федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

На правах рукописи

Казаков Сергей Владимирович

Автоматизация сборки генома и сравнительного анализа метагеномов для обучения геномной биоинформатике

Специальность 05.13.06 – Автоматизация и управление технологическими процессами и производствами (образование)

Диссертация на соискание ученой степени
кандидата технических наук

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

Санкт-Петербург – 2016 г.

ОГЛАВЛЕНИЕ

Определения....6

Список сокращений....9

Введение....10

Глава 1. Дезоксирибонуклеиновая кислота (ДНК), секвенирование ДНК и анализ данных секвенирования....20

1.1. ДНК....20

1.1.1. Секвенирование ДНК....22

1.1.2. Существующие методы секвенирования....23

1.1.2.1. Метод обрыва цепи....24

1.1.2.2. Метод дробовика....24

1.1.2.3. Высокопроизводительные методы секвенирования....25

1.1.2.4. Характеристики секвенаторов....26

1.1.3. Метагеномное секвенирование....28

1.2. Сборка генома....29

1.2.1. Постановка задачи восстановления геномной последовательности....30

1.2.2. Задача о наименьшей общей надстроке....30

1.2.2.1. Графовое представление задачи о наименьшей общей надстроке....35

1.2.2.2. Точные алгоритмы решения....37

1.2.2.3. Приближенные алгоритмы решения....38

1.2.2.4. Недостатки сборки генома через поиск наименьшей общей надстроки....40

1.2.3. Сборка генома на основе данных секвенирования с помощью гибридизации....41

1.2.4. Графовая постановка задачи о сборке генома....44

1.2.5. Методы учета двухцепочечной структуры ДНК....47

1.2.6. Сборка генома с учетом покрытия....48

1.2.7. Методы учета парной информации....49

1.2.8. Подходы к уменьшению используемой памяти при хранении графа де Брейна....53

1.2.9. Существующие программы сборки генома 55
1.2.10. Анализ программ по возможности их использования при обучении59
1.2.11. Использование результатов сборки 60
1.3. Анализ данных метагеномного секвенирования 61
1.3.1. Сравнительный анализ в метагеномике 62
1.3.2. Существующие подходы для сравнительного анализа метагеномов..63
1.4. Задачи, решаемые в диссертационной работе.....69
Выводы по главе 1.....71

Глава 2. Автоматизированный метод сборки генома *de novo* на основе совместного применения графа де Брейна и графа перекрытий 72

2.1. Анализ затрат памяти на хранение графов.....72
2.2. Метод сборки генома.....75
2.2.1. Исправление ошибок.....76
2.2.2. Сборка квазиконтигов 81
2.2.3. Сборка контигов.....86
2.2.3.1. Поиск перекрытий между квазиконтигами 87
2.2.3.2. Поиск и удаление покрываемых квазиконтигов 91
2.2.3.3. Поиск ненайденных перекрытий с помощью найденных 91
2.2.3.4. Удаление транзитивных перекрытий 93
2.2.3.5. Построение графа перекрытий и его упрощение 94
2.2.3.6. Поиск и вывод путей в графе 98
2.2.3.7. Нахождение консенсуса для путей 99
2.3. Реализация предложенных подходов.....99
2.4. Экспериментальное исследование 101
2.4.1. Использованные наборы данных 101
2.4.2. Методология экспериментов 102
2.4.3. Результаты экспериментов 104
2.4.4. Дополнительные эксперименты по анализу требуемых вычислительных ресурсов 107
Выводы по главе 2.....111

Глава 3. Автоматизированный метод сравнительного анализа метагеномов, основанный на анализе компонент связности в графе де Брейна 112

3.1. Метод сравнительного анализа метагеномов 112
3.1.1. Подсчет надежных *k*-меров 115
3.1.2. Выделение неветвящихся путей 116
3.1.3. Выделение компонент связности 116
3.1.4. Построение характеристических векторов 117
3.1.5. Построение матрицы расстояний 118
3.1.6. Выполнение кластеризации и отображение графических результатов 119
3.2. Реализация предложенного подхода 119
3.3. Экспериментальное исследование 121
3.3.1. Использованные метагеномные наборы данных 121
3.3.2. Методология экспериментов 123
3.3.3. Эксперименты с симулированными метагеномами: сравнение точности получаемых матриц 125
3.3.4. Эксперименты с метагеномами микробиоты метро Нью-Йорка: сравнение работы существующих решений 128
3.3.5. Эксперименты с метагеномами микробиоты кишечника человека: оценка работоспособности *MetaFast* на больших наборах данных 132
3.3.6. Эксперименты с метагеномами виром озер: сравнение возможностей анализа для новых микробиот 134
Выводы по главе 3 141
Глава 4. Внедрение результатов работы 142
4.1. Внедрение результатов работы в учебный процесс в Санкт-Петербургском политехническом университете Петра Великого 142
4.2. Внедрение результатов работы в учебный процесс в Университете ИТМО 144
4.3. Внедрение результатов работы в Казанском (Приволжском) Федеральном Университете 145
Выводы по главе 4 147
Заключение 148
Список источников 149

