графе из v_1 в v длины l_1. Обновление этого графа до графа путей длины l_1+1 требует $O(n)$ времени, где n – число конечных вершин, то есть вершин в l_1-ом слое. Для учета «правдоподобности» будем для каждой вершины *(v, l)* хранить информацию о значении $L_{s_1,s_2}(s(p),\varepsilon)$ и информацию, необходимую для пересчета этого значения для пути на единицу большей длины, только для одного пути p, который определяется по индукции из предыдущих вершин. Для вершины $(v_1, 0)$ – это путь состоящий из одной вершины v_1. Для вершины *(v, l)* рассмотрим все пути, заканчивающиеся в этой вершине. Из них выберем только те, префикс которых хранится в одной из вершин слоя $l - 1$. Из оставшихся путей выберем тот, значение $L_{s_1,s_2}(s(p),\varepsilon)$ для которого ближе к нулю, будем его обозначать за $L(v, l)$. При образовании нового слоя, если в какой-то вершине значение $L(v, l_1 + 1)$ будет не меньше чем порог L_{max}, то удалим эту вершину. Заметим, что такое отбрасывание не является эквивалентным отбрасыванию только «неправдоподобных» путей, а является только приближением.

Для множества $P_2^{l_2}$ построение аналогично. В качестве вершин используются пары вида *(v, l)* тогда и только тогда, когда есть путь длины l из v в v_2. Вершины *(u, l + 1)* и *(v, l)* соединены ребром, если в графе де

Брёйна есть ребро $u \to v$. В качестве $L(v, l)$ используется значение $L_{s_1,s_2}(\varepsilon, s(p))$ для некоторого пути p определяемого из индукции, аналогичной предыдущему случаю.

2.7 Функция похожести путей

Два пути называются *похожими*, если можно выбрать такой набор простых исправлений (замены, вставки или удаления символа), что на любой *k*-мер этих путей приходилось не больше заданного числа исправлений. В отличие от простого ограничения на расстояние Левенштейна между путями, такой выбор позволяет, во-первых, избежать концентрирования ошибок в одной части и, во-вторых, позволяет сравнивать части путей независимо. Рассмотрим пример сжатого графа де Брёйна, изображенный на рис. 4. Пусть путь *B* отличается от пути *C* на один нуклеотид, а *E* сильно отличается от *F*. Рассмотрим использование расстояние Левенштейна для определения похожести путей. По этому определению пути *ABD* и *ACD* являются похожими, и, следовательно, один из этих путей возможно удалить, практически не теряя информации. С другой стороны, пути *ABDEG* и *ACDFG* не являются похожими, и все ребра несут информацию, поэтому удалять их нельзя. Таким образов, при использовании расстояния Левенштейна для определения похожести может возникнуть противоречие. Если же рассматривать предложенный метод определения похожести, то можно четко определить, что пути *ABDEG* и *ACDFG* отличаются только из-за непохожести путей *E* и *F*. В этом случае путь *C* можно будет удалить, при этом будет выполняться свойство, что любой путь в исходном графе похож на какой-нибудь путь в получившемся.

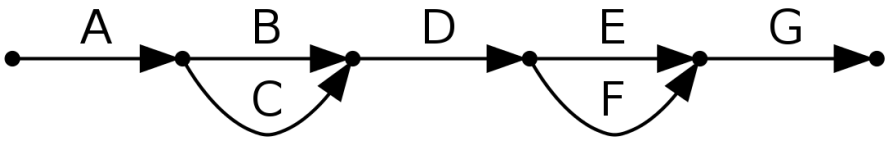

Рисунок 4 – Похожесть путей

2.8 Алгоритм нахождения консенсуса среди полученных путей

После того, как был получен граф, содержащий все возможные пути между вершинами v_{start} и v_{end}, необходимо убрать ошибки и небольшие реальные различия в последовательностях. Более формально, требуется выбрать такой подграф, что любой путь в исходном графе будет похож на какой-нибудь из путей в этом подграфе. При этом требуется, с одной стороны, минимизировать число оставшихся ребер, с другой – эти ребра должны быть достаточно большого веса.

Эта задача решается с помощью эвристического алгоритма в несколько итераций. На каждой итерации некоторые ребра помечаются, что означает, что они останутся в окончательном графе, а некоторые – удаляются. А именно, на каждой итерации выбирается путь между вершинами v_{start} и v_{end} максимального веса, содержащий хотя бы одно непомеченное ребро. Ребра на этом пути помечаются. Затем, находятся непомеченные ребра, все пути по которым похожи на какой-нибудь путь по помеченным ребрам, и эти ребра удаляются. На рис 5 показан пример работы этого алгоритма.

