Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Александров Антон Вячеславович

Разработка метода удаления ошибок из набора чтений нуклеотидной последовательности

Научный руководитель: ассистент кафедры КТ СПбГУИТМО
Ф. Н. Царев

Санкт-Петербург
2011

Оглавление

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, допускают при чтении ошибки, которые необходимо исправить.

Объект исследования - задача исправления ошибок в чтениях нуклеотидной последовательности.

Исследование состоит из следующих частей:

- разработать алгоритм эффективного исправления ошибок в чтениях нуклеотидной последовательности;
- реализовать разработанные алгоритмы в виде программного обеспечения ЭВМ;
- экспериментально проверить эффективность разработанного метода.

Научная новизна. В бакалаврской работе разработан новый метод исправления ошибок в чтениях нуклеотидной последовательности, эффективный для больших геномов.

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.

Глава 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.

Существет несколько компаний, выпускающих устройства для получения коротких чтений. Самыми распространенными на рынке этих устройств являются продукты компании *Illumina* [5].

1.2.5. Парные чтения

В последнее время популярным стало использование так называемых *парных чтений* [5]. 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.

1.2.6. 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*).

В выходных данных секвенаторов компнии Illumina последний тип ошибок встречается значительно (на несколько порядков) чаще, чем остальные, поэтому имеет смысл сконцентрироваться на исправлении именно ошибок замены.

1.2.7. 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.

В целом вероятность ошибок обычно невелика (порядка 0.001), но верна также тенденция сильного ухудшения качества нуклеотидов по мере приближения к концу чтения (для секвенаторов Illumina).

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

1.2.7.1. Формат FASTA

Файл в формате FASTA [6] 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).

1.2.7.2. Формат MultiFASTA

Формат MultiFASTA [6] представляет собой модифицированный формат FASTA, допускающий хранение нескольких последовательностей в одном файле. Строки, начинающиеся с «>», в этом формате используются для обозначения начала новой последовательности.

1.2.7.2. Формат FASTQ

Форматы FASTA и MultiFASTA не предусматривают возможности хранения величин качеств для нуклеотидов. Формат FASTQ [7] 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) [8] 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 Брюина

Многие современные сборщики генома используют в процессе сборки подграф так называемого *графа де Брюина* (рис. 1) [9]. 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.

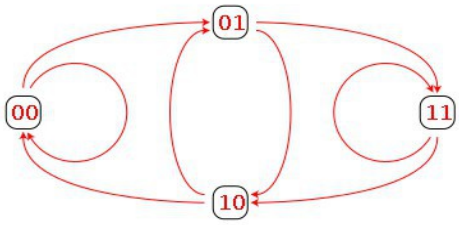

Рис.1. Пример графа де Брюина ($k = 2$) [9]

Если считать, что в геноме нет повторов (строк длины хотя бы *k*, встречающихся хотя бы два раза), то длинные пути в подграфе графа де Брюина 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 Брюина.

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 сегодняшний день неизвестно способов улучшить этот результат. На рис. 2 изображена схема процесса сборки скэффолдов из контигов, собранных из парных чтений.

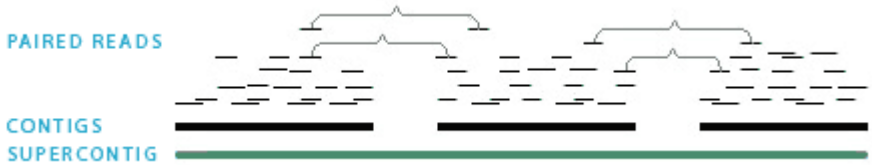

Рис. 2. Схематическое изображение процесса сборки.

1.3.2.4. Оценка для результатов

Для оценки результатов работы сборщика и, как следствие, для сравнения одного сборщика с другим обычно используется так называемая *величина N50*, или просто *N50* [10].

Пусть имеется набор контигов, для которого нужно посчитать N50. Отсортируем этот набор по длине. Пусть *f(N)* — суммарная длина всех контигов с длинами хотя бы N. Тогда N50 — максимальное значение N, для которого f(N) не меньше половины длины генома. Вообще говоря, для любого целого числа *%%*, лежащего в промежутке от 1 до 100, *N%%* - максимальное значение N, для которого f(%%) не меньше %% процентов от длины генома.

Заметим, что N50 тем больше, чем больше собралось больших контигов. Таким образом, эту величину можно использовать в качестве показателя эффективности работы сборщика. Помимо N50 часто используются такие значения, как N25 и N75.

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

2.1. Протокол SHRAP

SHRAP (Short Reads Assembly Protocol) — протокол сборки длинных геномов, основанный на иерархическом секвенировании (рис. 3) [11]. 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.