Печатные издания на русском языке.....149
Печатные издания на английском языке 149
Ресурсы сети Интернет.....158
Публикации автора 159
Статьи в журналах из перечня ВАК.....159
Публикации в рецензируемых изданиях, индексируемых Web of Science или Scopus.....160
Материалы конференций с участием автора 160
Приложение 1. Свидетельства о регистрации программ для ЭВМ.....163
Приложение 2. Пример отчета по лабораторной работе.....168

ОПРЕДЕЛЕНИЯ

В настоящей диссертационной работе применяются следующие термины с соответствующими определениями.

Биоинформатика – совокупность методов и подходов, применяемых для обработки биологических данных.

Геномная биоинформатика – область биоинформатики, ориентированная на изучение геномов живых организмов.

Сравнительная метагеномика – область метагеномики, которая занимается вопросами соотношения между разными метагеномами.

Геном – совокупность наследственного материала, заключенного в клетке организма.

Метагеном – совокупность генетического материала микроорганизмов (бактерий, архей, вирусов) из одной среды обитания (почва, водные ресурсы, кишечник человека и т. п.).

ДНК (*дезоксирибонуклеиновая кислота*) – биологический полимер, обеспечивающий хранение и передачу из поколения в поколение генетической информации.

РНК (*рибонуклеиновая кислота*) – одна из первых форм для хранения биологической информации, по структуре схожа с ДНК.

Секвенирование ДНК – определение нуклеотидной последовательности по имеющемуся образцу ДНК. Результатам секвенирования обычно являются набор чтении нуклеотидной последовательности.

Чтения – небольшие по длине (35–500 нуклеотидов) фрагменты нуклеотидной последовательности.

Библиотека – набор чтений, обычно получаемых при одном запуске секвенатора, которые имеют одинаковые характеристики.

Сборка генома – процесс получения больших фрагментов генома из небольших чтений. Результатом сборки являются контиги или скэффолды.

Сборка генома de novo – задача сборки еще неизвестного генома.

Контиг – непрерывный длинный фрагмент генома (обычно полученный компьютером). При сборке обычно предполагается, что контиги не удается расширить в длине по имеющимся данным.

Квазиконтиг – непрерывный фрагмент генома определенной, обычно небольшой длины (в среднем 100–1000 нукл.).

Скэффолд – набор контигов с заданным порядком их следования и оценками на расстояния между парами соседних контигов.

Референсная (эталонная) последовательность – законченный собранный геном высокого качества (один контиг или один скэффолд).

K-мер – строка длиной *K* символов.

K-мерный спектр – набор всех *k*-меров, присутствующих в нуклеотидной последовательности.

Взвешенный граф – граф, каждое ребро которого имеет вес – вещественное или целое число, сопоставленное с ним.

Граф де Брейна (*de Bruijn graph*) – ориентированный граф, вершинами которого являются *k*-меры, а ребро между двумя вершинами существует, если из *k*-мера, соответствующий первой вершине, можно получить второй путем добавления одного символа в конец первого *k*-мера и удаления одного символа из начала.

Граф перекрытий (*overlap graph*) – взвешенный ориентированный граф, вершинами которого являются чтения, а ребро между двумя вершинами существует, если соответствующие чтения перекрываются (вес ребра равен длине перекрытия).

Метрика N50 – длина такого контига, что все контиги с длиной большей или равной выбранной составляют не менее 50% длины итоговой сборки генома.

Нуклеотид – химическое соединение, являющееся частью ДНК.

Обход в ширину – алгоритм, осуществляющий обход всех вершин, достижимых из заданной, в порядке увеличения расстояния.

Префикс строки – подстрока строки, начинающаяся с первого символа этой строки.

Суффикс строки – подстрока строки, кончающаяся последним символом этой строки.

Бинарный поиск – алгоритм, позволяющий осуществлять поиск элемента в отсортированном массиве за время, пропорциональное двоичному логарифму длины массива.

Хеш-таблица – структура данных, позволяющая хранить пары вида (ключ, значение). Для каждого ключа может храниться не более одного значения. При работе используются хеш-функции, которые вычисляют некоторое числовое значение по заданному элементу.

СПИСОК СОКРАЩЕНИЙ

ДНК – дезоксирибонуклеиновая кислота.

РНК – рибонуклеиновая кислота.

Гб – гигабайт ($2^{30} = 1\,073\,741\,824$ байт).

Мб – мегабайт ($2^{20} = 1\,048\,576$ байт).