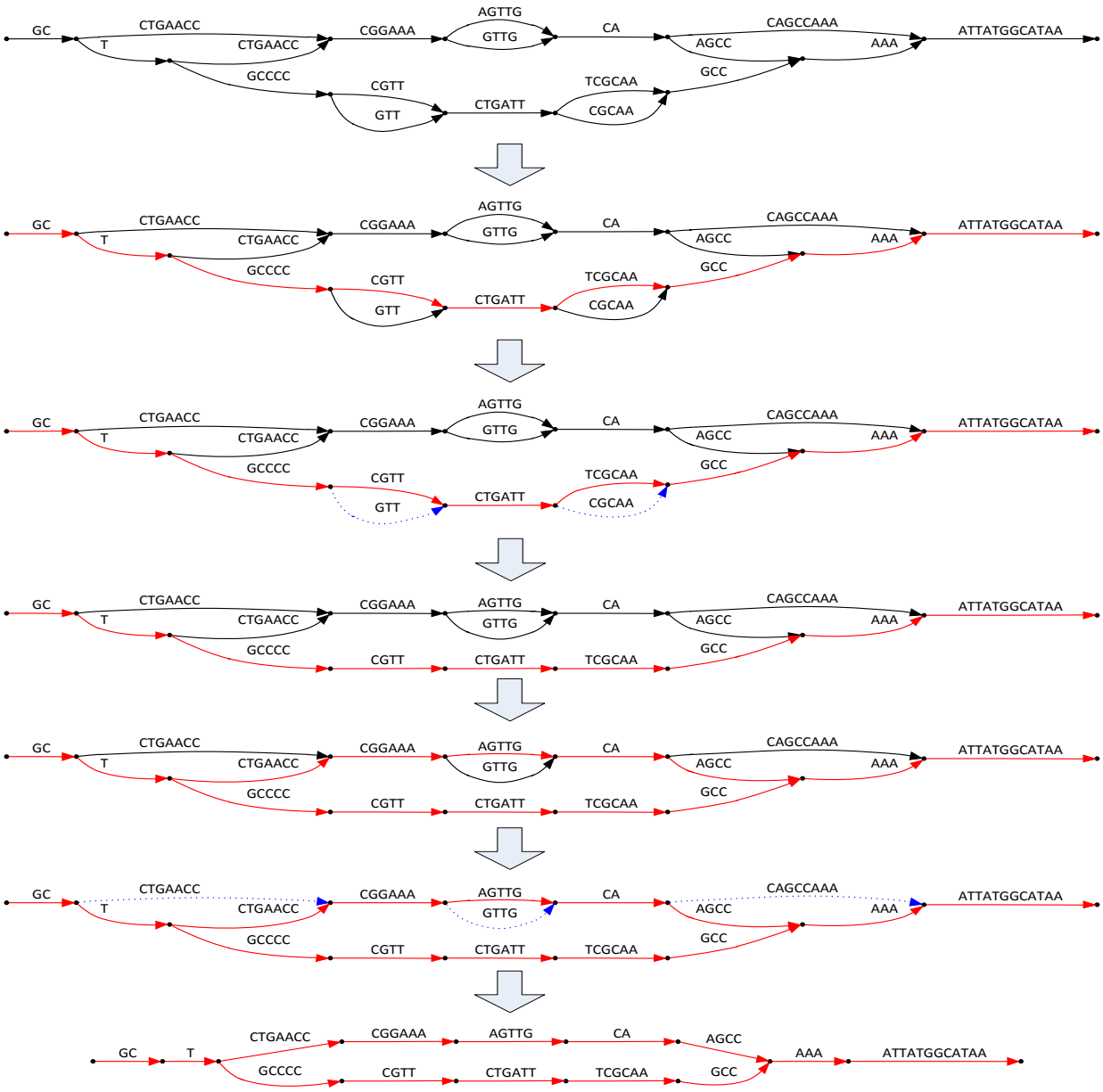

Рисунок 5 – Пример работы алгоритма нахождения консенсуса

Для того, чтобы найти ребра, все пути через которых похожи на какой-нибудь путь по помеченным ребрам, вводится булева функция `anyPathFromXHaveSimilarGoodPathFromY(x, y, doneErrors)`, которая возвращает `true`, только если любой путь, начинающийся с `x` (вершины или ребра), похож на какой-нибудь путь по помеченным ребрам, начинающийся с `y` (тоже вершины или ребра), при том, что уже сделаны ошибки, позиции которых описаны в `doneErrors`. Далее в данном разделе для краткости будем обозначать эту функцию как

`f`. Описание ошибок `doneErrors` – это список относительных позиций ошибок, которые удалены от текущей позиции не больше чем на *k*.

В самом начале функции `f` выполняется проверка, что число ошибок в `doneErrors` не превышает заданного порога (на практике используется значение около пяти), если превышает, то возвращается значение `false`. Затем отсекается второй граничный случай: если `x` и `y` совпадают с v_{end}, возвращается значение `true`.

Теперь рассмотрим общий случай, когда и `x` и `y` – это вершины, и они обе имеют исходящие ребра (для `y` – помеченные). Проверяется, что для каждого исходящего ребра `ex` из `x` верно, что любой путь, начинающийся с `ex`, похож на какой-нибудь путь по помеченным ребрам, начинающийся с y, причем уже сделаны ошибки `doneErrors`, то есть верно `f(ex, y, doneErrors)`. После того, как было выбрано ребро `ex`, с которого начинается путь из `x`, проверяются, что хотя бы для одного ребра `ey` из `y` верно `f(ex, ey, doneErrors)`. Заметим, что этот переход не точный, в том смысле, что возможна ситуация, когда одному из путей, начинающихся с `ex`, будет соответствовать путь, начинающийся с `ey1`, а другому – с `ey2`. На рис. 6 показан пример такой ситуации. В этом примере последовательность *A* имеет небольшую длину так, что *AC* похоже на *C*, последовательность B похожа на последовательность *b* но так, что *B* не похожа на *Ab*.