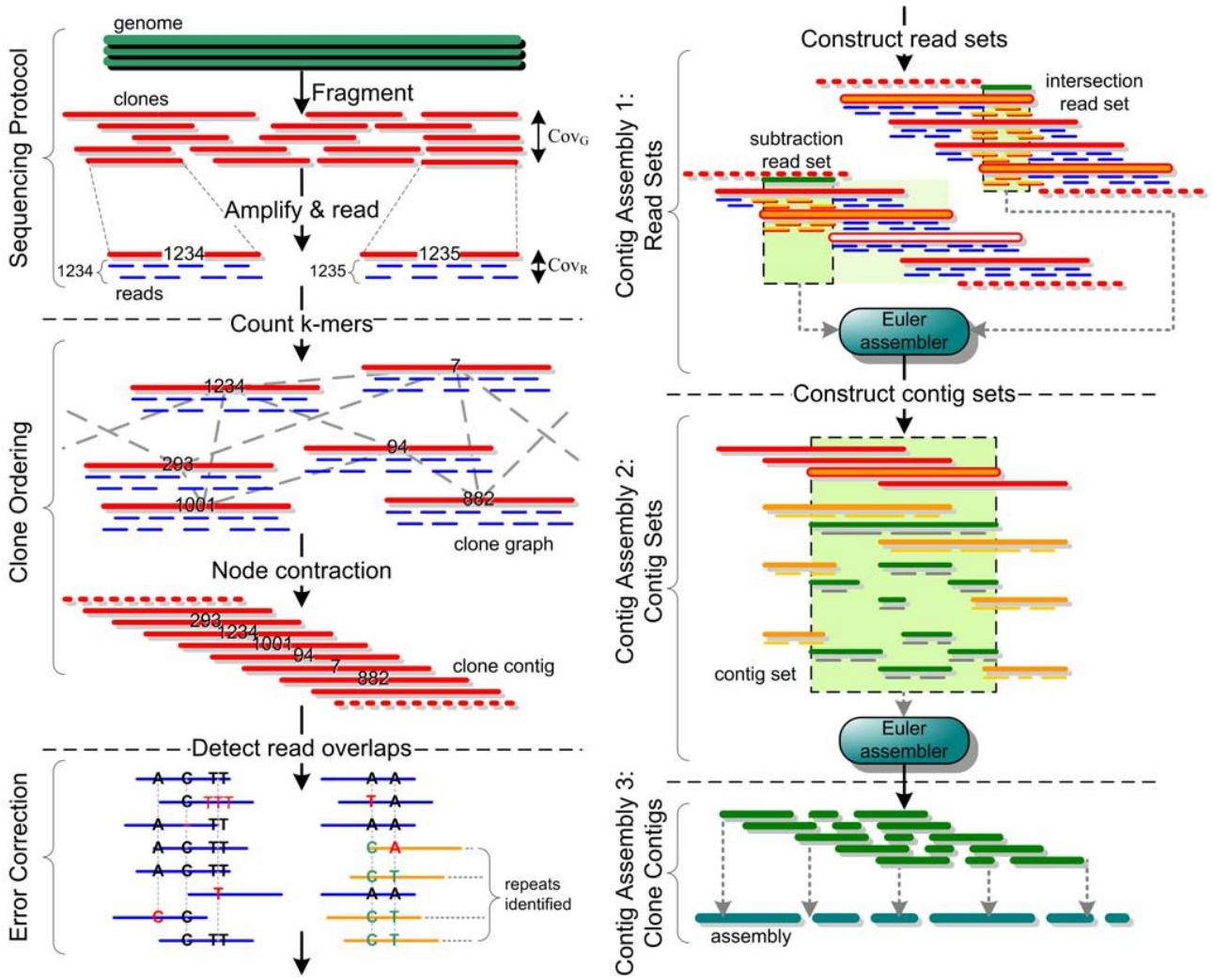

Рис. 3. Схема работы сборщика, основанного на иерархическом секвенировании.

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.

2.2. Сборщик AllPaths

Алгоритм исправления ошибок здесь [12] 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.

2.3. Сборщик Velvet

В этом подходе [13] применяется разновидность *графа де Брюина* (*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 $\widetilde{\widetilde{A}}$.

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 $\widetilde{\widetilde{B}}$ в $\widetilde{\widetilde{A}}$ (рис. 4).

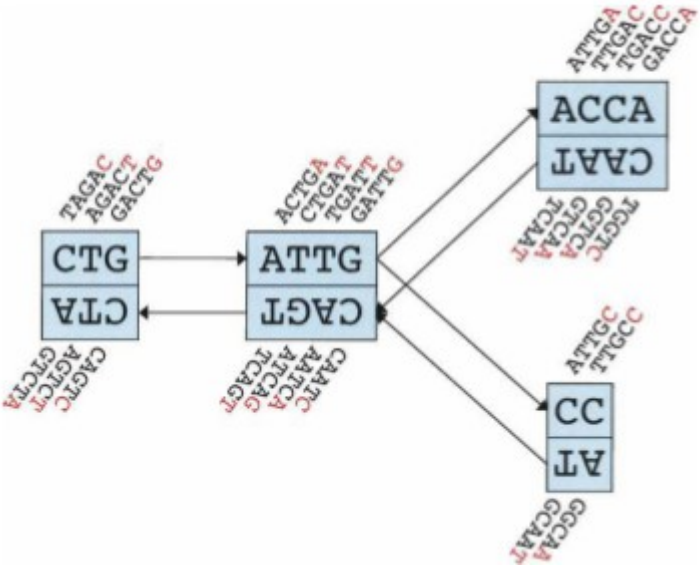

Рис. 4. Пример графа де Брюина для $k = 5$.

