Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Александров Антон Вячеславович

Разработка метода исправления ошибок вставки и удаления в наборе чтений нуклеотидной последовательности

Научный руководитель: к.т.н., ассистент кафедры КТ НИУ ИТМО
Царев Федор Николаевич

Санкт-Петербург
2013

Введение

Актуальность темы. Многие современные задачи биологии и медицины требуют знания генома живых организмов, который состоит из нескольких нуклеотидных последовательностей ДНК. В связи с этим возникает необходимость в дешевом и быстром методе секвенирования, то есть определения последовательности нуклеотидов в образце ДНК.

Устройства, осуществляющие чтение нуклеотидных последовательностей, – секвенаторы – допускают при чтении ошибки, которые необходимо исправить. Большинство программных средств, осуществляющих сборку генома, оптимизированы для работы с данными, содержащими только ошибки замены. Однако в последнее время все большую популярность получают секвенаторы, совершающие ошибки вставки и удаления, в связи с чем необходимо разработать эффективный метод их исправления.

Объект исследования - задача исправления ошибок вставки и удаления в чтениях нуклеотидной последовательности.

Исследование состоит из следующих частей:

- разработать алгоритм эффективного исправления ошибок вставки и удаления в чтениях нуклеотидной последовательности;
- реализовать разработанный алгоритм в виде программного обеспечения ЭВМ;
- экспериментально проверить эффективность разработанного метода.

Научная новизна. Разработан новый метод исправления ошибок вставки и удаления в чтениях нуклеотидной последовательности.

Кроме того, разработанный метод реализован в виде программного обеспечения, с которым были проведены эксперименты, подтверждающие его работоспособность.

Теоретическая и практическая значимость. Разработанный метод может использоваться как часть процесса секвенирования генома.

Структура работы. Работа состоит из введения и четырех глав.

В первой главе приведен обзор предметной области — рассмотрены необходимые для понимания тематики основы биоинформатики, объяснена актуальность задачи, описаны основные моменты, необходимые для понимания процесса сборки генома. Также в первой главе дается набор терминов, используемых в работе, и их определений.

Во второй главе кратко описаны уже существующие методы решения исследуемой проблемы, а также описаны их недостатки.

В третьей главе описан предлагаемый метод.

В четвертой главе описаны детали реализации метода, рассмотрены итоги и результаты работы, а также поставлены направления дальнейшей работы.

1. Обзор предметной области

1.1. Биоинформатика

Современные задачи, возникающие в биологии и медицине, требуют работы с большим объемом данных. Поэтому применение вычислительных устройств и алгоритмов находит все более широкое применение в этих областях науки.

1.1.1. ДНК

ДНК (дезоксирибонуклеиновая кислота) — химическое вещество, биологический полимер, обеспечивающий хранение и передачу из поколения в поколение генетической информации. В клетках эукариотов (например, животных и растений) ДНК находится в ядре каждой клетки организма.

С химической точки зрения ДНК состоит из двух последовательностей нуклеотидов (каждый нуклеотид представляет из себя азотистое основание, сахар и фосфатную группу). Нуклеотиды — это своего рода символы, из которых состоит ДНК.

В ДНК встречается четыре типа азотистых оснований — аденин, цитозин, гуанин и тимин, которые обозначаются соответственно *A, C, G* и *T*. Причем азотистые основания одной цепочки ДНК соединены с азотистыми основаниями другой водородными связями согласно принципу комплементарности — аденин соединяется только с тимином, а гуанин — с цитозином. Таким образом, одна из двух цепочек ДНК однозначно получается из другой путем разворачивания и замены каждого нуклеотида на соответствующий ему комплементарный.

Информация, хранящаяся в ДНК организмов, очень важна для их исследования, так как отражает важные свойства живых организмов — наследственность и изменчивость. Так, ДНК особей разных видов различаются значительно сильнее, тем ДНК особей одного вида, а ДНК

потомков одной особи значительно больше схожи, чем похожи в среднем ДНК двух особей одного вида.

Еще одно важное приложение исследования ДНК — генетические заболевания. У особей, зараженных одним генетическим заболеванием, наблюдаются одинаковые изменения в ДНК, что может быть использовано в медицине как для теоретического исследования заболевания, так и для лечения от него.

1.1.2. Направления биоинформатики

Биоинформатика — наука, осуществляющая применение математических и компьютерных технологий к решению биологических задач. Она включает в себя множество различных и совершенно независимых друг от друга областей: молекулярную биоинформатику, структурный анализ, молекурялное моделирование и другие. Список задач молекулярной биоинформатики, решаемых при помощи анализа ДНК, довольно широк и включает в себя следующие проблемы:

1. секвенирование (получение последовательности нуклеотидов по физическому образцу ДНК);
2. сравнение двух нуклеотидных последовательностей и выявление их сходств и различий (например, применяется в судебной медицине).

Методика, разработанная в данной работе, является частью решения первой из приведенных выше задач.

1.2. Секвенирование

1.2.1. Задача секвенирования

Секвенирование ДНК — определение нуклеотидной последовательности по имеющемуся образцу ДНК. В результате этого процесса получается линейная цепочка, отражающая последовательность

нуклеотидов в ДНК. Существует несколько методов секвенирования, различающихся по эффективности и стоимости.

1.2.2. Метод Сэнгера

Метод Сэнгера (метод обрыва цепи) [1] основан на присоединении к секвенируемой молекуле ДНК прямого или обратного секвенирующего праймера и синтезе de novo молекулы нуклеиновой кислоты с применением, например, дидезоксинуклеозидтрифосфатов (ddNTP). При этом синтезируются молекулы разной длины с определённым дидезоксинуклеотидом на конце. После разделения синтезированных молекул ДНК электрофорезом возможно определение первичной последовательности.

Данный метод позволяет за один этап получить последовательность ДНК длиной до 800-1000 нуклеотидов (считанную за раз последовательность нуклеотидов обычно называют *чтением*, а устройства, осуществляющие чтение, - *секвенаторами*).

1.2.3. Метод дробовика

При секвенировании методом дробовика (*shotgun sequencing*) [2][3] ДНК случайным образом дробится на мелкие участки, каждый из которых затем секвенируется каким-нибудь обычным методом, например, методом Сэнгера. Полученные фрагменты ДНК собираются в единую последовательность при помощи специального программного обеспечения. Для того, чтобы это было возможным, обеспечивается многократное *покрытие* генома чтениями.

Пусть имеется набор из N чтений длины l. Пусть также геном имеет длину L. Тогда говорят, что данный набор обеспечивает покрытие генома, равное $N * l / L$.

1.2.4. Методы нового поколения

Методы нового поколения (*next-generation sequencing*) [4] — собирательное название появившихся недавно методов, основанных на получении более коротких чтений (длиной до 500 нуклеотидов), благодаря чему они позволяют получать сотни миллионов чтений дешево и достаточно быстро. Это делает возможным получение большего покрытия генома, нежели при более длинных чтениях, однако приводит к значительному увеличению вычислительной трудоемкости задачи сборки генома из набора чтений.

1.2.5. Секвенаторы

Существует несколько компаний, выпускающих устройства для получения коротких чтений. Самыми распространенными на рынке этих устройств являются продукты компании *Illumina* [5], однако в последнее время секвенаторы компании *Ion Torrent* [6] стремительно набирают популярность благодаря своей дешевизне. Сравнительные характеристики секвенаторов этих двух компаний представлены на Рисунок 1.

Criteria	Ion Torrent Proton	Illumina HiSeq 2500
System Price	$243,000	$740,000
Annual Service cost (yr 2 & 3)	$19,400	$59,200
Per Gb cost	$16.67	$46.00
Time from library to data	8 hours	27 hours

Рисунок 1 – Сравнительные характеристики секвенаторов Illumina и Ion Torrent.

1.2.6. Парные чтения

В последнее время популярным стало использование так называемых *парных чтений* [7]. При прочтении парных чтений секвенатором выделяется

расположенный в случайном месте последовательности ДНК фрагмент, из которого затем считываются префикс и суффикс. Важно отметить, что эти префикс и суффикс считываются с разных нитей ДНК, причем неизвестно, какой был считан с прямой нити, а какой — с обратной. Поэтому удобно рассматривать не исходные геном и набор чтений, а дополненные своими обратно-комплементарными копиями.

Результатом работы секвенатора в случае использования парных чтений являются пары последовательностей, про которые известно, на каком расстоянии они располагались в исходной последовательности ДНК.

1.2.7. Ошибки в чтениях

Данная работа своим существованием обязана одной неприятной особенности процесса секвенирования. Особенность эта заключается в том, что в процессе чтения секвенаторами допускаются ошибки. Ошибки бывают трех типов:

- ошибки вставки — в основном проявляются в прочтении более длинных, чем в исходном геноме, последовательностей одинаковых нуклеотидов (например, вместо «*AA*» было прочитано «*AAA*») ;
- ошибки удаления — в этом случае в прочитанной нуклеотидной последовательности может не хватать одного нуклеотида (например, вместо «*A**C**GT*» было прочитано «*AGT*»);
- ошибки замены — в таких случаях некоторые нуклеотиды были прочитаны неверное (например, вместо нуклеотида *A* был прочитан нуклеотид *G*).