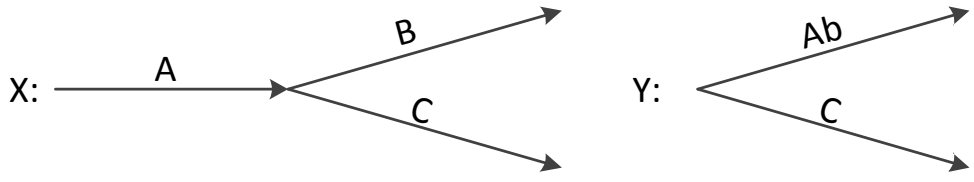

Рисунок 6 – Пример ситуации, когда в Y нельзя выбрать единственное исходящее ребро (латинскими буквами обозначены нуклеотидные последовательности)

В случае, когда зафиксированы ребра `ex` и `ey`, возможны четыре варианта:

а) нуклеотиды на этих ребрах совпадают и верно `f(ex.to, ey.to, doneErrors.appendMatch())`, где `e.to` означает конечную вершину ребра `e`, а `doneErrors.appendMatch()` возвращает новое описание ошибок со сдвигом номеров всех ошибок в `doneErrors` без добавления новой ошибки;

б) нуклеотиды на этих ребрах не совпадают и верно `f(ex.to, ey.to, doneErrors.appendError())`, где `doneErrors.appendError()` означает сдвиг номеров всех ошибок в `doneErrors` с добавлением новой с относительным сдвигом равным нулю;

в) верно `f(ex.to, ey, doneErrors.appendError())`;

г) верно `f(ex, ey.to, doneErrors.appendError())`.

Первый вариант соответствует отсутствию ошибки в этой позиции, второй вариант соответствует ошибке замены, два последних варианта соответствуют ошибкам вставки и удаления. Если хотя бы один из этих вариантов выполняется, то утверждение проверяемое функцией оказывается верным и возвращается значение `true`, в противном случае, возвращается `false`.

На каждом рекурсивном вызове выполняется продвижение вперед как минимум от одной вершины. Так как рассматриваются, только ребра и вершины, лежащие на пути от v_{start} до v_{end}, то эта функция завершится за конечное время либо из-за того, что и `x`, и `y` совпадают с v_{end}, либо из-за того, что число ошибок превысит порог.

Для того чтобы удалить лишние ребра, описанная функция вызывается в вершинах, в которых есть как помеченные, так и непомеченные ребра. Для каждого непомеченного ребра `ex` вычисляется

значение функции `f(ex, ex.from)`. Если эта функция вернула `true`, то это означает, что через ребро ex не проходит «интересных» путей, следовательно, это ребро можно удалить.

2.9 Качество квазиконтигов

Про качество получающихся квазиконтигов можно сделать следующее утверждение. Зафиксируем *k*. Пусть для случайной пары чтений:

а) вероятность того, что первый *k*-мер первого чтения и последний *k*-мер второго содержат ошибки, не больше ε_1;

б) вероятность того, что хотя бы один из геномных *k*-меров соответствующего этой паре чтений фрагмента покрыт меньше *h* раз, где *h* – порог на добавление в граф де Брёйна, не больше ε_2;

в) вероятность того, что пара чтений является химерной не больше ε_3;

г) вероятность того, что длина исходного фрагмента больше чем L_{max}, не больше ε_4.

Утверждение. При выполнении этих условий вероятность того, что для этой пары будет получен квазиконтиг и он будет непохож на соответствующую геномную последовательность, не превосходит ε, где $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_4$.

Доказательство. Рассмотрим случайную пару чтений. С вероятностью не меньшей $1 - \varepsilon_1 - \varepsilon_2 - \varepsilon_3 - \varepsilon_4 = 1 - \varepsilon$ для этих чтений не случится ни одного из событий а)–г). Так как пара чтений не будет химерной, длина исходного фрагмента будет не больше L_{max}, весь он будет покрыт k-мерами из графа де Брёйна и крайние *k*-меры чтений безошибочные, то в графе со всеми путями длины не большей L_{max}-*k* будет присутствовать путь, соответствующий исходному фрагменту генома. Возможны два варианта: либо по этому графу удалось восстановить

квазиконтиг, либо нет. То, что квазиконтиг удалось восстановить, означает, что после удаления лишних ребер в графе путей остался ровно один путь. Так как предложенный алгоритм удаления лишних ребер оставляет граф, в котором любой путь в исходном графе будет похож на какой-нибудь путь в полученном графе, то полученный квазиконтиг будет похож на соответствующий фрагмент генома, потому что, как было показано, путь для него содержится в исходном графе. Это означает, что полученный квазиконтиг может быть непохож ни на один из фрагментов геном, только если выполнилось одно из событий а)–г), вероятность чего не больше ε.