Построение графа де Бриюна — 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.

2.3.1. 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.

2.3.2. 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 (рис. 5). Заметим, что модифицирование путей — операция довольно дорогостоящая из-за дополнительной информации, хранящейся в графе.

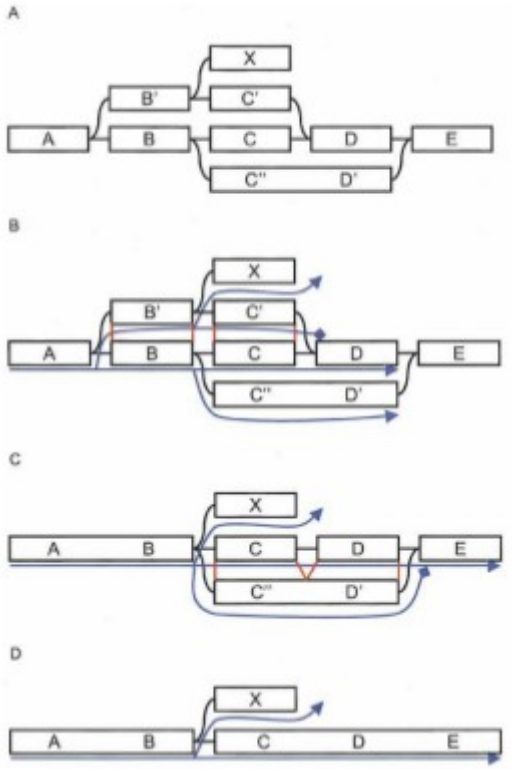

Рис. 5. Пример работы алгоритма удаления пузырей.

2.3.3. 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.

2.4. Алгоритм EULER-USR

Алгоритм [14] 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.

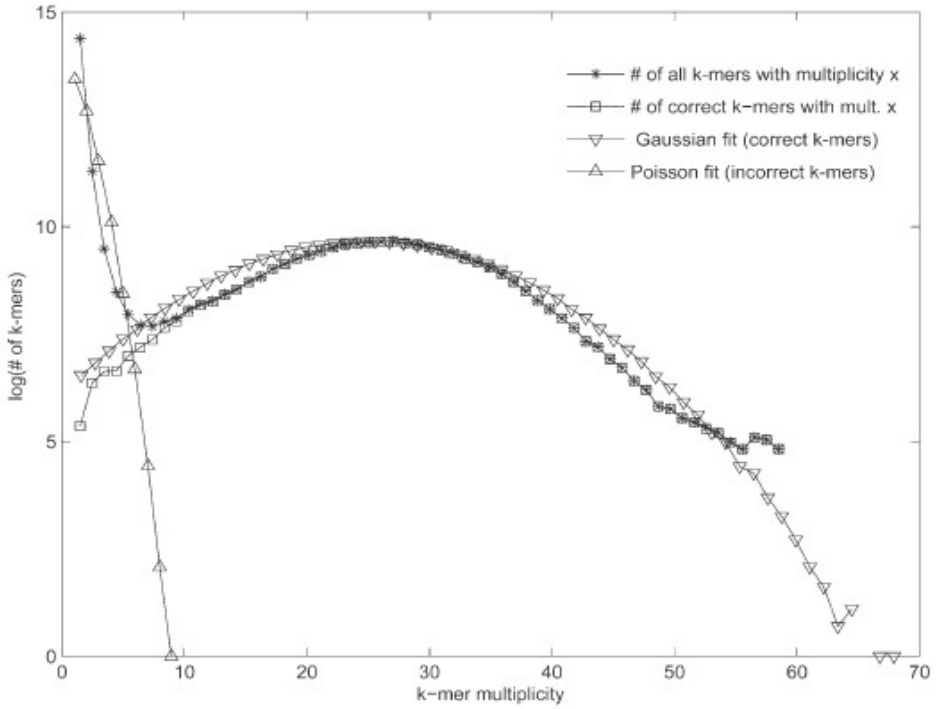

Рис. 6. Экспериментально полученное распределение *k*-меров.

2.5. 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 Брюина и работают с графовыми структурами, другие же основаны на частотном анализе *k*-меров. Все они довольно требовательны к вычислительным ресурсам и не очень хорошо масштабируются.