Нукл. – нуклеотид.

ОЗУ – оперативное запоминающее устройство.

ПО – программное обеспечение.

SNP (*single nucleotide polymorphism*) – однонуклеотидный полиморфизм (замена одного нуклеотида на другой в некоторой части генома).

CNV (*copy number variation*) – исследование числа копий определенных сегментов генома.

ВВЕДЕНИЕ

Актуальность темы исследования. За последние 40 лет основным методом получения информации о клетке живого существа и процессах, протекающих в ней, стало **секвенирование**. *Секвенирование* дезоксирибонуклеиновой кислоты (*ДНК*) – процесс определения последовательности нуклеотидов в молекуле ДНК. Эта молекула обеспечивает хранение и передачу генетической информации. Иными словами, секвенирование позволяет получить по физической субстанции ДНК или РНК (рибонуклеиновая кислота) ее нуклеотидную последовательность в цифровом (электронном) виде. При этом процесс секвенирования состоит из двух частей – физико-химической (непосредственный процесс «чтения» нити ДНК или РНК) и компьютерной (обработка полученных «сырых» данных). Компьютерная часть обычно называется «*сборкой генома*». Ее наличие обусловлено тем, что физико-химическая часть секвенирования не позволяет получить всю цепочку ДНК целиком, которая необходима для изучения генома, а только ее маленькие фрагменты (*чтения*). Компьютерная часть позволяет решить эту проблему [9]. Таким образом, *сборка генома* – процесс получения больших фрагментов генома (ДНК) из небольших чтений. *Сборка генома de novo* – задача сборки еще неизвестного генома. Методы компьютерного анализа составляют основу *геномной биоинформатики*, которая является составной частью *биоинформатики* и ориентирована на изучение геномов живых организмов.

По мере развития технологий секвенирования развивались и программы для сборки генома. Они становились все более сложными [44], и, как правило, строились на основе модульной архитектуры – состояли из набора модулей, каждый из которых ответственен за выполнение своей задачи (этапа). Эта архитектура обычно является иерархической – каждый этап может состоять из подэтапов. Другими особенностями программ по

сборке генома являются: ориентированность на специалистов узкой направленности (в основном биоинформатиков), возможность работы только под операционной системой *Linux* и требование для работы больших объемов оперативной памяти [48–50, 43]. Поэтому такие программы обычно запускают на серверах или на кластерах. При этом описанные особенности затрудняют использование указанных программ при обучении, так как их установка, настройка и запуск на компьютерах обучающихся, которые обычно являются персональными, плохо осуществимы.

Со временем развитие технологий секвенирования привело к расширению границ его применимости. Секвенирование стало применяться не только для получения генома отдельного организма, но и для анализа набора геномов (метагеном) [12]. *Метагеном* – совокупность геномов микроорганизмов (бактерий, архей, вирусов) из одной среды обитания (почва, водные ресурсы, кишечник человека и т. п.). Компьютерный анализ таких данных включает в себя методы сравнительного анализа набора метагеномов [78, 105], методы определения таксономического состава метагенома (какие бактерии находится в метагеноме) [65, 68, 71–73] и другие. Этому направлению в биоинформатике также необходимо обучать. При этом существующие программы для анализа метагеномов, как и в случае с программами сборки генома, плохо подходят для такого использования, так как являются сложными и труднонастраиваемыми.

Таким образом, разработка автоматизированных методов сборки генома *de novo* и сравнительного анализа метагеномов, которые применимы в образовательном процессе, является **актуальной задачей.**

В соответствии с паспортом специальности 05.13.06 «Автоматизация и управление технологическими процессами и производствами (образование)» диссертация относится к следующей области

исследований: «20. Разработка автоматизированных систем научных исследований».

Цель диссертационной работы – разработка автоматизированных методов сборки генома *de novo* и сравнительного анализа метагеномов, оптимизированных по объему используемой оперативной памяти, а также расширение их области применимости для использования при обучении.

Для этого решаются следующие **основные задачи**:

1. Произвести анализ существующих методов сборки генома *de novo* и сравнительного анализа метагеномов по возможности их применения в образовательном процессе.
2. Разработать автоматизированный метод сборки генома *de novo* на основе совместного применения графа де Брейна и графа перекрытий, оптимизированный по объему используемой памяти.
3. Разработать автоматизированный метод сравнительного анализа метагеномов на основе анализа графа де Брейна, оптимизированный по вычислительным ресурсам.
4. Произвести экспериментальные сравнения программ, реализующих предлагаемые и существующие методы, по метрикам качества получаемых результатов и необходимым вычислительным ресурсам.