Секвенаторы компании Ion Torrent совершают ошибки вставки и удаления значительно чаще, чем ошибки замены, в связи с чем возникает необходимость разработки методов их исправления.

1.2.8. Выходные данные секвенатора

Данные, получаемые секвенатором, состоят не только из прочитанных фрагментов нуклеотидной последовательности (или из пар фрагментов в случае парных чтений). Кроме этого они содержат информацию о качестве прочтения каждого из нуклеотидов цепочки, представленную в виде последовательности вероятностей ошибочного прочтения нуклеотида для каждой из позиций прочитанного фрагмента.

Существует несколько форматов для выходных данных секвенатора, созданных для удобства обработки полученных данных.

1.2.8.1. Формат FASTA

Файл в формате FASTA [8] представляет собой простой текстовый файл. Первая строка файла должна начинаться с «;» или «>» — в этой строке обычно содержится идентификационный номер библиотеки чтений и самого чтения. Последующие строки, начинающиеся с «;» или «>», воспринимаются как комментарии. Остальная часть файла — последовательность латинских букв, обозначающих нуклеотиды (обычно A, C, G или T).

1.2.8.2. Формат MultiFASTA

Формат MultiFASTA [8] представляет собой модифицированный формат FASTA, допускающий хранение нескольких последовательностей в одном файле. Строки, начинающиеся с «>», в этом формате используются для обозначения начала новой последовательности.

1.2.8.3. Формат FASTQ

Форматы FASTA и MultiFASTA не предусматривают возможности хранения величин качеств для нуклеотидов. Формат FASTQ [9] призван решить эту проблему.

Хранение каждой последовательности в FASTQ-файле занимает 4 строки. Первая строка начинается с «@» и, как и в формате FASTA, служит

для идентификации библиотеки чтений и самого чтения. Вторая строка содержит саму нуклеотидную последовательность в том же формате, что и в FASTA. Третья строка содержит «+» и отделяет строку с последовательностью от следующей за ней строкой с качеством. Для записи величины качества используется следующий алгоритм:

1. Пусть вероятность того, что нуклеотид на некоторой позиции прочитан неверно, равна *p*.
2. Рассчитаем величину качества Phred (Phred quality score) [10] по формуле: $Q = -10 \log_{10} p$.
3. Округлим это число до ближайшего целого и запишем в качестве результата ASCII-символ, соответствующий полученному числу, увеличенному на 33.

Третий шаг немного различается для различных секвенаторов, но общий смысл остается таким.

Хранение нескольких последовательностей в одном файле в формате FASTQ осуществляется так же, как в формате MultiFASTA.

1.3. Сборка генома

1.3.1. Граф де Брёйна

Многие современные сборщики генома используют в процессе сборки подграф так называемого *графа де Брёйна* (рис.Рисунок 2) [11]. Вершинами этого графа являются *k-меры*, то есть строки длины *k* над алфавитом, состоящим из обозначений нуклеотидов. Из вершины *u* ведет ребро в вершину *v* тогда и только тогда, когда суффикс длины *(k-1)* строки, соответствующей вершине *u*, совпадает с префиксом строки, соответствующей вершине *v*. Подграф, используемый сборщиками, состоит из вершин, соответствующих *k*-мерам, встречающимся в чтениях, и ребер,

инцидентных вершинам, соответствующие *k*-меры которых перекрываются на *(k-1)* символ в чтениях, то есть идут друг за другом.

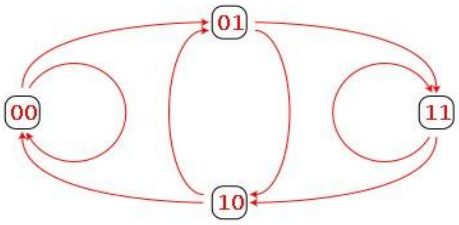

Рисунок 2 – Пример графа де Брёйна (k = 2) [9]

Если считать, что в геноме нет повторов (строк длины хотя бы *k*, встречающихся хотя бы два раза), то длинные пути в подграфе графа де Брёйна соответствуют длинным подстрокам всей последовательности ДНК. Таким образом, для сборки генома можно немного модифицировать и использовать различные алгоритмы на графах, что и используется в той или иной степени в различных сборщиках.

1.3.2. Процесс сборки

Входными данными для сборщика является набор чтений (парных или нет, длинных или нет, с качеством или нет). Задача сборщика — восстановить как можно большую часть последовательности ДНК.

Несмотря на то, что алгоритмы, используемые различными сборщиками, могут быть основаны на совершенно разных идеях, существует несколько этапов сборки, осуществляемых всеми сборщиками.

1.3.2.1. Заполнение промежутков по парным чтениям

Этот этап осуществляется не всеми сборщиками и только в том случае, когда входные данные представлены парными чтениями. В результате работы алгоритма заполнения промежутков получается набор длинных (порядка 500 нуклеотидов) подстрок генома, которые затем можно

использовать как длинные чтения для последующей сборки контигов. Таким образом, нет необходимости разрабатывать специальные сборщики для входных данных, представленных в виде парных чтений.

1.3.2.2. Сборка контигов

После исправления ошибок в чтениях (и, в случае парных чтений, заполнений промежутков) обычно происходит сборка длинных непрерывных подстрок генома — *контигов* (*contigs*). Контиги обычно имеют различную длину - от сотен нуклеотидов до десятков и сотен тысяч.

Для сборки контигов применяются различные алгоритмы, например, основанные на поиске путей в графе де Брёйна.

1.3.2.3. Сборка скэффолдов

Некоторые сборщики прекращают свою работу после сборки контигов. Таким образом, результатом их работы является набор подстрог генома довольно большой длины. Такие данные уже могут быть использованы для решения некоторых биологических задач.

Другие сборщики после сборки контигов собирают *скэффолды* (*scaffolds*) — последовательности контигов, разделенных промежутками известной длины. Также скэффолды называются иногда *суперконтигами* (*supercontigs*).

На этом процесс сборки обычно заканчивается, так как на данный момент неизвестно способов улучшить этот результат. На Рисунок 3 изображена схема процесса сборки скэффолдов из контигов, собранных из парных чтений.

Рисунок 3 – Схематическое изображение процесса сборки.

1.3.3. Анализ результатов сборки

При оценке качества сборки принято пользоваться несколькими метриками. Все они являются численными, то есть сборке в соответствие ставится число, используемое при оценке качества сборки и сравнении результатов работы одного сборщика с результатом работы другого.

1.3.3.1. Суммарная длина контигов

Этот параметр является своего рода индикатором того более-менее успешной сборки. Обычно при сборке генома какого-то организма заранее известна длина его генома. Если суммарная длина контигов значительно превосходит длину генома, это означает, что в чтениях очень много неисправленных ошибок. Если же, наоборот, суммарная длина контигов значительно меньше длины генома, это означает, что часть генома не покрыта чтениями, то есть часть информации о геноме безвозвратно утеряна.

1.3.3.2. Семейство оценок *NXY*

Пусть имеется набор контигов. Отсортируем этот набор по длине. Пусть *f(N)* — суммарная длина всех контигов с длинами хотя бы *N*. Тогда *NXY* — максимальное значение *N*, для которого величина *f(N)* составляет не менее *XY* процентов от длины генома. На практике наиболее часто используется значение величины *N50* [12] – такая длина, что контиги такой или большей длины покрывают хотя бы половину генома. Понятно, что чем больше значение этой метрики, тем лучше. Помимо N50 часто используются такие значения, как *N25*, *N75* и *N90*.

1.3.3.3. Длина самого длинного контига, средняя длина контига

При фиксированной N50 также имеет значение, какой длины самый длинный из собранных контигов, а также средняя длина контигов. Чем больше эти величины, тем лучше.

1.3.3.4. Картирование контигов на известный геном

В случае, когда собираемый геном уже собран, он может быть использован для оценки новых методов сборки. Так, многие алгоритмы сборки генома были тестированы на чтениях бактерии E.coli. При этом методе тестирования для каждого контига производится попытка найти фрагмент в геноме, из которого он получен. В результате некоторые контиги являются подстроками генома, а некоторые – нет. Понятно, что чем меньше контигов содержат ошибки, тем лучше. Так же при помощи такого метода можно определить среднее покрытие генома контигами, а также какие участки генома покрыты плохо или не покрыты вовсе. Эти данные можно использовать для тщательного анализа сборщика.

2. Описание известных подходов к задаче исправления ошибок в чтениях нуклеотидной последовательности

2.1. Программное средство Quake

Подход, применяемый в сборщике Quake, изложен в [13] .

Алгоритм основан на частотном анализе так называемых q-меров. Как и во многих других алгоритмах, рассматриваются не сами чтения, а k-меры. Но вместо того чтобы подсчитывать для каждого k-мера, сколько раз он встречается в чтениях, суммируются вероятности того, что все нуклеотиды в k-мере прочитаны верно (то есть произведения соответствующих величин качеств нуклеотидов на каждой позиции k-мера). Затем считается, что полученное распределение должно быть смесью двух распределений (распределения правильных k-меров и распределения неправильных), и исходя из этого вычисляется порог, по которому все k-меры делятся на надежные и ненадежные.