Глава 3. Описание предлагаемого метода

3.1. Введение

Предлагаемый в данной работе метод основан на частотном анализе содержащихся в чтениях *k*-меров и не использует графа де Брюина. 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. Последнее соображение вытекает не только из увеличения вероятности попадания чтения на конкретную подстроку при увеличении ее длины, но и из факта наличия в чтениях ошибок и маленькой вероятности того, что одна и та же ошибка была допущена в одной и той же позиции генома более одного раза (если считать, что ошибочные прочтения различных позиций генома являются независимыми событиями, а средняя вероятность ошибки равна *p*, то вероятностный вес конкретной подстроки длины *k*, содержащей n ошибок, составляет $C_k^n(1-p)^{k-n}p^n(\frac{1}{3})^n$, что при $k=30, n=1, p=0.001$ равно примерно 0.009).

3.2. Идея метода

Для каждого *k*-мера, присутствующего в чтениях (прямых или обратно-комплементарных) хотя бы раз, подсчитаем, сколько раз он встречается в чтениях. На основании этой статистики все *k*-меры можно разделить на 2 группы — «надежные» *k*-меры и «подозрительные». «Надежные» *k*-меры — это

те, которые встречаются в чтениях достаточно большое число раз — не меньше значения некоторого порога *t*, которое является параметром алгоритма. С «надежными» *k*-мерами делать ничего не нужно, предполагается, что в них ошибок нет. «Подозрительные» же *k*-меры вызваны либо плохим покрытием тех частей генома, откуда они были прочитаны, либо, что гораздо более вероятно, наличием в них одной или нескольких ошибок, которые надлежит исправить.

После выделения «подозрительных» *k*-меров для каждого из них необходимо решить, в какой именно позиции могла быть совершена ошибка. Для этого предлагается перебрать все позиции *k*-мера (их ровно *k* штук) и все возможные нуклеотиды, попробовать заменить имеющийся нуклеотид на перебираемый и проанализировать получившийся *k*-мер. Если новый *k*-мер попадает в группу «надежных», значит, возможно, рассматриваемый *k*-мер является результатом его ошибочного прочтения. Если в течение перебора был найден только один «надежный» *k*-мер, получающийся из «подозрительного» путем замены одного нуклеотида на другой, полагается, что данный «подозрительный» *k*-мер исправлен, а соответствующее исправление запоминается. Если таких *k*-меров несколько, неясно, какое из исправлений запоминать, поэтому в таких случаях «подозрительные» *k*-меры не исправляются. И наконец, если не было найдено ни одного способа исправить «подозрительный» *k*-мер, полагается, что в нем было совершено больше одной ошибки, после чего запускается аналогичная процедура, но с исправлением не одного, а сразу двух нуклеотидов. Аналогичным образом можно пытаться изменить не только пары, но и тройки, а также кортежи из большего числа нуклеотидов, однако данное обобщение ощутимо сказывается на быстродействии алгоритма.

К выбору величины *k* следует подходить достаточно серьезно, так как этот параметр сильно влияет на работу алгоритма. При выборе значения этого параметра следует учитывать следующие простые, но важные соображения:

- Величина k должна быть значительно меньше длины чтений. Если длина k-мера будет сравнима с длиной чтения, то большинство k-меров будет встречаться в чтениях один раз, что не даст никакой информации для исправления ошибок.
- Величина k должна быть достаточно большой, чтобы вероятность того, что случайный k-мер заданной длины встречается в геноме, была ничтожно маленькой. В противном случае некоторые k-меры, содержащие ошибку, не будут исправлены только потому, что есть достаточно большая вероятность того, что эти k-меры на самом деле встречаются в геноме, а тогда они могут много раз встречаться в чтениях.

3.3. Предлагаемый алгоритм

Для начала отметим, что качество нуклеотидов для чтений Illumina сильно ухудшается к концу чтения, поэтому для данного алгоритма имеет смысл обрезать куски чтений плохого качества, так как, с одной стороны, благодаря им образуется много «подозрительных» k-меров, с другой стороны, их не получится исправить из-за большого числа ошибок. Поэтому от каждого отрезается наидлиннейший префикс, в котором каждый нуклеотид имеет вероятность ошибки меньше 0.1 (рис. 7).