Научная новизна. В работе получены следующие новые научные результаты, которые выносятся на защиту:

1. Автоматизированный метод сборки генома *de novo* на основе совместного применения графа де Брейна и графа перекрытий. Программа, разработанная на основе этого метода, позволяет производить сборку малых и средних по размеру геномов на персональных компьютерах под управлением трех самых распространенных операционных систем (*Windows, macOS/OS X, Linux*), что отличает ее от существующих программ.
2. Автоматизированный метод сравнительного анализа метагеномов, основанный на анализе компонент связности в графе де Брейна.

Разработанный метод отличается от существующих тем, что он выполняет «упрощенную» сборку метагеномов вместо стандартной, позволяя значительно сократить требуемые вычислительные ресурсы.

Методы исследований. В работе используются методы теории графов, дискретной математики, теории сложности и математической статистики.

Положения, выносимые на защиту. На защиту выносятся:

1. Автоматизированный метод сборки генома *de novo* на основе совместного применения графа де Брейна и графа перекрытий.
2. Автоматизированный метод сравнительного анализа метагеномов, основанный на анализе компонент связности в графе де Брейна.

Отличия разработанных методов от существующих указаны в разделе *Научная новизна*.

Достоверность научных положений и выводов, полученных в диссертации, подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, результатами экспериментов по использованию предложенных в диссертации методов и их статистическим анализом.

Теоретическое значение работы состоит в том, что показана применимость алгоритмов сборки генома *de novo* и сравнительного анализа метагеномов для работы на персональных компьютерах, обычно применяемых при обучении геномной биоинформатике.

Практическое значение работы состоит в том, что разработанные методы реализованы в виде исполняемых программ с открытым исходным кодом, которые позволяют производить сбоку генома *de novo* и сравнительный анализ метагеномов на персональных компьютерах обучающихся под управлением трех самых распространенных операционных систем (*Windows, macOS/OS X, Linux*). При этом предлагаемые методы позволяют существенно уменьшить объем

используемой оперативной памяти по сравнению с существующими решениями.

Использование и внедрение результатов работы. Результаты диссертационной работы были использованы в учебном процессе в Санкт-Петербургском политехническом университете Петра Великого в рамках магистерской программы «Прикладная математика и информатика. Биоинформатика» (имеется акт внедрения) и в Университете ИТМО при проведении занятий по биоинформатике на кафедре «Компьютерные технологии» (имеется акт внедрения). Результаты работы также использовались в Казанском (Приволжском) Федеральном Университете в лаборатории масс-спектрометрии при выполнении научно-исследовательских работ по анализу геномов шести малоизученных бактерий (имеется акт внедрения).

Апробация результатов работы. Основные результаты работы докладывались на следующих международных и российских конференциях, семинарах и школах:

- VIII Всероссийская межвузовская конференция молодых ученых (2011, Санкт-Петербург);
- II Международная научно-практическая конференция «Постгеномные методы анализа в биологии, лабораторной и клинической медицине: геномика, протеомика, биоинформатика» (2011, Новосибирск);
- XIX Всероссийская научно-методическая конференция «Телематика'2012» (2012, Санкт-Петербург);
- Международная научно-практическая конференция «Постгеномные методы анализа в биологии, лабораторной и клинической медицине» (2012, 2014, Казань);
- Всероссийская научная конференция по проблемам информатики СПИСОК (2012, 2016, Матмех СПбГУ);
- Первый всероссийский конгресс молодых ученых (2012, Санкт-Петербург);

- Первая Международная школа-конференция студентов, аспирантов и молодых ученых «Биомедицина, материалы и технологии XXI века» (2015, Казань);
- Летняя школа по биоинформатике (2015, Москва);
- VII Международная научная конференция «Компьютерные науки и информационные технологии» (2016, Саратов);
- *de novo* Genome Assembly Assessment Project workshop (dnGASP) (2011, Барселона);
- «Bioinformatics 2012» Conference (2012, Стокгольм);
- 8th International Conference on Intelligent Systems and Agents (2014, Лиссабон);
- Moscow Conference on Computational Molecular Biology (2015, Москва).

Личный вклад автора. Автором лично разработаны: идея совместного использования графа де Брейна и графа перекрытий для сборки генома, методы исправления ошибок и сборки контигов при сборке генома, метод выполнения «упрощенной сборки» при анализе метагеномов, а также реализация всех предложенных методов.

Публикации. Основные результаты по теме диссертации изложены в 19 публикациях [101–119], четыре из которых изданы в российских журналах, рекомендованных ВАК [101–104], четыре – в изданиях, индексируемых в международных базах цитирования *Web of Science* [105, 107] и *Scopus* [106, 108].