Затем для каждого чтения, содержащего ненадежные k-меры, производится попытка исправления некоторых нуклеотидов. Исправление считается успешным, если после него все k-меры в чтении становятся надежными. Поскольку для каждого чтения существует несколько успешных исправлений, для каждого исправления при помощи формулы Байеса вычисляется его вероятность, после чего исправления перебираются в порядке уменьшения вероятности. Как только находится успешное исправление, оно применяется.

Поскольку в данном алгоритме используется достаточно небольшое значение k (15 для небольших геномов и 19 – для сравнимых по размеру с человеческим), для определения, является ли какой-то k-мер надежным, хранится битовый массив, требующий всего 128 МБ памяти для k, равного 15, и 32 Гб – для 19, что весьма неплохо.

2.2. Программное средство ECHO

Алгоритм, используемый сборщиком *ECHO*, изложен в [14]. Алгоритм состоит из двух шагов:

а) Поиск перекрывающихся чтений.

б) Исправление ошибок.

На первом шаге для каждого *k*-мера создается список чтений, в которых этот *k*-мер присутствует. Затем внутри каждого списка определяется, какие чтения накладываются друг на друга. На работу этого этапа существенно влияют величины трех параметров – величина *k*-мера, минимальная длина наложения между двумя чтениями и максимальная доля несовпадений нуклеотидов, стоящих на соответствующих местах в чтениях. Выбор значения *k* осуществляется перед началом работы алгоритма на основе длин чтений и их среднего качества. Выбор же последних двух параметров происходит значительно сложнее. Для начала они полагаются равными неким начальным значениям. После чего запускается алгоритм нахождения перекрытий в чтениях. Полученные чтения имеют некое распределение. Далее выбирается некое Пуассоновское распределение, которое больше всего похоже на полученное. Затем новые значения для минимальной длины наложения и максимальной доли ошибок выбираются таким образом, чтобы полученное распределение чтений было наиболее близко к Пуассоновскому.

В начале второго этапа для каждого из четырех символов алфавита вычисляется вероятность того, что он находится в заданной позиции, при условии того, что какой-то другой символ был прочитан на этой позиции. Для вычисления этих вероятностей используется *EM-алгоритм*. После подсчета этих величин выбор нуклеотида на каждой позиции происходит согласно принципу наибольшего правдоподобия.

2.3. Программное средство Hammer

Алгоритм, используемый в программном средстве *Hammer,* описан в [15]. На первом шаге работы *Hammer* происходит сортировка всех присутствующих в чтениях *k*-меров. Затем по *k*-мерам строится *граф Хемминга* (неориентированный граф, вершинами которого являются *k*-меры, а ребро между вершинами *u* и *v* существует в том случае, когда расстояние Хемминга [16] между соответствующими *k*-мерами не превосходит некоторого заранее выбранного числа *t*). После построения графа в нем ищутся компоненты связности. Если число *t* было выбрано небольшим, а *k*-меры достаточно длинные, то каждая компонента связности с большой вероятностью была порождена одним *k*-мером, прочитанным несколько раз неверно. В таком случае решение о том, каким *k*-мером была порождена эта компонента связности, принимается по правилу консенсуса, то есть на каждой позиции выбирается тот нуклеотид, который встречается на ней чаще остальных.

Важной особенностью этого метода является отсутствие требования равномерного покрытия генома чтениями.

2.4. Программное средство HiTEC

Алгоритм, используемый в программном средстве *HiTEC,* описан в [17]. В этом алгоритме сначала для каждого *k*-мера подсчитывается, какие символы следуют за ним в чтениях и сколько раз. Затем для каждого *k*-мера из возможных символов, идущих в чтениях за ним, выбираются те, которые встречаются достаточно большое число раз (пороговое значение рассчитывается теоретически). Если такой нуклеотид один, то нуклеотиды, стоящие на позициях, следующих в чтениях за рассматриваемым *k*-мером, заменяются им.

2.5. Протокол SHRAP

SHRAP (Short Reads Assembly Protocol) — протокол сборки длинных геномов, основанный на иерархическом секвенировании (Рисунок 4) [18]. Иерархическое секвенирование (*hierarchical sequencing*) — исторически один из первых способов секвенирования. Первый шаг протокола состоит в том, чтобы выделить много фрагментов генома длиной около 150 *k*b (*kilobase*, т. е. тысяч нуклеотидов). Эти фрагменты называются *клонами* (*clones*) и обеспечивают достаточно большое покрытие генома (порядка десятикратного). Затем из каждого клона получаются чтения длиной около 200 нуклеотидов. Чтения обеспечивают небольшое покрытие клона — порядка двукратного. Таким образом, в итоге чтения обеспечивают двадцатикратное покрытие генома.

В традиционном иерархическом секвенировании примерное расположение клонов в геноме известно, поэтому сначала сами клоны собираются по чтениям, а затем информация о расположении клонов используется для сборки их в более длинные куски. В предлагаемом методе информация о расположении клонов отсутствует, поэтому метод сборки совершенно другой.

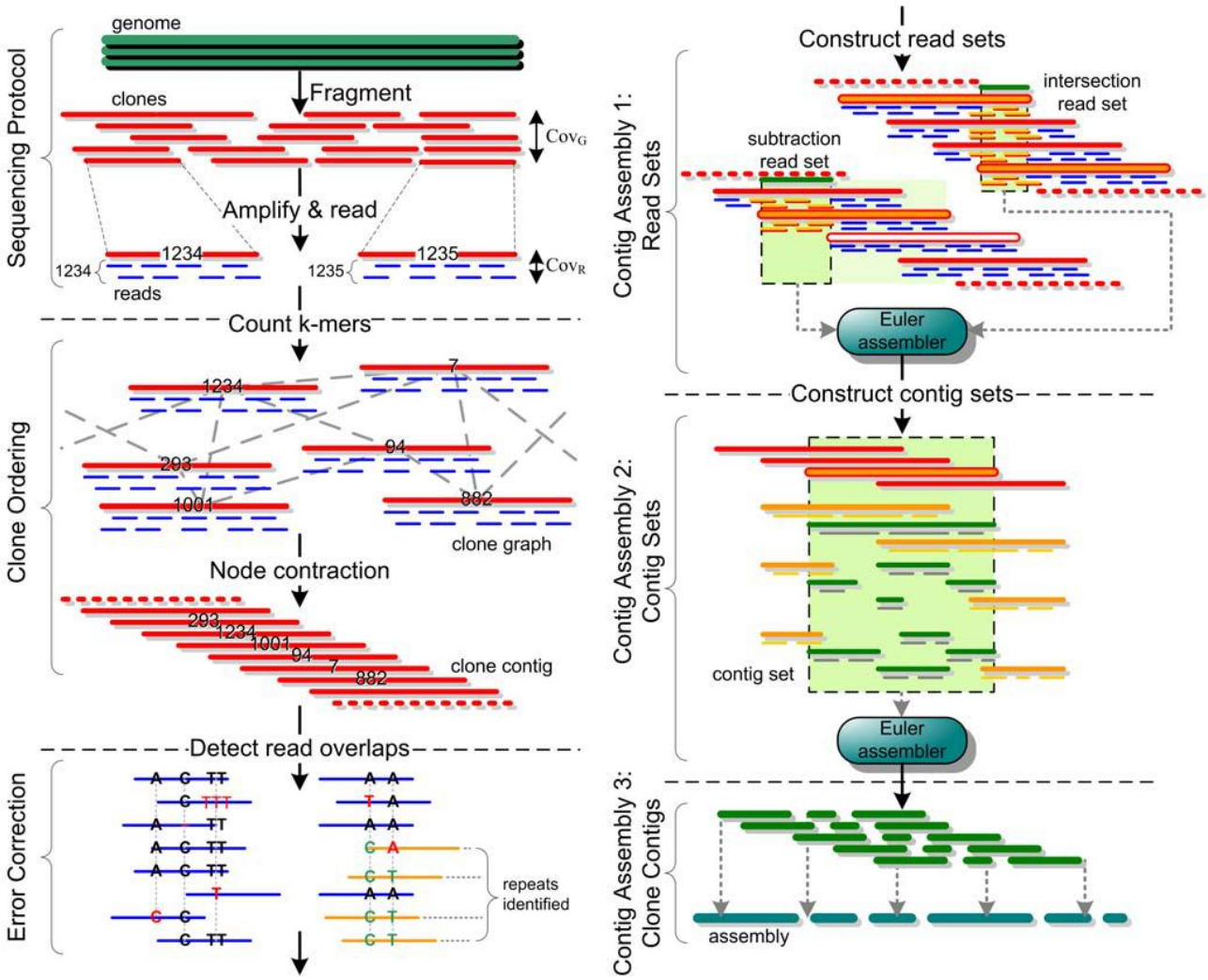

Рисунок 4 – Схема работы сборщика, основанного на иерархическом секвенировании.

Сначала при помощи чтений определяются пересекающиеся клоны, из которых в итоге строятся *клоны-контиги* (*clone contigs*), т. е. цепочки клонов, сильно пересекающихся друг с другом, расположенные в том порядке, в котором они расположены в геноме. Информация о расположении клонов-контигов используется в дальнейшем для сборки чтений и исправления ошибок в них.