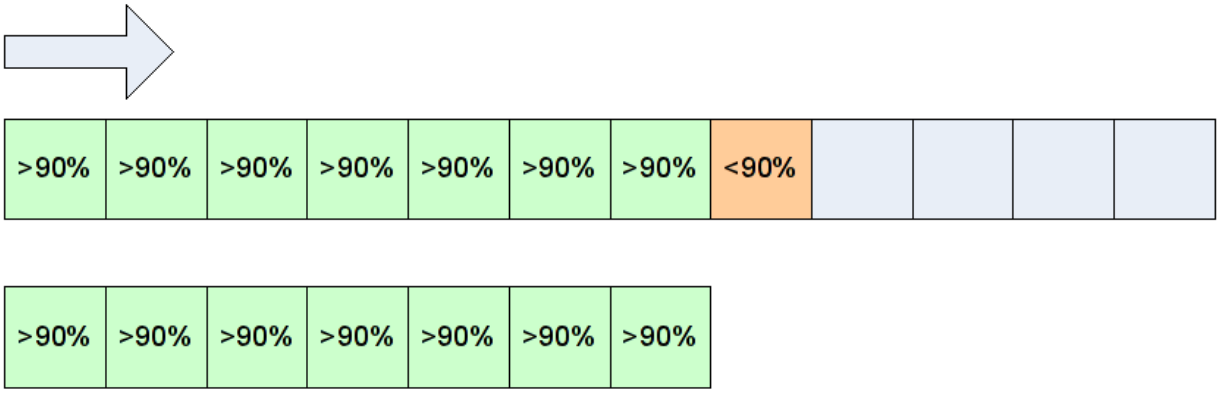

Рис. 7. Обрезание чтений исходя из качества нуклеотидов в них

Также важно заметить, что все *k*-меры можно разбить на группы на основании префикса небольшой длины, с которого они начинаются. Если при этом не исправлять ошибки в префиксе, то исправление не выводит *k*-мер из группы, что позволяет исправлять ошибки отдельно в группах, то есть еще больше уменьшить количество потребляемой алгоритмом оперативной памяти. Это не повлечет за собой неисправленные ошибки в префиксах, так как при обращении *k*-мера его префикс становится суффиксом (рис. 8).

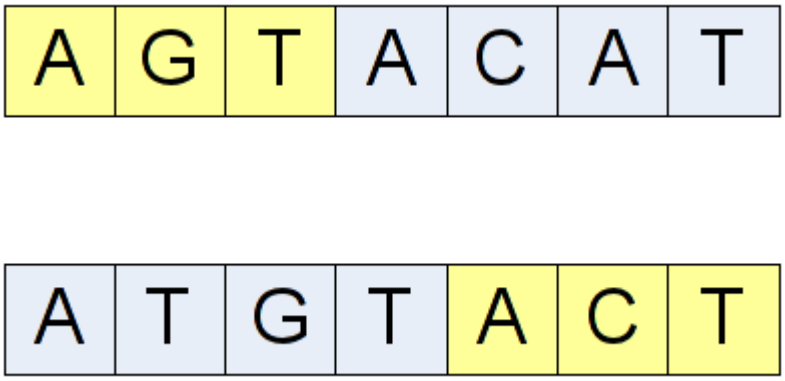

Рис. 8. *k*-мер и его обратно-комплементарная копия. Желтым цветом выделен префикс, на основе которого решается, в какую группу попадет этот *k*-мер

Таким образом, алгоритм состоит из следующих шагов:

1. Обрезание *k*-меров на основании качества.
2. Разбиение *k*-меров на группы на основании префиксов.
3. Выполнение для каждой группы:
    1. Сбор статистики по *k*-мерам.
    2. Исправление ошибок.

Важно отметить, что алгоритм поиска ошибок в *k*-мерах легко распараллеливается, так как для обработки одного *k*-мера ему требуется только доступ на чтение к общей структуре данных, хранящей статистику по содержанию *k*-меров в чтениях, а также кратковременный доступ на запись для сохранения результата.

3.4. Выбор параметра *t*

Существует два основных подход к выбору параметра *t* — теоретический и практический. При теоретическом подходе считается, что распределение частот *k*-меров является смесью двух распределений — распределения, соответствующего «надежным» *k*-мерам, и распределения, соответствующего «подозрительным» *k*-мерам. Считается, что позиции ошибок при чтении независимы. При такой модели оба указанных выше распределения являются распределениями Пуассона с разными параметрами.

Для нахождения значения порога находится первый локальный минимум этого распределения. Частота, соответствующая этому минимуму, считается пороговым значением.

В практическом подходе не делается никаких предположений о вероятностных моделях. Вместо этого распределение частот *k*-меров строится прямо из данных чтений. По полученному распределению аналогично теоретическому подходу частота, соответствующая первому локальному минимуму, выбирается в качестве порога.

3.5. Приблизительная оценка эффективности предложенного метода

Простота предлагаемого метода позволяет довольно просто оценить, насколько он эффективен. При этом возникают следующие соображения:

1. Предположим, что в каждом *k*-мере исправляется не более *N* ошибок. Тогда те *k*-меры, в которых больше *N* ошибок, точно не будут исправлены. Однако из этого не следует, что нужно выбирать большое значение *N*, увеличение его влечет за собой увеличение трудоемкости метода. Кроме того, при больших значениях *N* велика вероятность наличия *k*-мера, который можно исправить несколькими способами, так как уменьшается число позиций *k*-мера с фиксированными нуклеотидами.