Свидетельства о регистрации программ для ЭВМ. В рамках диссертационной работы получено пять свидетельств о регистрации программ для ЭВМ:

- № 2011614454 от 06.06.2011 г. «Программное средство для удаления ошибок из набора чтений нуклеотидной последовательности»;
- №2012616774 от 27.07.2012 г. «Программное средство для сборки квазиконтигов из парных чтений»;

- № 2013616471 от 09.07.2013 г. «Программное средство, реализующее алгоритм поиска перекрытий между квазиконтигами»;
- № 2013619155 от 26.09.2013 г. «Программное средство, реализующее запуск этапов сборки генома через графический интерфейс пользователя»;
- № 2013660881 от 21.11.2013 г. «Программное средство, реализующее алгоритм упрощения графа перекрытий при сборке геномных последовательностей»;

Участие в научно-исследовательских работах. Некоторые результаты диссертации были получены при выполнении следующих научно-исследовательских работ: «Разработка методов сборки генома, сборки транскриптома и динамического анализа протеома» (Государственный контракт № 14.B37.21.0562, 2012–2013 гг.) и «Разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям» (Государственный контракт № 16.740.11.0495, 2011–2013 гг.). Автор является победителем конкурса грантов для студентов вузов, расположенных на территории Санкт-Петербурга, аспирантов вузов, отраслевых и академических институтов, расположенных на территории Санкт-Петербурга 2013 и 2014 гг., темы проектов: «Разработка алгоритма упрощения графа перекрытий при сборке геномных последовательностей» (2013 г.) и «Сборка контигов геномных последовательностей на основе принципа максимального правдоподобия» (2014 г.).

Объем и структура диссертации. Диссертация состоит из введения, четырех глав, заключения и двух приложений. Объем диссертации составляет 171 страницу, с 37 рисунками, 16 таблицами и тремя листингами. Список источников содержит 119 наименований.

В **первой главе** приводится обзор предметной области, включающий в себя определения базовых понятий, существующих методов секвенирования ДНК, метагеномного секвенирования, подходов к сборке

геномных последовательностей *de novo* и анализу данных метагеномного секвенирования. Проводится анализ возможности использования существующих программ для обучения биоинформатике, приводятся таблицы сравнения. На основе результатов обзора формулируются задачи, решаемые в диссертации.

Во **второй главе** приводится описание предлагаемого метода сборки генома *de novo*. Метод основан на совместном применении графов де Брейна и графов перекрытий и позволяет использовать преимущества обеих структур данных. Предложенный метод использует парные чтения, полученные с устройств-секвенаторов, и состоит из трех этапов:

- исправление ошибок секвенирования (основано на анализе *k*-мерного спектра);
- сборка квазиконтигов из парных чтений (выполняется на графе де Брейна);
- сборка контигов из квазиконтигов (выполняется на графе перекрытий).

Для подтверждения целесообразности использования разных графов на разных этапах сборки, приводятся полученные автором соотношения требуемых ресурсов на хранение данных от параметров исходных данных для двух графов.

В первой главе также описывается экспериментальное исследование, которое было проведено для сравнения разработанной программы с существующими. Для этого было использовано три набора данных, свободно доступных в сети Интернет. Экспериментальное исследование показало, что предлагаемый сборщик *ITMO Genome Assembler* отработал эффективнее по памяти, чем сборщики *SPAdes, Velvet, MaSuRCA, Newbler* и *Minia*, однако использовал больше памяти, чем сборщик *SparseAssembler*. При этом по числу найденных генов в сборке предлагаемый подход незначительно уступает *SPAdes* (разница от **0.1%** до **0.6%**), но показал лучшие результаты по сравнению со всеми остальными сборщиками (разница может достигать **81.7%**). При этом сборщики *SPAdes, Velvet,*

MaSuRCA и *Newbler* не смогли произвести сборку среднего по размеру генома при ограничении в памяти в 16 Гб в отличие от предлагаемого решения, которое уложилось в 4 Гб памяти.

В **третьей главе** приводится описание разработанного метода сравнительного анализа метагеномов *MetaFast*, основанного на анализе компонент связности в графе де Брейна. Предлагаемый метод частично основан на описанном в главе 2 методе *de novo* сборки генома, однако, в отличие от него, использует непарные чтения. Метод состоит из четырех этапов:

- выполнение «упрощенной» сборки для каждого метагенома отдельно (выполняется на графе де Брейна);
- построение общего графа де Брейна для всех метагеномов и выделение компонент связности в нем (каждая компонента далее используется как единичный признак, определение признака дано в главе 1);
- вычисление характеристического вектора для каждого метагенома, где элемент вектора – число *k*-меров из компоненты связности в данном метагеноме;
- вычисление попарного расстояния между метагеномами на основе полученных характеристических векторов (с использованием индекса Брея-Кертиса).

В третьей главе также приводятся результаты экспериментального исследования, целью которого было сравнение предлагаемого подхода с существующими по качеству получаемых результатов и необходимым вычислительным ресурсам. Указанные исследования проводились на четырех наборах данных, свободно доступных в сети Интернет.