На стадии исправления ошибок информация о расположении клонов-контигов используется для ускорения вычисления наложений чтений друг на друга. Для каждого чтения рассматриваются только те чтения, расположенные в том же клоне-контиге, причем в клоне, пересекающимся с

клоном обрабатываемого чтения. Для нахождения пересечения между чтениями используются 16-меры.

На первой стадии для каждого чтения строится множество других чтений, пересекающихся с ним. Затем для каждого чтения проводится два теста — тест на ошибочность (*error-rate test*) и тест на коррелированность (*correlation test*). Тест на ошибочность отфильтровывает чтения, содержащие слишком много отличий от остальных чтений (больше утроенного ожидаемого числа ошибочно прочитанных нуклеотидов). В тесте на коррелированность выделяются чтения, повторяемость которых вызвана повторами в геноме. Чтения, не прошедшие хотя бы один тест, удаляются из множества.

После удаления не прошедших тесты чтений множество *транзитивно* пополняется. Назовем множество чтений, перекрывающихся с p, R_p. Для транзитивного пополнения рассматривается каждая пара чтений p и q из R_r, и если их расположение относительно чтения r влечет их перекрытие, то p добавляется в R_q, а q — в R_p.

На второй стадии чтения одного множества вновь выстраиваются в цепочку, после чего из множества снова удаляются не проходящие тесты чтения. На третьей стадии при помощи применения простого правила большинства к каждой позиции из чтений составляется цепочка нуклеотидов.

2.6. Сборщик AllPaths

Алгоритм исправления ошибок здесь [19] имеет дело с набором чтений и является первой частью алгоритма сборки. В процессе работы алгоритма чтения делятся на три группы: чтения, которые оставляются без изменений, чтения, в которых осуществляются исправления, и чтения, которые удаляются и больше не используются.

Для начала подсчитывается частотная статистика k-меров. Для каждого m вычисляется количество k-меров, встречающихся в чтениях ровно

m раз. Полученная функция имеет два пика. Первый, резкий, находится в точке m=1 и связан с ошибками в чтениях. Второй же, гораздо более гладкий, обусловлен статистическим распределением *k*-меров при большом покрытии и с ошибками не связан. Между этими двумя пиками есть минимум функции, располагающийся в точке m_1. Большинство *k*-меров, отвечающих точкам левее m_1, содержат ошибки, тогда как большинство располагающихся справа от нее ошибок не содержат. Правые *k*-меры называются *надежными* (*strong*).

Подсчет статистики производится для нескольких значений *k* — 16, 20, 24. Для каждого из значений определяется m_1. Если для некоторого чтения все *k*-меры для всех значений *k* являются надежными, это чтение оставляется без изменений и считается правльным. В противном случае производится попытка исправить один или два нуклеотида (большее количество исправлений возможно, но также влечет за собой дополнительное увеличение времени работы алгоритма). Каждому изменению ставится в соответствие вероятность, пропорциональная качеству нуклеотида, на который производится замена. Если наиболее вероятная замена оказывается хотя бы в 10 раз более вероятной, чем вторая по величине вероятности, то эта замена осуществляется и чтение считается исправленным. В противном случае чтение удаляется.

2.7. Сборщик Velvet

В этом подходе [20] применяется разновидность *графа де Брёйна* (*de Bruijn graph*). В этом графе каждая вершина хранит упорядоченный список *k*-меров, причем соседние *k*-меры в списке пересекаются по (*k*-1) символам. Вся информация, хранящаяся в вершине, может быть представлена первым *k*-мером в списке и списком последних нуклеотидов каждого *k*-мера.

Каждая вершина имеет «прикрепленную» к ней вершину-двойника, в которой хранится список тех же *k*-меров, но развернутых и

комплементарных. Пара из вершины и ее двойника называется *блоком*. Вершину, являющуюся двойником вершины *A*, будем обозначать *A*.

Вершины в графе соединяются ориентированными ребрами по тому же принципу, по которому *k*-меры соседствуют в списках, хранящихся в вершинах, — суффикс длины *k*-1 последнего *k*-мера вершины, из которой выходит ребро, совпадает с префиксом первого *k*-мера вершины, в которую входит это ребро. Из-за симметрии блоков наличие ребра из вершины *A* в вершину *B* всегда влечет наличие обратного ребра из *B* в *A* (Рисунок 5).

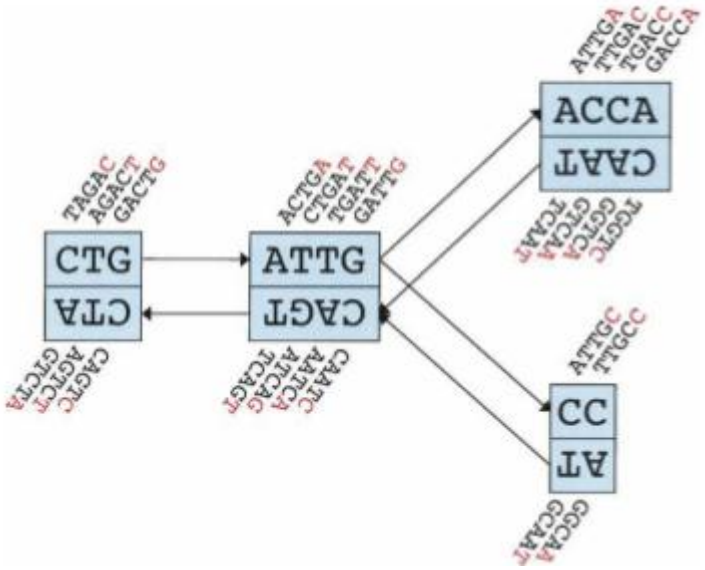

Рисунок 5 – Пример графа де Брёйна для k = 5.

Построение графа де Брёйна — первая стадия работы сборщика. После построения граф упрощается, после чего на нем запускается алгоритм исправления ошибок.

Ошибки чтения порождают в графе структуры трех типов: *отростки* (*tips*), *пузыри* (*bulges*) и ошибочные ребра. Первые возникают из-за ошибок на концах чтений, вторые — из-за ошибок внутри чтений, а третьи — в результате повторных ошибок. Эти три типа ошибок удаляются одна за другой.

2.7.1. Удаление отростков

Ошибки данного типа порождают в графе обрывающиеся цепочки вершин, поэтому их удаление на первый взгляд не представляет особых трудностей. Однако не все обрывающиеся цепочки вершин небольшой длины вызваны ошибками чтения — некоторые из них могут быть вызваны небольшим покрытием чтениями небольшого куска генома. Для того чтобы различать эти два случая, используются два критерия: длина цепочки и *критерий альтернативности* (*minority count*).

Критерий длины цепочки заключается в том, что удаляются только цепочки длины меньше *2k*. Такой выбор обусловлен максимальной длиной ошибочного куска, вызванного наличием двух идущих подряд ошибочно прочитанных нуклеотидов.

Критерий альтернативности заключается в том, что ребро, ведущее в первую вершину цепочки, должно иметь меньший вес, чем любое другое ребро, ведущее из той же вершины. Иными словами, путь в отросток должен быть альтернативой более общему пути в графе.

Таким образом, отросток удаляется только при выполнении двух приведенных выше критериев.

2.7.2. Удаление пузырей

Пузырем называется несколько путей между одной парой вершин, содержащих схожие последовательности символов. Нахождение таких путей осуществляется при помощи модифицированного поиска в ширину. Запущенный из произвольной вершины, алгоритм посещает вершины в порядке увеличения расстояния от этой вершины, причем под расстоянием между парой вершин, соединенных ребром, понимается длина последовательности, хранящейся в вершине назначения, деленная на вес ребра, соединяющего эти вершины в этом направлении. Как только поиском обнаруживается ребро, ведущее в уже посещенную вершину, из нее и

текущей вершины запускается поиск ближайшего общего предка. После нахождения общего предка строки, соответствующие путям из этого предка в рассматриваемую вершину, сравниваются. Если они достаточно похожи, то есть различия в них могут быть списаны на ошибки в чтениях, то пути объединяются, причем из двух путей выбирается кратчайший согласно выбранной метрике (Рисунок 6). Заметим, что модифицирование путей — операция довольно дорогостоящая из-за дополнительной информации, хранящейся в графе.

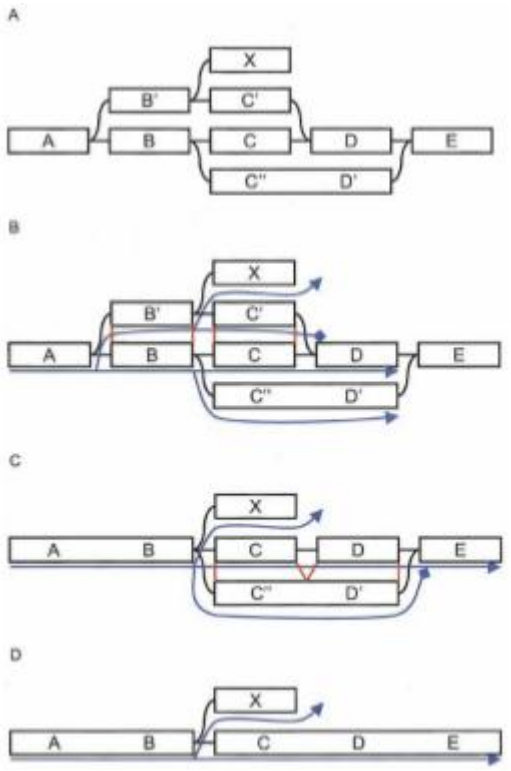

Рисунок 6 – Пример работы алгоритма удаления пузырей.