2.10 Алгоритм автоматического определения библиотек парных чтений

Предлагаемый метод принимает на вход парные библиотеки, для которых известны их параметры: распределение чтений по файлам, направление чтений, параметры распределения длин фрагментов. Для удобства использования, предлагается алгоритм, позволяющий принимать только набор из всех файлов и определять все параметры библиотек автоматически.

После построения графа де Брёйна, алгоритм перебирает все пары файлов с близкими именами и запускает сборку квазиконтигов для первой тысячи пар чтений. При этом перебираются все варианты взаимных направлений чтений, а в качестве максимальной длины фрагмента выбирается число, заведомо большее, чем возможная длина (на практике выбирается значение 10000). Затем для каждого файла ставится в соответствие файл и направление чтений, для которых было собрано больше всего квазиконтигов (и больше порога в 5%). Если для двух файлов каждый из них соответсвует другому и направления чтений согласованны, то эти два файла считаются парными. Затем для каждой библиотеки по

распределению длин собравшихся квазиконтигов можно с достаточной точностью восстановить параметры распределения длин фрагментов.

2.11 Расширение алгоритма

Предложенный алгоритм может быть расширен на работу не только с графом путей, соединяющих две вершины, а со всем графом де Брёйна. Алгоритм упрощения графа будет отличаться из-за того, что в графе де Брёйна сложнее определить две достаточно удаленные точки, через которые, скорее всего, проходит геномный путь. Соответственно необходимо использовать другой алгоритм определения помечаемых путей. Важно, чтобы при этом для любой вершины, в которую входит помеченное ребро, либо отсутствовали исходящие ребра, либо было бы исходящее помеченное ребро. Кроме того, необходимо изменить граничные условия для выхода из функции `anyPathFromXHaveSimilarGoodPathFromY`, а именно, вместо проверки условия `x=y=`v_{end}, надо проверить выполнение одного из двух вариантов, в котором следует выйти из функции, вернув значение `true`:

а) `x=y` – по свойству из радела 2.7;

б) у `x` нет исходящих ребер – для пути нулевой длины всегда существует похожий на него путь длины ноль.

2.12 Выбор значения параметра *k*

Выбор значения параметра *k* является важным аспектом предложенного метода. С одной стороны, чем больше *k*, тем меньше повторов длины большей *k* и, соответственно, граф де Брёйна получается более простым. С другой стороны, при б*о*льших значениях *k* увеличивается вероятность *k*-мера генома быть непокрытым из-за конечности покрытия генома чтениями и наличия ошибок. В работе [17] было предложено выбирать *k* для которого в чтениях присутствует наибольшее число геномных *k*-меров. В этой работе описан программный

инструмент *Kmergenie*, реализующий этот метод. По результатам экспериментальных исследования (см. раздел 3) оказалось, что выбор значения k этим инструментом в случае чтений без ошибок вставки и удаления дает значение k, близкое к оптимальному для предлагаемого метода сборки квазиконтигов. К сожалению, для чтений с ошибками вставки и удаления инструмент *Kmergenie* дает плохой результат.

2.13 Распределенная сборка квазиконтигов

Предлагаемый метод можно использовать для распределенной сборки. Для этого необходимо разбить чтения на группы, так чтобы они перекрывались. Для этого для всех пар чтений вычислим, сколько k-меров входят как в первое чтение, так и во второе. Чем больше таких общих k-меров, тем более вероятно, что позиции, из которых были прочитаны эти чтения, находятся в геноме на небольшом расстоянии друг от друга.

Построим взвешенный граф, вершины которого будут соответствовать чтениям, а ребро между двумя вершинами будет, если у соответствующих чтений есть общие k-меры. Весом ребра будет количество общих k-меров.

Например, для чтений AAGA, ATAG, GTAC, ATAC, TGAC, TTGA количество различных 2-меров приведено в табл. 2. Граф для этих чтений приведен на рисунок 7.

Таблица 2 – Количество общих *2*-меров, для чтений

	AAGA	**ATAG**	**GTAC**	**ATAC**	**TGAC**	**TTGA**
AAGA	-	1	0	0	1	1
ATAG	1	-	1	2	0	0
GTAC	0	1	-	2	1	0
ATAC	0	2	2	-	1	0
TGAC	1	0	1	1	-	2
TTGA	1	0	0	0	2	-

Для работы последующих этапов алгоритма важно, чтобы чтения, являющиеся концами одного и того же парного чтения попали в одну компоненту. Для этого они рассматриваются, как одно чтение и граф строится между парами таких чтений.