Результаты экспериментов подтвердили высокую точность получаемых матриц расстояния предлагаемого решения (корреляция Спирмена $r = 0.96$ с истинной матрицей расстояния на симулированных данных и $r = 0.81–0.91$ на реальных наборах). Также была показана высокая производительность предложенного метода по сравнению с

существующими подходами и преимущество в независимости от базы референсных геномов по сравнению с традиционными подходами.

В **четвертой главе** приводится описание внедрения результатов диссертационной работы в образовательную и исследовательскую деятельность. Результаты диссертации внедрены в учебный процесс в Санкт-Петербургском политехническом университете Петра Великого и в Университете ИТМО, а также использовались в Казанском (Приволжском) Федеральном Университете при выполнении научно-исследовательских работ.

В **заключении** сформулированы результаты, полученные в диссертационной работе.

ГЛАВА 1. ДЕЗОКСИРИБОНУКЛЕИНОВАЯ КИСЛОТА (ДНК), СЕКВЕНИРОВАНИЕ ДНК И АНАЛИЗ ДАННЫХ СЕКВЕНИРОВАНИЯ

В настоящей главе приводятся результаты обзора предметной области, включающие в себя определения базовых понятий, существующих методов секвенирования ДНК, метагеномного секвенирования, подходов к сборке геномных последовательностей *de novo* и анализу данных метагеномного секвенирования. Проводятся результаты анализа существующих программ сборки генома и анализа метагеномов по возможности их использования для обучения биоинформатике. На основе результатов обзора формулируются задачи, решаемые в диссертации.

Некоторые части обзора, представленные в данной главе, также были опубликованы в работах автора [101, 110, 111], отчетах по государственным контрактам [1, 97–99] и в магистерской диссертации автора [2].

1.1. ДНК

Дезоксирибонуклеиновая кислота (*ДНК*) – 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.

К середине двадцатого века было установлено, что вся наследственная информация об организме находится в молекулах ДНК. В 1953 году Д. Уотсоном и Ф. Криком была предложена модель строения ДНК в виде двухцепочечной спирали [3], что позже было подтверждено в других экспериментах.

Таким образом, молекула ДНК состоит из двух цепочек, каждая из которых является последовательностью нуклеотидов: аденина (кодируется символом A), тимина (кодируется символом T), гуанина (G) и

цитозина (C). Нуклеотиды в парах A-T и G-C комплементарны друг другу, и в двух цепочках ДНК на одинаковых позициях находятся нуклеотиды, комплементарные друг другу. Два конца любой цепочки ДНК различаются с химической точки зрения, один из них называется *5′*, другой – *3′*. Поэтому у любой цепочки можно определить направление: обычно прямым считается направление от *5′* к *3′*, что обусловлено тем, что большинство биологических процессов по обработке последовательности ДНК протекают именно в этом направлении. Цепочки в одной спирали противонаправлены друг другу.

Наследственная информация расположена не в одной молекуле ДНК, а, вообще говоря, в нескольких, обычно расположенных в хромосомах. Кроме того, у некоторых организмов, к которым относятся большинство млекопитающих, набор хромосом удвоен – такие организмы называются *диплоидными*. В каждой паре находится по одной хромосоме от каждого из родителей, причем эти хромосомы, за исключением пары половых хромосом, отличаются в относительно небольшом числе нуклеотидов. Отличие в одном нуклеотиде называется *однонуклеотидным полиморфизмом* (*single nucleotide polymorphism, SNP*). Также существуют организмы с утроенным, учетверенным и т. д. наборами хромосом.

В 1958 году Ф. Криком было сформулировано правило, названное впоследствии «Центральной догмой молекулярной биологии» [4]. Она постулирует правило передачи информации от ДНК через РНК (*рибонуклеиновую кислоту*) к белку. Процесс передачи информации от ДНК к РНК называется *транскрипцией*, а процесс построения белка по РНК называется *трансляцией*. В дальнейшем были открыты и иные переходы информации (обратная транскрипция, репликация). Таким образом, белки, создаваемые клеткой и осуществляющие всю работу клетки, напрямую связаны с информацией, закодированной в ДНК.

РНК, как одна из первых форм для хранения биологической информации, по структуре схожа с ДНК. Для некоторых вирусов РНК и по

сей день выполняет эту роль (так как в таких клетках отсутствует ДНК). Однако функции РНК в современных клетках намного шире, чем хранение наследственной информации. Существует несколько типов РНК: *матричные РНК* (*мРНК*), *транспортные* (*тРНК*), *рибосомальные* (*рРНК*) и другие. Они участвуют при передаче генетической информации от ДНК к белку (как временный носитель информации), в сплайсинге эукариотических матричных РНК, в процессе трансляции, осуществляемым рибосомами, а также сами являются структурной и каталитической основой рибосом.