2.7.3. Удаление ошибочных ребер

Удаления ошибочных ребер производится на основе их веса. Считается, что после работы предыдущих стадий ребра, имеющие слишком маленький вес, то есть слишком мало покрытые, вызваны ошибочными чтениями.

2.8. Алгоритм EULER-USR

Алгоритм [21] состоит из трех шагов:

1. Определение длины префиксов с неплохим качеством и исправление ошибок в них на основе частотного анализа *k*-меров. После этой операции остается множество префиксов чтений, практически не содержащих ошибок.
2. Построение на основе *k*-меров полученных префиксов *графа повторов* (*repeat-graph*).
3. Упрощение построенного графа.

Для исправления ошибок используются простые соображения, по которым *k*-меры делятся на две группы — *надежные* (*solid*) и все остальные. Надежными называются *k*-меры, которые встречаются в чтениях не меньше некоторого заранее выбранного числа *m*. Если все *k*-меры одного чтения являются надежными, то считается, что чтение не содержит ошибок. В противном случае производится попытка исправить ошибки в нем. Рассматриваются только ошибки замены, так как ошибки вставки и удаления редки в чтениях Illumina.

Для исправления ошибок жадным образом ищется минимальное число исправлений, которое необходимо осуществить, чтобы сделать каждый *k*-мер в чтении надежным. Для каждого исправления на каждой позиции записывается, сколько *k*-меров становится надежными в результате этого исправления. Исправление, которое делает максимальное число *k*-меров надежными, применяется, если это число не меньше некоторого заранее выбранного порога *t*. Когда находится подходящее исправление, выводится максимальный по длине префикс чтения, содержащий только надежные *k*-меры. Суффикс *k*-мера, считающийся менее надежным, будет исправлен на более поздней стадии работы алгоритма.

Для выбора параметра *k* полагается, что распределение *k*-меров является смесью двух распределений — распределения ошибочных *k*-меров и

распределения правильных (рис. 6). И те, и другие *k*-меры распределены по закону Пуассона, но распределение правильных *k*-меров имеет большое среднее, поэтому аппроксимируется распределением Гаусса. Параметр *m* выбирается как точка минимума распределения *k*-меров, параметры которых определяются исходя из частотного анализа *k*-меров.

2.9. Сборщик SOAPdenovo

Подход, используемый в сборщике *SOAPdenovo,* изложен в [22]. Исправление ошибок в этом сборщике осуществляется в два этапа. На первом этапе для каждого *k*-мера определяется, сколько раз он встречается в наборе чтений, после чего *k*-меры, встречающиеся меньше определенного числа раз, помечаются ошибочными. Затем в чтениях, содержащих ошибочные *k*-меры, выделяется область, не содержащая ошибок. Эта область расширяется в обе стороны до тех пор, пока не будет встречен ошибочный нуклеотид. Затем производится попытка исправить нуклеотид так, чтобы рассматриваемая область стала безошибочной. Если есть хотя бы один из трех возможных вариантов нуклеотидов на рассматриваемой позиции приводит к этому, выбирается тот, который приводит к наиболее часто встречаемому *k*-меру. В противном случае нуклеотид на позиции не изменяется.

На втором шаге строится граф де Брёйна, с которым производится процедура удаления «пузырей», аналогичная процедуре их удаления в сборщике *Velvet*.

2.10. Метод цифровой нормализации

Метод цифровой нормализации подробно описан в [23]. Целью данного метода является не исправление ошибок в чтениях, а уменьшение количества данных за счет удаления избыточных чтений. При этом происходит также значительное сокращение ошибок в чтениях.

Поскольку чтения равномерно распределены по геному, для полного покрытия генома необходимо число чтений, значительно превышающее размер генома. Это приводит к тому, что большая часть генома покрыта чтениями десятки или даже сотни раз, из-за чего данные секвенирования имеют очень большой объем. Большая часть этих данных избыточна и может быть без потерь для эффективности сборки удалена.

Для определения, какие чтения удалять, производится следующая оценка. Если в чтении нет ошибки, все *k*-меры в нем имеют примерно одинаковую частотную характеристику (Рисунок 7). В противном случае частота ошибочных *k*-меров значительным образом отличается от частоты безошибочных. Каждая ошибка порождает не более k ошибочных *k*-меров. Таким образом, если чтение имеет длину хотя бы 3k и содержит не более одной ошибки, медиана частотных характеристик содержащихся в нем *k*-меров примерно отражает реальное покрытие того участка генома, из которого получено это чтение.

Разработчиками этого метода было проведено экспериментальное исследование, показавшее строгую корреляцию медианной частотной характеристики чтения и реального покрытия для известных геномов и библиотек чтений из них.

Таким образом, алгоритм дискретной нормализации работает следующим образом. Чтения обрабатываются по одному. Для рассматриваемого чтения вычисляется медианная частотная характеристика (на основе уже рассмотренных чтений). Если эта характеристика меньше некоторого заранее выбранного значения, чтения допускается, в противном случае – удаляется из набора. Допуск чтения означает, что содержащиеся в нем *k*-меры добавляются в структуру данных, используемую для вычисления частотных характеристик.

После сокращения покрытия происходит удаление *k*-меров со слишком маленькой частотной характеристикой, за счет чего происходит сокращение доли ошибочных *k*-меров в чтениях.

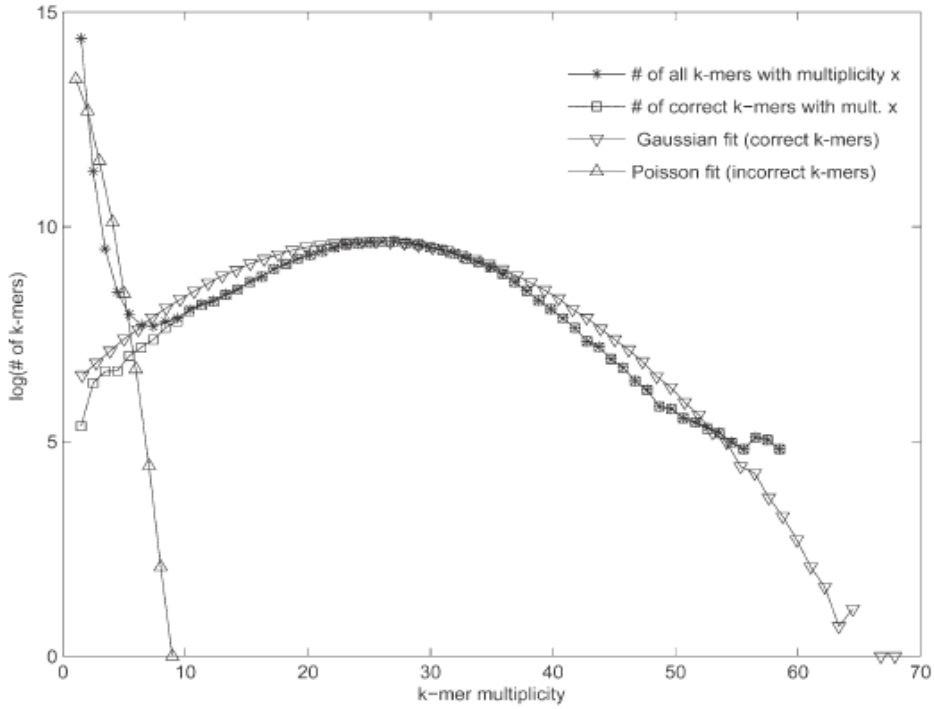

Рисунок 7 – Экспериментально полученное распределение k-меров.

2.11. Достоинства и недостатки описанных подходов

Все описанные выше и другие известные подходы к решению задачи исправления ошибок можно разделить на два типа: одни используют граф де Брёйна и работают с графовыми структурами, другие же основаны на частотном анализе *k*-меров. Все они довольно требовательны к вычислительным ресурсам, недостачно эффективно работают с ошибками вставки и удаления, а также не очень хорошо масштабируются.

3. Описание предлагаемого метода

3.1. Введение

Предлагаемый в данной работе метод основан на определении перекрывающихся чтений и не использует графа де Брёйна. Для эффективного исправления ошибок необходимо, чтобы каждая позиция генома была прочитана несколько раз, так как это единственный способ отличить правильно прочитанный нуклеотид от прочитанного неверно. Это, ввиду небольшой вероятности ошибки, дает право считать, что наибольшее число раз нуклеотид на каждой позиции был прочитан верно. На практике используются наборы чтений, покрывающие геном несколько десятков раз. Важно отметить, что не только отдельные позиции всего генома были прочитаны несколько десятков раз, но и небольшие его подстроки (не длиннее самих чтений) встречаются в чтениях несколько раз, причем чем длиннее подстрока, тем меньше шансов, что несколько различных чтений ее содержат.