2. Для того, чтобы *k*-мер с ошибкой мог быть исправлен, необходимо и достаточно, чтобы его безошибочный вариант был прочитан хотя бы *t* раз, а также не было другого «надежного» *k*-мера, отличающегося от этого не более, чем в *N* позициях.

Таким образом, при правильном (соответствующем исходным данным) выборе параметров предлагаемый метод не сможет обработать лишь 3 типа *k*-меров:

- Те, которые были хотя бы *t* раз прочитаны неверно (возможно, по-разному) и отличающиеся от них в не более, чем *N* позициях. Вероятностная мера *t* одинаково неверных *k*-меров равна вероятностной мере одного, возведенной в степень *t*. При значениях *t,* не меньших 3, значение этой меры для $k=30, N=1, p=0.001$ имеет порядок -6 или меньший.
- Те, которые неверно определились как ошибочные в связи с повторами в геноме. Вероятностная мера этих *k*-меров вычисляется при помощи биологических соображений и тоже достаточно мала (имеет порядок -3 или меньший).
- Те, в которых больше *N* ошибок. При достаточно большом *N* таких доля *k*-меров также имеет маленький порядок (-4 и ниже).

Глава 4. Реализация алгоритма и полученные результаты

4.1. Реализация алгоритма

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

Ниже приведена реализация функции `findNFixes`, которая осуществляет поиск возможностей исправления ошибок в *k*-мере. Максимальное число возможных ошибок (*n*) передается в функцию как параметр. Предпочтение отдается меньшему числу ошибок — как только удалось исправить какое-то, не превышающее *n*, число ошибок так, что *k*-мер стал «надежным», функция прекращает свою работу.

Параметр *DEFINITELY_GOOD_THRESHOLD* — это упоминавшийся выше параметр *t*.

```
    private int findNFixes(long str, int n, int begin) {
        if (n == 0) {
            Info minfo = hm.get(str);
          if ((minfo != null) && (minfo.count.intValue() >=
DEFINITELY_GOOD_THRESHOLD)) {
                newStr = str;
                return 1;
            } else {
                return 0;
            }
        }

        int k = 0;
       int maxPos = (begin == 0) ? (len - 1) : begin +
MAXIMAL_FIXES_DISTANCE;
        maxPos = Math.min(maxPos, len - n);
          for (int pos = begin; (k < 2) && (pos <= maxPos);
pos++) {
            for (long xor = 1; xor <= 3; xor++) {
                long nstr = str ^ (xor << (2 * pos));
                int r = findNFixes(nstr, n - 1, pos + 1);
                if (r == 2) {
```

```
                    k = 2;
                    break;
                }
                if (r == 0) {
                    continue;
                }
                k++;
                if (k == 2) {
                    break;
                }
            }
        }
        return k;
    }
```

4.2. Результаты

4.2.1. Проект dnGASP

Описанный в данной работе метод исправления ошибок был разработан и применен в рамках проекта dnGASP, в котором участникам предлагалось восстановить синтетическую последовательность ДНК длиной около 1,8 миллиардов нуклеотидов, используя для этого любые уже имеющиеся или специально разработанные средства. При этом хотя исходный геном был синтетическим, он представлял собой смесь геномов различных живых организмов, то есть задача была близка к реальной.

В качестве исходных данных участникам была предоставлена библиотека парных чтений синтезированного генома. Эти чтения обеспечивали достаточно большое покрытие генома — 44. Также в них искусственным образом были внесены ошибки, чтобы сделать задачу максимально похожей на реальную.

Описываемый метод был использован в качестве одного из этапов алгоритма сборки генома. Алгоритм запускался на 24-ядерном компьютере с 24 ГБ оперативной памяти, что слишком мало для того, чтобы для исправления ошибок можно было использовать уже существующие методы.

Для оценки эффективности работы приведем некоторую статистику по *k*-мерам, содержащимся в чтениях, собранную два раза — до исправления ошибок и после него.

До запуска алгоритма в исходных данных было 6,5 миллиардов различных *k*-меров, из которых 3 миллиарда «надежных». Алгоритм работал около суток, после чего в данных стало 3,9 миллиардов (что на 40% меньше, чем в начале) различных *k*-меров, из которых 3,3 миллиарда «надежных». Таким образом, количество «подозрительных» *k*-меров уменьшилось с 3,5 миллиардов до 0,6 миллиардов (примерно на 83%).

Число *k*-меров после исправления ошибок стало достаточно маленьким, чтобы граф де Брюина, используемый на последующих этапах сборки генома, мог уместиться в оперативную память среднего по вычислительной мощности сервера (примерно 64 ГБ).

На рисунках 9 и 10 приведены графики частот *k*-меров в данных до исправления ошибок и после него. Данные, по которым построены графики, приведены в приложении 1.