В процессе изучения работы клетки ученые поняли, что без непосредственного анализа наследственной информации, хранящейся в ДНК, разобраться в этой сложной системе невозможно.

1.1.1. Секвенирование ДНК

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

Существующие технологии не позволяют прочитать всю цепочку ДНК целиком (как и слишком большие ее части). Для секвенирования применяются другие технологии.

Современными методами секвенирования ДНК являются *методы нового поколения* (*next-generation sequencing*, *NGS*) [47]. При секвенировании этими методами ДНК случайным образом дробится на мелкие участки, длиной до 500 нуклеотидов, каждый из которых затем считывается.

Для того чтобы было возможным восстановить исходную последовательность ДНК, обеспечивается многократное покрытие генома чтениями. Пусть имеется набор из N чтений длины L, и пусть геном имеет длину G. Тогда говорят, что данный набор обеспечивает покрытие

генома c, определяемое формулой $c = \frac{N \cdot L}{G}$. Данное определение было введено в 1988 году в работе [8].

Для еще большего удешевления процесса считывания используются так называемые парные чтения. Получение таких чтений происходит следующим образом. Секвенатором выделяется фрагмент ДНК, расположенный в случайном месте, из которого затем считываются префикс и суффикс. Результатом работы секвенатора в случае использования парных чтений являются пары последовательностей, про которые известно, на каком расстоянии они располагались в исходной последовательности ДНК.

Современные технологии (такие как, например, *Illumina GAIIx* [90]), позволяют получить длину фрагмента около 500 нуклеотидов, а длину считанных префиксов и суффиксов до 150 нуклеотидов.

Полученные данные собираются в единую последовательность при помощи специального программного обеспечения – *сборщика генома*. Из-за наличия повторов в геномной последовательности однозначно восстановить саму последовательность практически невозможно, поэтому требуется восстановить как можно более длинные непрерывные фрагменты геномной последовательности.

1.1.2. Существующие методы секвенирования

В данном разделе описаны несколько существующих методов секвенирования. В основе любого метода лежит технология, позволяющая физически узнавать, в какой последовательности находятся нуклеотиды в некоторой части молекулы ДНК. Заметим, что при физическом «прочтении» могут возникать ошибки – изменение одного нуклеотида на другой, вставка или удаление небольшого числа нуклеотидов. При этом для каждого прочитанного нуклеотида известна числовая характеристика, обычно называемая *качеством прочтения*, – величина, на основе которой

можно вычислить вероятность того, что нуклеотид был прочитан неправильно.

1.1.2.1. Метод обрыва цепи

Ф. Сенгером к 1977 году был разработан один из первых методов секвенирования – метод обрыва цепи [5], также называемый методом Сенгера и «методом терминаторов». Образец цепочки ДНК, который подвергается секвенированию, делится на четыре группы. Каждая группа ответственна за свой нуклеотид. После этого в каждой группе происходит *de novo* синтез молекулы нуклеиновой кислоты из, например, радиоактивно помеченных нуклеотидов, причем такой синтез может завершиться только в нуклеотиде, который соответствует данной группе. После этой операции в каждой группе получаются молекулы разных весов (с разной длиной), в зависимости от того, в каких позициях в исходной цепочке находится нуклеотид, соответствующий выбранной группе. Далее в каждой группе происходит разделение молекул по весам с точностью до одного нуклеотида (с помощью электрофореза), что, в свою очередь, позволяет узнать, в каких позициях этот нуклеотид встречается в исходной цепочке.

В настоящее время этот метод все ещё используется, позволяя напрямую секвенировать последовательности из 800–1000 нуклеотидов.

1.1.2.2. Метод дробовика

Для того чтобы секвенировать большие геномы (больше тысячи нуклеотидов), почти одновременно было предложено два метода: метод обхода хромосомы (*chromosome walking*) [6] и метод дробовика (*shotgun sequencing*) [7].

Метод обхода хромосомы состоит в том, чтобы сначала секвенировать первую тысячу нуклеотидов. После этого, зная последние нуклеотиды прочитанного фрагмента и зацепившись за его конец, можно секвенировать очередную тысячу нуклеотидов и продолжить такую

процедуру. Однако со временем было показано, что такой метод оказывается неэффективным для больших геномов.

Более удобным оказался *метод дробовика*, при котором сначала происходит разделение случайным образом молекул ДНК на небольшие фрагменты, которые затем секвенируются с помощью других методов (например, метода обрыва цепи). За счет большого числа итераций описанной процедуры, получается большое число последовательностей, которые покрывают весь геном. Благодаря тому, что такие части перекрываются друг с другом, можно попытаться восстановить всю исходную последовательность.