3.2. Идея метода

Рассмотрим чтения, полученные из одного фрагмента генома (Рисунок 8). Поскольку доля ошибок невелика (значительно меньше половины), они должны быть похожи. По этим чтениям по принципу консенсуса можно восстановить фрагмент генома, из которого они были прочитаны. После этого по строке можно восстановить сами чтения.

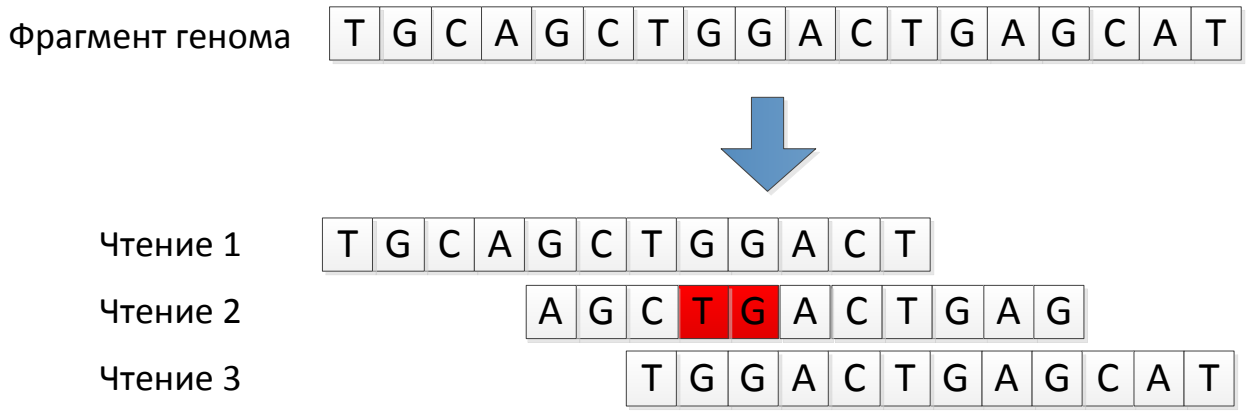

Рисунок 8 – Фрагмент генома и чтения, произведенные из этого фрагмента. Красным выделено место, содержащее ошибки.

3.3. Предлагаемый алгоритм

Поскольку сама геномная последовательность в процессе сборки недоступна, для выделения группы чтений, полученных из одного фрагмента генома, нельзя просто найти подстроку генома, больше всего похожую на набор чтений. Вместо этого рассмотрим *k*-меры – подстроки чтений длины *k*.

3.4. «Хорошие» и «плохие» k-меры

Если бы ошибок не было, каждый *k*-мер задавал бы фрагмент генома (или несколько, если в геноме есть повторы длины хотя бы *k*). В случае наличия в чтениях ошибок некоторые *k*-меры не являются подстроками чтений.

Поскольку ошибки происходят с небольшой вероятностью, вероятность того, что один и тот же *k*-мер будет прочитан несколько раз с одинаковым набором ошибок, очень мала. Из этого вытекает, что те *k*-меры, которые встречаются в наборе чтений мало раз, являются ошибочными, остальные же являются реальными подстроками генома (Рисунок 9). Будем называть редко встречающиеся *k*-меры «плохими», а часто встречающиеся – «хорошими».

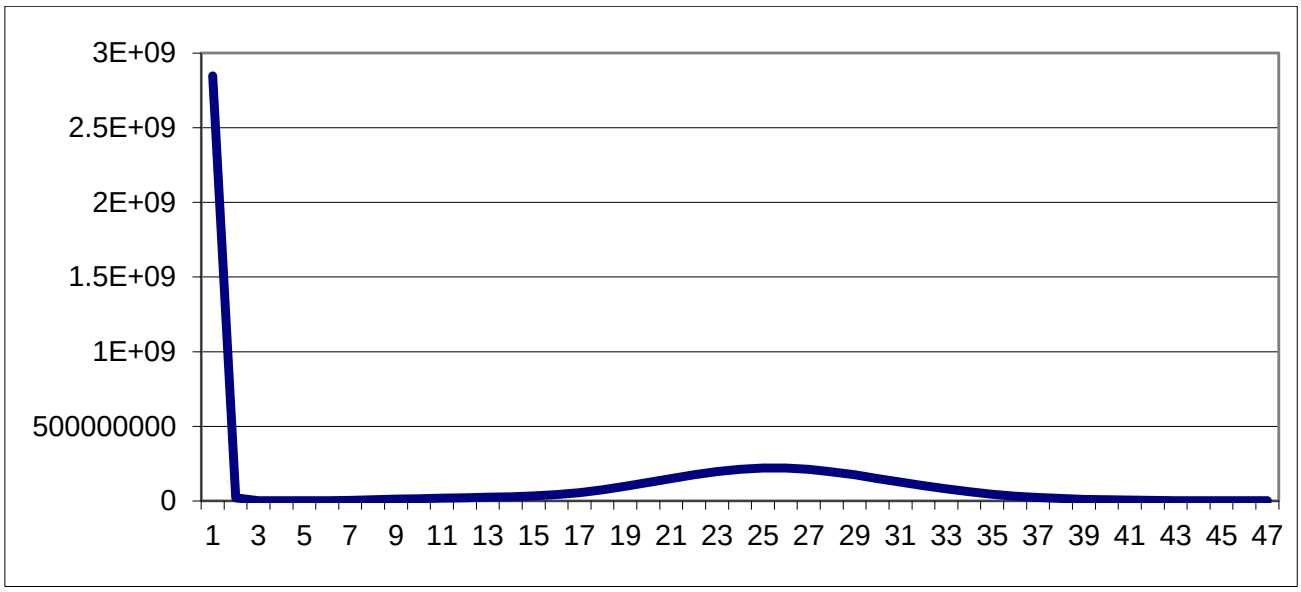

Рисунок 9 – Распределение частот k-меров в чтениях. По оси X отложена частота k-мера, по оси Y – число k-меров с такой частотной характеристикой. Первый пик находится в точке 1 и соответствует ошибочным k-мерам.

В качестве порогового значение выбирается значение, соответствующее первому минимуму числа *k*-меров в чтениях.

После нахождения всех «хороших» *k*-меров для каждого из них вычисляется набор чтений, которые содержат рассматриваемый *k*-мер. Эти чтения составляют одну группу. Группы обрабатываются независимо друг от друга.

3.5. Консенсус

3.5.1. Кластеризация

Поскольку в геноме обычно есть повторы небольшой длины, некоторые *k*-меры встречаются в нем в нескольких местах. Это означает, что чтения, загруженные по какому-то *k*-меру, могут сильно отличаться друг от друга. В этом случае их можно разделить на группы, внутри которых они

будут различаться гораздо меньше, после чего группы можно будет обрабатывать отдельно друг от друга (Рисунок 10).

Понятно, что повторы, длина которых приближается к длине чтений, так разрешить невозможно. Однако повторы, длина которых не сильно превышает длину *k*-мера, таким образом учесть можно.