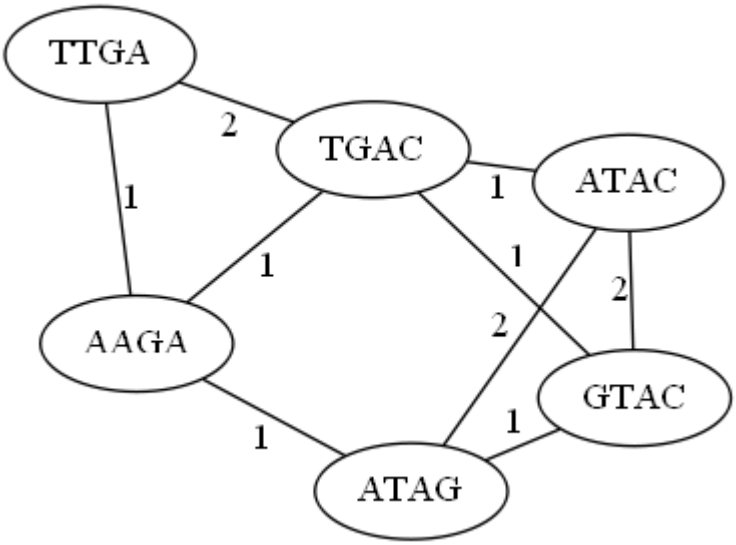

Рисунок 7 – Граф для чтений AAGA, ATAG, GTAC, ATAC, TGAC, TTGA

Далее граф разбивается на несколько компонент таким образом, чтобы максимальный вес ребра, концы которого находятся в разных компонентах, был минимальным; при равном максимальном весе ребра вес следующего такого ребра должен быть минимальным, и так далее.

При этом максимальный размер компоненты не должен быть больше некоторой величины, так как на последующих этапах сборки генома из

чтений одной компоненты будут собираться квазиконтиги. Для решения этой задачи будем применять алгоритм, похожий на алгоритм Краскала построения минимального остовного дерева [18].

Для приведенного на рис. 7 графа разбиение на компоненты размера два показано на рис. 8 (вершины первой компоненты обозначены белым цветом, второй компоненты – светло-серым, третьей компоненты – темно-серым). При этом максимальное ребро с концами в разных компонентах имеет вес 2, а второе по величине такое ребро имеет вес 1.

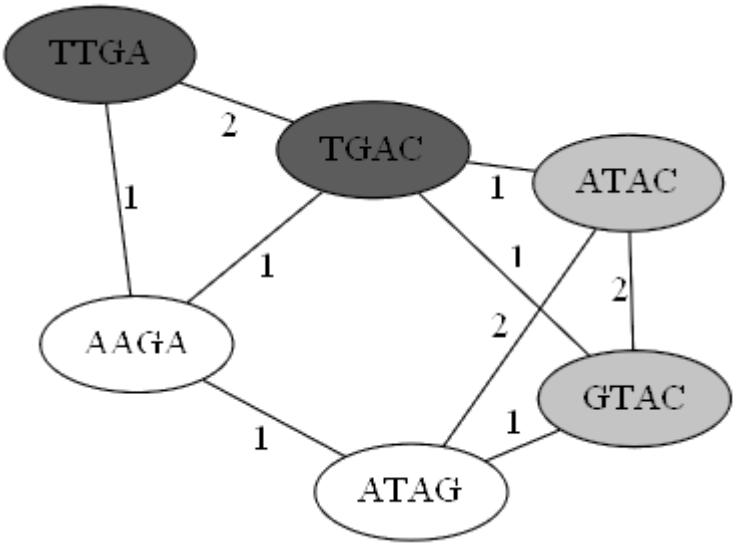

Рисунок 8 – Разбиение на компоненты

В каждой компоненте связности получившегося графа будет запущен алгоритм восстановления квазиконтигов.

2.14 Выводы по главе 2

В данной главе был рассмотрен предлагаемый алгоритм сборки, было доказано утверждение про качество получающихся квазиконтигов. Также были расмотрены вопросы автоматического определения параметров библиотек и выбора значения параметра *k*.

3 Экспериментальные исследования

Предложенный алгоритм был реализован на языке программирования *Java* и был протестирован на двух наборах данных бактериальных геномов *E. Coli* и *P. Stutzeri*.

3.1 Набор данных для генома *E. Coli*

Геном *E. Coli* известен и имеет длину 4,6 миллиона нуклеотидов. Для тестирования предложенного метода была выбрана библиотека парных чтений *SRR001665*, полученная с помощью секвенатора компании *Illumina*. Длина чтений составляет 36 нуклеотидов, средний размер фрагмента – около 200, число чтений – 10 миллионов. Чтобы повысить точность анализа результатов сборки квазиконтигов референсный геном *K-12 MG1655* был вручную модифицирован так, чтобы реальные различия между этим геномом и геномом, из которого была получена библиотека, не учитывались. Для этого было произведено несколько однонуклеотидных замен и несколько вставок повторяющихся фрагментов на основе анализа картирования чтений на геном. Получившийся геном доступен по адресу [19].

В эксперименте был проведен анализ качества квазиконтигов в зависимости от значения параметра *k*. Для этого для всех *k* из множества {17, 21, 25, 29, 33} была запущена сборка с ограничением на максимальный квазиконтиг 500 с использованием 4 ГБ оперативной памяти. Время работы составляло около 40 минут.

В каждом случае были определены и записаны следующие характеристики:

а) доли исходов сборки квазиконтигов: какая часть чтений была успешно восстановлена (строка «Один путь»), для какой части число не было путей (строка «Нет путей»), для какой – число путей было слишком большим (строка «Много путей»), для

какой – не получилось упростить граф до одного пути (строка «несколько путей»;

б) доля генома, не покрытая квазиконтигами и число непокрытых участков;

в) насколько хорошо квазиконтиги картировались на референсный геном: считалась доля числа квазиконтигов, в которых доля правильных нуклеотидов (*качество картирования*) была больше заданного порога (картирование производилось с помощью программы `blastn`, вызванной с параметрами `-dust no -gapopen 10 -gapextend 2`);

г) значение *N*50 для контигов, собранных из этих квазиконтигов.