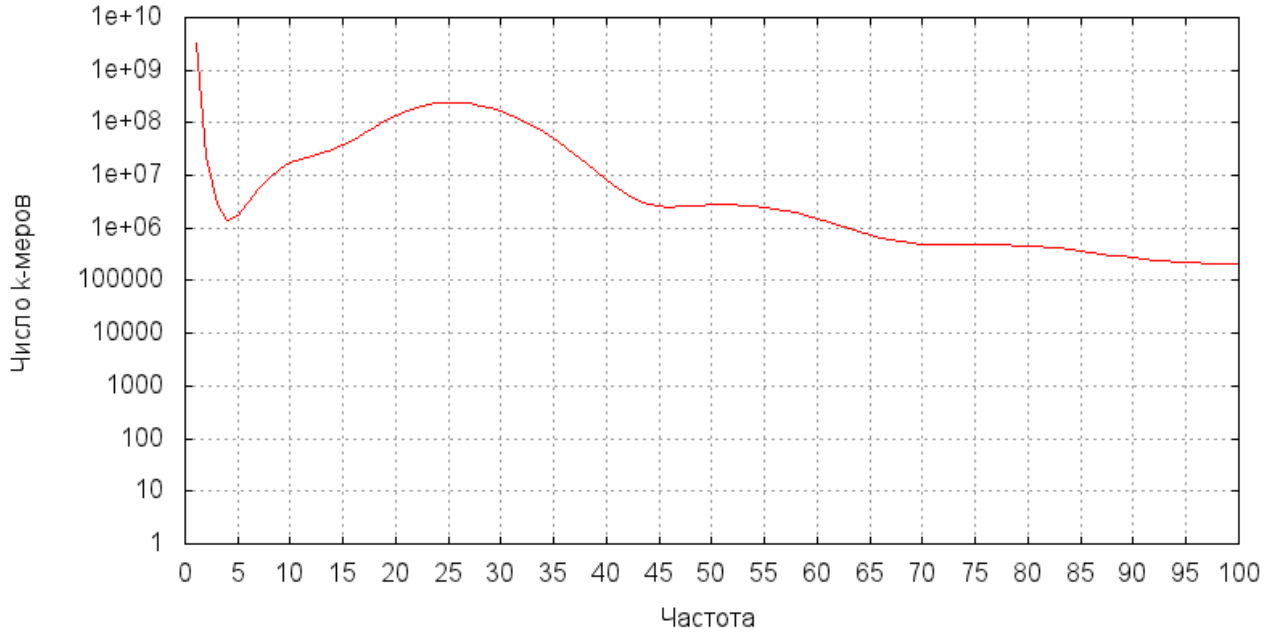

Рис. 9. Распределение частот *k*-меров в данных dnGASP до исправления ошибок.

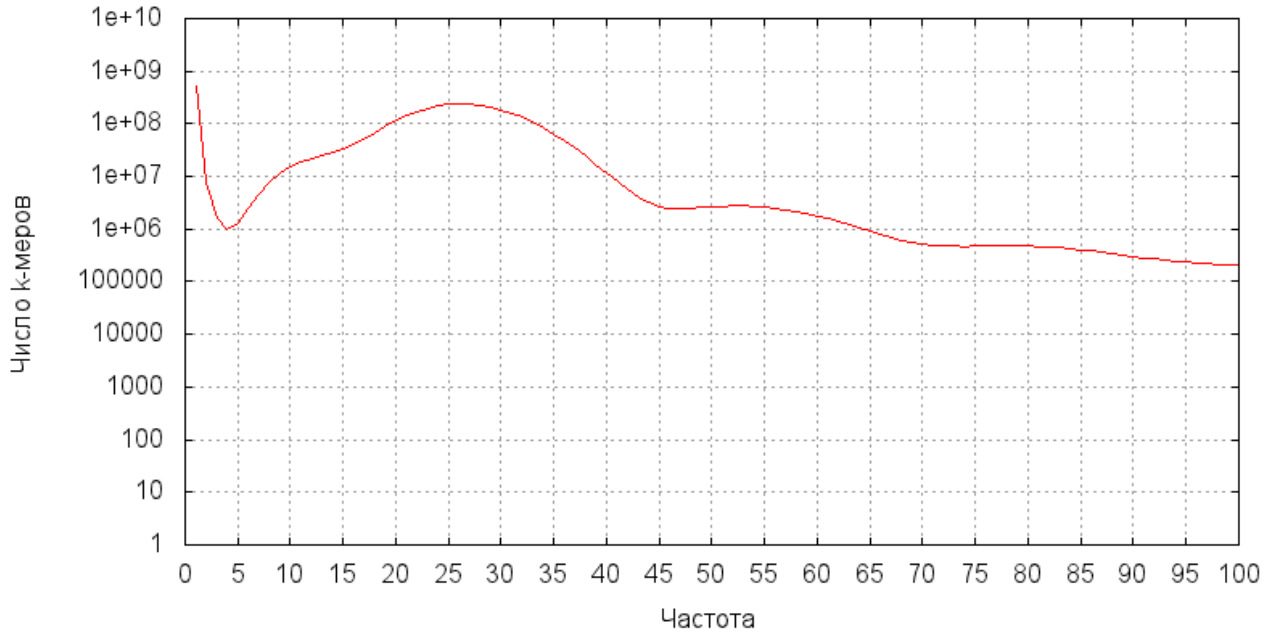

Рис. 10. Распределение частот *k*-меров в данных dnGASP после исправления ошибок.