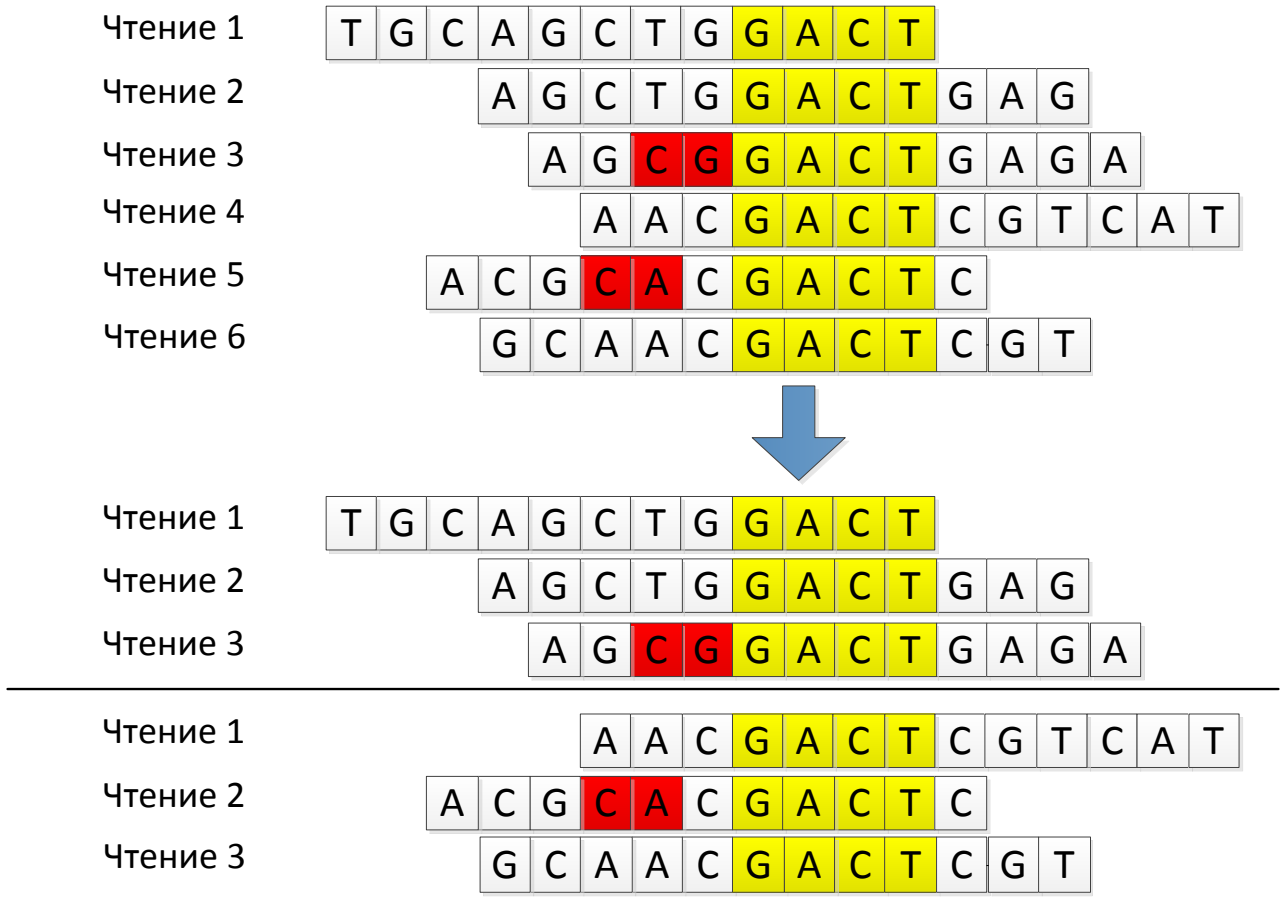

Рисунок 10 – Кластеризация чтений, полученных из разных мест генома.

3.5.2. Разрезание чтений

Поскольку все чтения имеют общий *k*-мер, удобно каждое чтений разделить на три части – левую, среднюю и правую (Рисунок 11). Затем левые части разворачиваются, после чего группы левых и правых частей обрабатываются отдельно друг от друга. Это делается для того, чтобы все обрабатываемые строки в группе начинались в одной позиции.

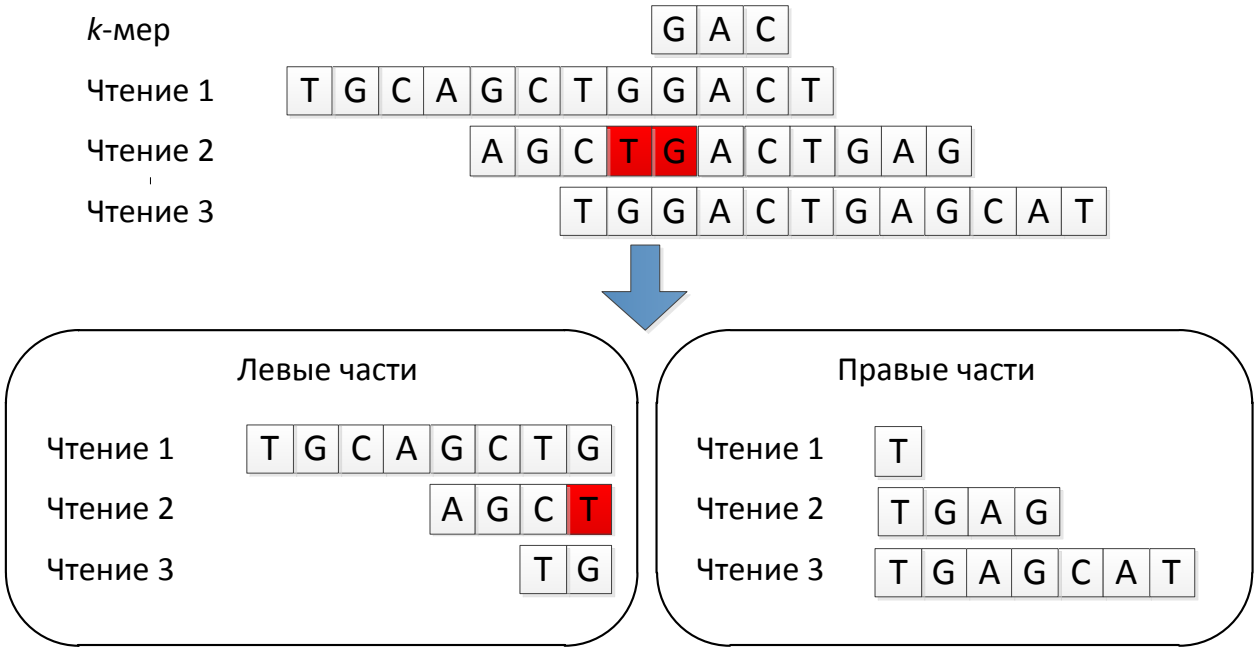

Рисунок 11 – Выделение из чтений левой и правой частей.

3.5.3. Алгоритм получения строки-консенсуса

Строка-консенсус восстанавливается инкрементально. При этом на шаге с номером i поддерживаются текущий префикс искомой строки, имеющий длину i, и множество индексов для каждой из данных строк. Индекс i для строки x означает, что подозревается, что префикс строки x длины i соответствует текущему префиксу строки-консенсуса. В начале работы алгоритма искомая строка пуста, а индексы каждой из строк содержат лишь ноль. Для увеличения длины текущего префикса на единицу перебирается все 4 варианта продолжения, для каждого из них для каждого индекса вычисляется, как изменяется расстояние Левенштейна между префиксом строки-консенсуса и соответствующей индексу подстрокой. Если увеличение длины префиксов строки-консенсуса и данной строки происходит без увеличения расстояние, то данная строка «поддерживает» добавляемый к строке-консенсусу символ. В итоге к строке-консенсусу добавляется тот символ, который поддерживает больше всего чтений.

Таким образом, алгоритм состоит из следующих шагов:

1. Сбор статистики по *k*-мерам;
2. Определение порога и построение множества «хороших» *k*-меров;
3. Построение индекса чтений для всех «хороших» *k*-меров;
4. Для каждого *k*-мера:
    1. Разделение каждого чтения на 3 части
    2. Нахождение строки-консенсуса для каждой части чтений
    3. Исправление ошибок в чтениях

Важно отметить, что алгоритм поиска ошибок в *k*-мерах легко распараллеливается, так как соответствующие различным *k*-мерам чтения могут обрабатываться независимо друг от друга.

3.6. Оценка времени работы предложенного метода

Основная работа алгоритма происходит при построении строки-консенсуса для группы строк. Пусть имеется N строк, для которых необходимо найти консенсус. Пусть самая длинная из них имеет длину L. Тогда всего алгоритм сделает не более L шагов. На каждом шаге необходимо пройтись по всем N строкам и для каждой из них обновить расстояние Левенштейна до текущего префикса. Это делается за $O(K)$, где K – длина текущего префикса. Таким образом, суммарное время обработки группы чтений не превосходит $N * (O(1) + O(2) + \ldots + O(L)) = O(N * L^2)$. L в данной оценке разная для каждой группы, однако не превосходит максимальной длины чтений R.

Если средняя частота *k*-мера в чтениях равна C, то время работы алгоритма не превосходит $O(L * C * R^2)$, где L – число «хороших» *k*-меров (длина генома). Таким образом, число элементарных операций, необходимых для сборки бактериального генома (длина – несколько миллионов нуклеотидов) со средней частотой *k*-мера, равной 30, и средней длине чтения, равной 100, примерно равно $1 * 10^{13}$. Вычислительному узлу с 24

процессорами, совершающими по 10^8 элементарных операций в секунду, на это требуется около часа, что, принимая во внимание, что остальные стадии сборки аналогичного по размеру генома могут требовать несколько часов работы, является неплохим результатом.

4. Экспериментальные исследования

Для оценки эффективности предложенного метода было проведено несколько экспериментов. Тестирование проводилось на двух библиотеках – искусственно сгенерированном наборе чтений из генома бактерии *Escherichia coli* и реальной библиотеке чтений бактерии *Pseudomonas stutzeri*.

4.1. Средства тестирования

Для оценки качества тестирования одни и те же библиотеки собирались разными сборщиками – сборщиком *ABySS* [24] и *ITMO-denovo-assembler* [25]. Причем для каждого сборщика проводилось два эксперимента – сборка осуществлялась как из исходных чтений, так и из исправленных. *ABySS* не предназначен для работы с чтениями, содержащими ошибками вставки и удаления, поэтому результаты работы *ABySS* на исправленных чтениях отражают качество работы этапа исправления ошибок. Также использовался сборщик *Mira-assembler* [26] поддерживающий чтения секвенаторов *Ion Torrent*, то есть умеющий работать с ошибками вставки и удаления.

4.2. Библиотека E.coli

Бактерия *Escherichia coli* имеет геном длиной около 4,5 миллионов нуклеотидов. Сгенерированная библиотека обеспечивала 20-кратное покрытие генома чтениями длиной около 100 нуклеотидов.

Библиотека была сгенерирована в два этапа. На первом этапе были сгенерированы безошибочные подстроки генома. На втором этапе в эти подстроки были внесены ошибки. Такой способ генерации библиотеки позволил сравнить результаты сборки безошибочных чтений и сборки чтений с ошибками с применением исправления ошибок.