Результаты приведены в таблице 3.

Таблица 3 – Результаты эксперимента по сборке генома *E. Coli* для разных значений *k*.

	k=17	*k*=21	*k*=25	*k*=29	*k*=33
Один путь	88,3%	**90,5%**	88,3%	84,7%	69,6%
Много путей	5,2%	1,1%	**0,0%**	**0,0%**	**0,0%**
Несколько путей	1,5%	1,2%	1,8%	1,4%	**1,0%**
Нет путей	**5,0%**	7,2%	9,9%	13,9%	29,3%
Не покрыто генома	1,35%	0,56%	0,39%	**0,33%**	1,35%
Число непокрытых участков	530	217	153	**134**	604
Доля чтений с качеством картированния меньше 100%	0,66%	0,51%	0,40%	0,36%	**0,32%**
Доля чтений с	0,17%	0,15%	0,11%	0,10%	**0,08%**

качеством картированния меньше 99%					
Не картировано	210	143	136	76	**0**
N50	$8{,}6 \cdot 10^3$	$18{,}9 \cdot 10^3$	$24{,}0 \cdot 10^3$	$\mathbf{24{,}3 \cdot 10^3}$	$2{,}0 \cdot 10^3$
Число контигов	981	453	408	**400**	3363
Суммарная длина контигов	$4{,}54 \cdot 10^6$	$4{,}58 \cdot 10^6$	$4{,}58 \cdot 10^6$	$4{,}58 \cdot 10^6$	$4{,}57 \cdot 10^6$

Некартированные чтения в основном представляют собой химерные чтения из фрагмента генома *E. Coli* с координатами 1207000–1210000. Эти квазиконтиги были получены из химерных парных чтений.

При запуске инструмента *Kmergenie* на этих данных получилось, что больше всего геномных *k*-меров должно получиться при *k*=27, что достаточно хорошо соответствует полученными результатами.

Для сравнения качества сборки квазиконтигов на этих же данных был запущен сборщик *GapFiller*. Время работы составило 27 часов. Было использовано 7 ГБ оперативной памяти.

Результаты сравнения приведены в таблице 4.

Таблица 4 – Результаты сравнения сборки квазиконтигов генома *E. Coli*.

	Предлагаемый алгоритм	*GapFiller*
Восстановленно	84,7%	**93**%
Не покрыто генома	0,33%	**0,29%**
Число непокрытых участков	134	**126**

Доля чтений с качеством картированния меньше 100%	**0,36%**	0,74%
Доля чтений с качеством картированния меньше 99%	0,10%	0,15%
Не картировано	**76**	211

Таким образом, сборщик GapFiller чуть выигрывает в покрытии генома квазиконтигами, но требует больше вычислительных ресурсов и обладает чуть меньшим качеством получающихся квазиконтигов.

3.2 Набор данных для генома *P. stutzeri*

Геном *P. Stutzeri* имеет длину около 5 миллиона нуклеотидов. Имелось три библиотеки чтений. Две библиотеки одиночных чтений: библиотека из 2,3 миллионов чтений со средней длиной чтения 100 и библиотека из 2,7 миллионов чтений со средней длиной чтения 200. Третья библиотека состояла из 2,1 миллионов парных чтений со средней длиной 80 нуклеотидов и средним размеров вставки 2000 нуклеотидов. Непарные чтения использовались только для построения графа де Брёйна.

Сборка квазиконтигов была запущена для k из {25, 29, 33, 37, 41, 45}. Сборка производилась из неисправленных чтений из всех библиотек. Так как отсутствует хороший референсный геном результаты проверки качества квазиконтигов отсутствуют. Соответственно, в таблице 5 приведены статистики по разным случаям, получившимся при сборке и параметры контигов, полученных из этих данных.

Таблица 5 – Результаты эксперимента по сборке генома *P. Stutzeri* для разных значений *k* из всех неисправленных чтений.

	k=25	*k*=29	*k*=33	*k*=37	*k*=41	*k*=45
Один путь	**47,2%**	44,9%	41,2%	37,5%	33,7%	29,9%
Много путей	2,8%	**0%**	**0%**	**0%**	**0%**	**0%**
Несколько путей	7,1%	6,9%	5,2%	4,1%	3,1%	**2,5%**
Нет путей	**34,1%**	37,4%	40,4%	42,6%	44,7%	46,1%
N50	$14{,}3 \cdot 10^3$	$15{,}8 \cdot 10^3$	$16{,}0 \cdot 10^3$	$\mathbf{16{,}4 \cdot 10^3}$	$13{,}5 \cdot 10^3$	$12{,}6 \cdot 10^3$
Число контигов	486	**473**	499	494	576	598
Суммарная длина контигов	$4{,}67 \cdot 10^6$	$4{,}71 \cdot 10^6$	$4{,}79 \cdot 10^6$	$4{,}75 \cdot 10^6$	$4{,}91 \cdot 10^6$	$4{,}89 \cdot 10^6$
Максимальный контиг	$50{,}9 \cdot 10^3$	$57{,}1 \cdot 10^3$	$58{,}2 \cdot 10^3$	$56{,}8 \cdot 10^3$	$49{,}9 \cdot 10^3$	$\mathbf{66{,}9 \cdot 10^3}$