4.2.2. Итоги и дальнейшие направления работы

Был разработан метод исправления ошибок, основанный на частотном анализе *k*-меров. Также было проведено экспериментальное исследование разработанного метода, показавшее работоспособность метода на данных, близких к реальным. Разработанный метод может эффективно использовать как достаточно маленький объем ресурсов, так и большой. В настоящий момент исследуются возможные изменения, которые можно осуществить для улучшения эффективности и быстродействия предложенного метода. Например, планируется в большей степени учитывать информацию о качестве прочитанных нуклеотидов, предоставляемую секвенаторами. Кроме того, рассматривается возможность учета взаимосвязи *k*-меров в чтениях.

Заключение

Проведенная работа имеет следующие результаты:

- разработан алгоритм исправления ошибок в наборе чтений нуклеотидной последовательности, превосходящий существующие методы по масштабируемости, а также гораздо менее требовательный к вычислительным ресурсам;
- проведен анализ применимости данного метода к набору чтений;
- разработанный алгоритм реализован в виде программы для ЭВМ;
- разработанная программа успешно протестирована на синтетических данных, составленных из геномов живых организмов;

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. P. 2601–10.

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

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

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

[6] http://en.wikipedia.org/wiki/FASTA_format

[7] http://en.wikipedia.org/wiki/FASTQ_format

[8] *Ewing B, Green P.* Base-calling of automated sequencer traces using phred. II. Error probabilities. // *Genome Res*. 8(3):186-194

[9] http://en.wikipedia.org/wiki/De_bruijn_graph

[10] http://en.wikipedia.org/wiki/N50_statistic

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

[12] *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.

[13] *Daniel R. Zerbino, Ewan Birney.* Velvet: Algorithms for de novo short read assembly using de Bruijn graphs.

[14] *Mark J. Chaisson, Dumitru Brinza and Pavel A. Pevzner.* De novo fragment assembly with short mate-paired reads: Does the read length matter?

Приложение 1. Данные до и после исправлений

n	Количество *k*-меров, содержащихся в чтениях *n* раз (до исправления ошибок)	Количество *k*-меров, содержащихся в чтениях *n* раз (после исправления ошибок)
1	3130509297	532770846
2	23978495	7370346
3	2833469	1644074
4	1405930	964250
5	1725467	1344162
6	3108862	2532697
7	5542599	4654354
8	8933360	7702092
9	12938954	11420948
10	17085135	15424645
11	20881921	19152192
12	24230586	22400507
13	27471307	25229360
14	31523923	28305739
15	37799720	32806030

16	47684652	40057072
17	62325861	51332438
18	82298947	67490362
19	107159743	88729412
20	135555086	114335600
21	165252192	142632754
22	193402998	171280538
23	217247228	197635546
24	234200378	219115389
25	242664845	233477182
26	241828429	239406922
27	232140446	236370280
28	214891711	225145906
29	192086970	207101590
30	166024726	184147485
31	138955979	158526266
32	112622571	132191488
33	88545487	106922872

34	67641312	83997181
35	50267171	64138710
36	36386246	47678694
37	25729448	34547920
38	17850736	24481497
39	12228555	17031140
40	8359047	11704969
41	5816821	8024776
42	4216106	5603302
43	3274746	4074792
44	2777564	3170799
45	2556617	2683700
46	2511250	2471540
47	2553259	2431984
48	2632677	2475747
49	2718603	2568770
50	2782930	2658262
51	2808657	2728760

52	2791588	2770382
53	2735871	2770092
54	2632558	2726030
55	2497079	2642596
56	2322248	2519509
57	2133322	2363632
58	1927375	2178675
59	1722393	1992026
60	1520430	1784901
61	1325020	1583972
62	1153745	1389619
63	997751	1213840
64	861306	1049738
65	753160	909628
66	668082	790598
67	599797	695844
68	549866	620804
69	517298	564282

70	494095	524151
71	482642	497198
72	477257	479743
73	475630	474016
74	477263	470737
75	479429	470848
76	480998	475365
77	479848	477194
78	476366	480184
79	468307	479912
80	458905	471901
81	445166	466868
82	429245	455716
83	409881	441110
84	390765	423482
85	367541	406266
86	346968	384481
87	326814	362576

88	307798	343473
89	289152	321062
90	272594	301967
91	258091	283574
92	246713	268498
93	236797	255458
94	227976	241154
95	222284	233104
96	219401	224752
97	216547	219596
98	215956	216093
99	214181	213106
100	212337	211109