Результаты сборки представлены в Таблица 1.

	Суммарная длина контигов	*N50*	Максимальная длина контига
Безошибочные чтения, *ABySS*	4582767	43154	162361
Неисправленные чтения с ошибками, *ABySS*	3003406	163	981
Исправленные чтения с ошибками, *ABySS*	4593871	28492	92287
Неисправленные чтения с ошибками, *Mira-assembler*	4593118	21162	83750

Таблица 1 Результаты сборки искусственных чтений E.coli.

Первая строка показывает, насколько хорошо вообще *ABySS* может обработать данные чтения.

Вторая строка таблицы доказывает, что *ABySS* не приспособлен для обработки чтений с ошибками вставки и удаления. Это выражается в маленькой суммарной длине контигов (примерно 2/3 длины всего генома), а также в маленьком значении *N50* (примерно вдвое больше длины чтения).

Третья строка таблицы показывает, что исправление ошибок значительно улучшает ситуацию, потому что результаты сборки *ABySS* на исправленных всего в полтора раза хуже сборки из безошибочных чтений.

Четвертая строка таблицы показывает, что приспособленный для чтений *Ion Torrent* сборщик *Mira-assembler* справляется с задачей хуже, чем разработанный метод.

4.3. Библиотека P.stutzeri

Бактерия *Pseudomonas stutzeri* имеет геном длиной около 4,5 миллионов нуклеотидов. Библиотека состояла из 4 файлов: l3_1_in.iontor.fastq, l3_2_in.iontor.fastq, 100_in.iontor.fastq и 200_in.iontor.fastq. Файлы l3_1 и l3_2 содержали парные чтения и обеспечивали суммарное 35-кратное покрытие. Средняя длина чтений в файлах l3_1 и l3_2 – 83 нуклеотида. Библиотека 100 содержала 4,6 миллионов чтений со средней длиной 117 нуклеотидов, обеспечивая 26-кратное покрытие генома. Наконец, библиотека 200 содержала 5,5 миллионов чтений со средней длиной 231 нуклеотид, обеспечивая 62-кратное покрытие. Таким образом, суммарное покрытие генома чтениями составило 123.

Результаты сборки представлены в Таблица 2.

	Суммарная длина контигов	*N50*	Максимальная длина контига
Неисправленные чтения *l3_1* и *l3_2, ABySS*	5134815	3227	15646
Исправленные чтения *l3_1* и *l3_2, ABySS*	4863666	9926	68660
Неисправленные чтения *l3_1* и *l3_2, ITMO-denovo-assembler*	5321832	7118	49527
Исправленные чтения *l3_1* и *l3_2, ITMO-denovo-assembler*	4672051	11619	66366
Неисправленные чтения *l3_1* и *l3_2, Mira-assembler*	5205159	5285	47004
Неисправленные чтения *l3_1*, *l3_2* и *100, ABySS*	4495333	17881	69269
Исправленные чтения *l3_1*,	4558926	18878	76335

l3_2 и *100*, *ABySS*			
Неисправленные чтения *l3_1*, *l3_2* и *100*, *ITMO-denovo-assembler*	5076230	13267	67933
Исправленные чтения *l3_1*, *l3_2* и *100*, *ITMO-denovo-assembler*	4878434	20290	98357
Неисправленные чтения *l3_1*, *l3_2* и *100*, *Mira-assembler*	5250041	15252	95547
Неисправленные чтения *l3_1*, *l3_2*, *100* и *200*, *ABySS*	4500845	18034	61569
Исправленные чтения *l3_1*, *l3_2*, *100* и *200*, *ABySS*	4501487	22017	63725
Неисправленные чтения *l3_1*, *l3_2*, *100* и *200*, *ITMO-denovo-assembler*	5126675	14755	51078
Исправленные чтения *l3_1*, *l3_2*, *100* и *200*, *ITMO-denovo-assembler*	4799664	22265	74866

Таблица 2 Результаты сборки реальных чтений P.stutzeri.

Как и на искусственных данных, на настоящих чтениях разработанный алгоритм значительно улучшает качество сборки, причем как сборщиком *ABySS*, так и *ITMO-denovo-assembler*. Также заметно, что качество сборки исправленных чтений любым сборщиком выше, чем качество сборки при помощи *Mira-assembler*.

4.4. Итоги и дальнейшие направления работы

Был разработан метод исправления ошибок, основанный на поиске перекрытий между чтениями. Также было проведено экспериментальное исследование разработанного метода, показавшее работоспособность метода как на искусственных, так и на реальных данных. Результаты, полученные с использованием разработанного метода, превосходят по основным характеристикам результаты, достижимые при помощи распространенных аналогов предложенного метода. Разработанный метод с легкостью распараллеливается на большое число процессоров, что делает его использование возможным для больших объемов данных.

В настоящий момент исследуются пути увеличения эффективности работы предложенного метода. Так, планируется добавить возможность загрузки чтений не по одному *k*-меру, а по нескольким соседним. Это позволит загружать за раз больше чтений и значительно уменьшит число итераций алгоритма. Также планируется в большей мере использовать данные о качестве нуклеотидов, предоставляемые секвенатором.

Заключение

Проведенная работа имеет следующие результаты:

- разработан алгоритм исправления ошибок вставки и удаления в чтениях геномной последовательности, превосходящий существующие по эффективности;
- разработанный алгоритм реализован в виде программы для ЭВМ;
- разработанная программа успешно протестирована на синтетических и реальных данных.

Таким образом, поставленные задачи выполнены, цель работы достигнута. Также поставлены цели по улучшению разработанного метода.

Источники

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

2 Staden R. A. strategy of dna sequencing employing computer programs // Nucleic Acids Research. 1979. Vol. 6, no. 7. Pp. 2601–10.

3 Anderson S. Shotgun dna sequencing using cloned dnase i-generated fragments // Nucleic Acids Research. 1981. Vol. 9, no. 13. Pp. 3015–27.

4 Schuster S. C. Next-generation sequencing transforms today's biology // Nature Methods. 2008. Vol. 5. Pp. 16–18.

5 http://www.illumina.com/

6 http://www.iontorrent.com/

7 Roach J., Boysen C., Wang k., Hood L. Pairwise end sequencing: a unified approach to genomic mapping and sequencing // Genomics. 1995. Vol. 26. Pp. 345–353.

8 http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml

9 Peter J. A. Cock1,, Christopher J. Field2, Naohisa Goto, Michael L. Heuer, Peter M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants // Nucleic Acids Research. Vol. 38, no 6. Pp. 1767–1771.

10 Ewing B, Green P. Base-calling of automated sequencer traces using phred. II. Error probabilities. // Genome Res. 8(3). Pp. 186–194

11 de Bruijn N. G. A Combinatorial Problem // Koninklijke Nederlandse Akademie v. Wetenschappen. Vol. 49. Pp. 758–764.

12 Miller JR, Koren S, Sutton G. Assembly algorithms for next-generation sequencing data // Genomics. 2010 Jun. 95(6). Pp. 315–327.

13 Kelley D., Schatz M., Salzberg S. Quake: quality-aware detection and correction of sequencing errors. // Genome Biology. 2010. Vol. 11. No. 11. Pp. R116.

14 Kao W.-C., Chan A., Song Y. ECHO: A reference-free short-read error correction algorithm. // *Genome Research. 2011. Vol. 21. No. 7. Pp. 1181–1192.*

15 Medvedev P., Eric Scott, Boyko Kakaradov, Pavel Pevzner. Error correction of high-throughput sequencing datasets with non-uniform coverage. // Bioinformatics. 2011. Vol. 27. No. 13. Pp. i137–i141.

16 Hamming R, Error detecting and error correcting codes // Bell System Technical Journal. Vol. 29. No 2. Pp. 147–160.

17 Ilie L., Fazayeli F., Ilie S. HiTEC: accurate error correction in high-throughput sequencing data. // Bioinformatics. 2011. Vol 27. No 3. Pp. 295–302.

18 Andreas Sundquist, Mostafa Ronaghi, Haixu Tang, Pavel Pevzner, Serafim Batzoglou. Whole-Genome Sequencing and Assembly with High-Throughput, Short-Read Technologies.

19 Jonathan Butler, Iain MacCallum, Michael kleber, Ilya A. Shlyakhter, Matthew k. Belmonte, Eric S. Lander, Chad Nusbaum, David B. Jaffe. ALLPATHS: De novo assembly of whole-genome shotgun microreads.

20 Zerbino D. R., Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. // Genome Research. 2008. Vol. 18, no. 5. Pp. 821–829.

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

22 Li R., Zhu H., et al. De novo assembly of human genomes with massively parallel short read sequencing. // Genome Res. 2010. Vol. 20. No. 2. Pp. 265–272.

23 Brown C., et al. A Reference-Free Algorithm for Computational Normalization of Shotgun Sequencing Data. // arXiv:1203.4802 [q-bio.GN]

24 Simpson J. T., Wong K., Jackman S. D., Schein J. E., Jones S. J. M., Birol I. Abyss: a parallel assembler for short read sequence data. // Genome Res. Jun 2009. Vol. 19, no. 6. Pp. 1117–1123.

25 http://genome.ifmo.ru

26 http://mira-assembler.sourceforge.net