Также для тех же значений *k* была проведена сборка из исправленных чтений. Результаты приведены в таблице 6

Таблица 6 – Результаты эксперимента по сборке генома *P. Stutzeri* для разных значений *k* из всех неисправленных чтений.

	k=25	*k*=29	*k*=33	*k*=37	*k*=41	*k*=45
Один путь	63,3%	**63,5%**	62,7%	61,3%	58,9%	56,6%
Много путей	**0,0%**	**0,0%**	**0,0%**	**0,0%**	**0,0%**	**0,0%**
Несколько путей	11,6%	8,6%	6,7%	5,5%	4,9%	**4,2%**
Нет путей	**16,3%**	17,0%	17,4%	17,4%	17,6%	17,5%
N50	$19{,}5 \cdot 10^3$	$20{,}9 \cdot 10^3$	$\mathbf{22{,}8 \cdot 10^3}$	$21{,}0 \cdot 10^3$	$19{,}0 \cdot 10^3$	$18{,}5 \cdot 10^3$
Число контигов	368	355	**328**	349	376	399

Суммарная длина контигов	$4{,}65 \cdot 10^6$	$4{,}70 \cdot 10^6$	$4{,}67 \cdot 10^6$	$4{,}72 \cdot 10^6$	$4{,}71 \cdot 10^6$	$4{,}72 \cdot 10^6$
Максимальный контиг	$65{,}2 \cdot 10^3$	$71{,}4 \cdot 10^3$	$64{,}0 \cdot 10^3$	$63{,}7 \cdot 10^3$	$63{,}7 \cdot 10^3$	$\mathbf{76{,}2 \cdot 10^3}$

На рис. 9 приведен график зависимости N50 от *k* при сборке из исправленных и неисправленных чтений.

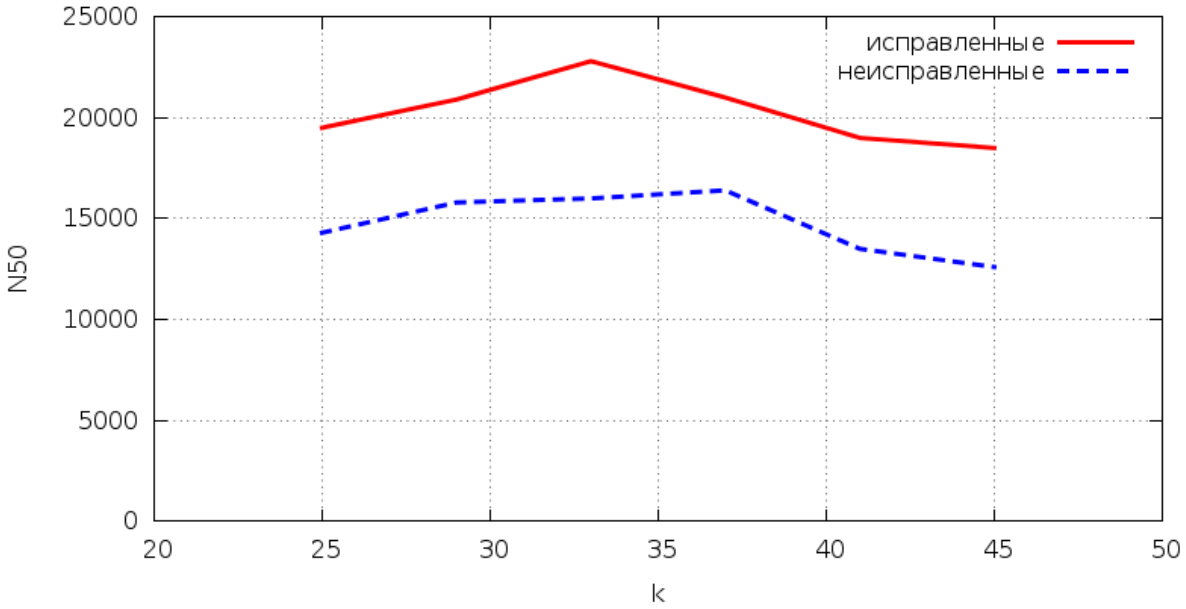

Рисунок 9 – Зависимость N50 от значения параметра k при сборке из исправленных и неисправленных чтений.

Инструмент *Kmergenie* на этих данных предложил использовать *k*=21, что получилось, что больше всего геномных k-меров должно получиться при *k*=27, что достаточно хорошо соответствует полученными результатами.

Было произведено сравнение лучших полученных контигов с контигами, полученными с помощью коммерческой программы *CLC Genomics Workbench*, которая на настоящий момент является одной из лучших программ для сборки генома из чтений с ошибками вставки и удаления [20]. Результаты сравнения приведены в таблице 7

Таблица 7 – Сравнение результаты по сборке генома *P. Stutzeri* с использованием предложенного метода и сборщиком CLC .

	ITMO	CLC
Число контигов	**390**	1784
Суммарная длина	$4{,}80 \cdot 10^6$	$5{,}17 \cdot 10^6$
N50	$\mathbf{22{,}3 \cdot 10^3}$	$18{,}7 \cdot 10^3$
Максимальный контиг	74,9	$\mathbf{96{,}8 \cdot 10^3}$

3.3 Выводы по главе 3

В настоящей главе были представлены результаты экспериментов. Было показано, что предлагаемый метод эффективнее своих аналогов на парных чтениях без ошибок вставки и удаления. На чтениях с ошибками вставки и удаления он позволяет получить результаты по некоторым параметрам превосходящие результаты сборки существующими методами.

Заключение

В работе был предложен метод учета парных информации для сборки геномов из чтений с ошибками вставки и удаления. Особенностями предложенного метода являются наличие оценки на качество получаемых результатов, достаточно небольшое требование к оперативной памяти, поддержка почти неограниченного размера *k*-мера и автоматического определения парных библиотек. Предложенный метод был реализован на языке программирования *Java* и экспериментально протестирован. Полученные результаты по некоторым параметрам превосходят результаты других сборщиков генома.

Список литературы

1. Сергушичев А.А. Разработка метода восстановления фрагментов нуклеотидной последовательности по парным чтениям. НИУ ИТМО, 2011. Бакалаврская выпускная квалификационная работа.
2. Watson J.D., Crick F.H. Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. // Nature. 1953. Vol. 171. No. 4356. pp. 737–738.
3. Sanger F., Nicklen S., and Coulson A.R. Dna sequencing with chain-terminating inhibitors. // Proc Natl Acad Sci U S A. Dec 1977. Vol. 74. No. 12. pp. 5463–5467.
4. Chinault A.C., Carbon J. Overlap hybridization screening: isolation and characterization of overlapping DNA fragments surrounding the leu2 gene on yeast chromosome III. // Gene. Feb 1979. Vol. 5. No. 2. pp. 111–126.
5. Staden R. A strategy of dna sequencing employing computer programs // Nucleic Acids Res. Jun 1979. Vol. 6. No. 7. pp. 2601–2610.
6. Quail M., al E. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers // BMC genomics. 2012. Vol. 13. No. 1. P. 341.
7. Myers E.W. The fragment assembly string graph // Bioinformatics. 2005. Vol. 21. No. supl 2. pp. ii79-ii85.
8. Medvedev P., Georgiou K., Myers G., and Brudno M. Computability of models for sequence assembly // Lecture Notes Comput. Sci. 2007. Vol. 4645. pp. 289–301.
9. Edmonds J., Johnson E.L. Matching, Euler tours and the Chinese postman // Mathematical Programming. 1973. Vol. 5. No. 1. pp. 88-124.
10. Nagarajan N., Pop M. Parametric complexity of sequence assembly: theory and applications to next generation sequencing // Journal of Computational Biology. 2009. Vol. 16. No. 7. pp. 897–908.

11. Pevzner P., Tang H., and Waterman M. An Eulerian path approach to DNA fragment assembly // Proc. Natl. Acad. Sci. USA. 2001. Vol. 98. pp. 9748–9753.
12. Magoč T., Salzberg S.L. FLASH: fast length adjustment of short reads to improve genome assemblies // Bioinformatics. 2011. No. 27(21). pp. 2957-2963.
13. Nadalin F., Vezzi F. P.A. GapFiller: a de novo assembly approach to fill the gap within paired reads // BMC Bioinformatics. 2012. No. 13(Suppl 14):S8.
14. Medvedev P., Pham S., Caisson M., Tesler G., and Pevzner P. Paired de Bruijn Graphs: A Novel Approach for Incorporating Mate Pair Information into Genome Assemblers // Journal Of Computational Biology. 2011. Vol. 18. No. 11. pp. 1625–1634.
15. Pham S.K., Antipov D., Sirotkin A., Tesler G., Pevzner P., and Alekseyev M. Pathset Graphs: A Novel Approach for Comprehensive Utilization of Paired Reads in Genome Assembly // Journal Of Computational Biology. 2012. Vol. 19. pp. 1–13.
16. Okanohara D., Sadakane K. Practical entropy-compressed rank/select dictionary. Computing Research Repository, 2006. arXiv:cs/0610001v1.
17. Chikhi R., Medvedev P. Informed and Automated k-Mer Size Selection for Genome Assembly. HiTSeq ed. 2013.
18. Кормен Т., Лейзерсон Ч., Ривест Р., and Штайн К. Алгоритмы: построение и анализ. Москва: Вильямс, 2011.
19. URL: http://genome.ifmo.ru/files/experiments/ecoli/coli_srr001665.fasta (дата обращения: 15.Июнь.2013).
20. // First look at Ion Torrent data: De novo assembly: [сайт]. URL: http:// pathogenomics.bham.ac.uk/blog/2011/05/first-look-at-ion-torrent-data-de-novo-assembly/ (дата обращения: 17.06.2013).