1.1.2.3. Высокопроизводительные методы секвенирования

Со временем развитие вычислительной техники и появление новых технологий сделали секвенирование более дешевым, а методы – более производительными.

Например, перспективной является следующая достаточно дешевая и эффективная технология: сначала вычленяется случайно расположенный в геноме фрагмент, а затем происходит считывание двух последовательностей с его концов. Эти последовательности называются *парными чтениями*. Процесс повторяется такое число раз, чтобы обеспечить достаточно большое покрытие генома чтениями. Заметим, что размеры фрагментов могут варьироваться от средних (около 200 нуклеотидов) до достаточно больших (10 000 нуклеотидов), а размеры чтений в целом уменьшаются при увеличении размера фрагмента. Например, в секвенаторе *Genome Analyser IIx* компании *Illumina* [90], использующем такую технологию, размер фрагмента составляет примерно 500 нуклеотидов, а размер чтений – около 100.

1.1.2.4. Характеристики секвенаторов

В работах [10, 11] было проведено сравнение существующих секвенаторов. Некоторые характеристики таких сравнений приведены в таблице 1.

Таблица 1 – Технические характеристики секвенаторов нового поколения

Платформа	*Illumina GAIIx*	*Illumina HiSeq* 2000	*Illumina MiSeq* version 2	*Ion Torrent PGM* «318» chip	*PacBio RS*
Стоимость платформы	$250 000	$690 000	$128 000	$80 000	$695 000
Прочитывают баз за проход, млн нукл.	96 000	600 000	7 500	1 500	0.03
Стоимость прочтения за млн. нукл.	$190	$41	$140	$600	$2000
Длительность одного запуска	14 дней	11 дней	39 часов	7 часов	2 часа
Процент ошибок	~0.1%	~0.1%	~0.1%	~1%	~13%[1]
Длина чтения, нуклеотидов	2 x 150	2 x 100	2 x 250	400	3 000
Парные чтения	Да	Да	Да	Нет	Нет

[1] Может быть уменьшена до ~1% для консенсуса из трех независимых чтений

Стоит отметить, что общая стоимость секвенирования с каждым годом падает. Это объясняется появлением новых эффективных технологий, которые могут удешевлять некоторые этапы процесса секвенирования, а также уменьшением цены на реагенты, необходимые для секвенирования.

Ученые из международного института исследования генома человека (*National Human Genome Research Institute, NHGRI*) уже многие годы отслеживают изменения цен на услуги секвенирования [88, 89]. При этом фиксируется минимальная цена за каждый квартал года. Полученные данные отображают в виде графика, который показан на рисунке 1 (рисунок получен с официального сайта организации [88]). Хотя такой график не показывает всех изменений на рынке услуг секвенирования, общая тенденция в его развитии отчетливо видна.

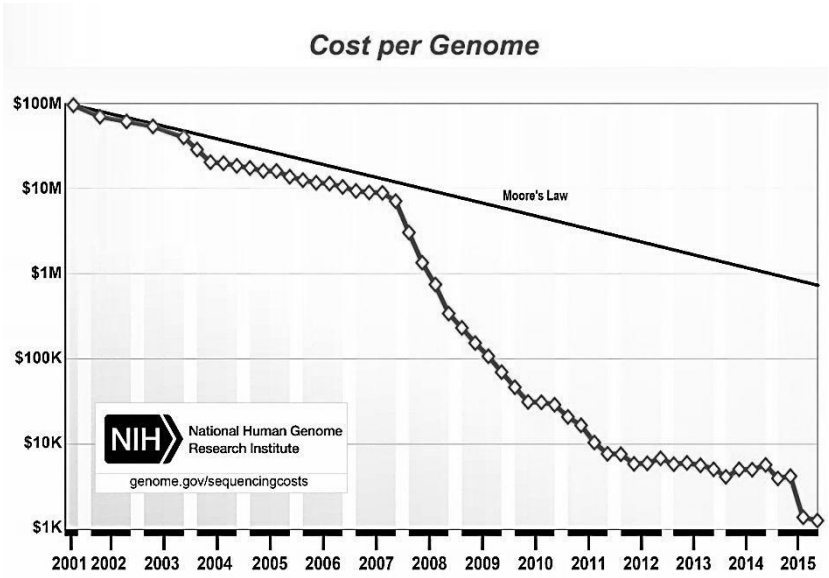


Рисунок 1 – Снижение стоимости секвенирования генома человека. График взят с сайта `http://www.genome.gov/sequencingcosts`.

1.1.3. Метагеномное секвенирование

Изначально секвенирование ДНК применялось для определения последовательности ДНК отдельного организма. Со временем распространение секвенаторов и удешевление стоимости секвенирования сделали возможным использование их не только для этой цели. Секвенирование стало применяться и для анализа сред без изолирования конкретных микроорганизмов [12]. Это привело к рождению *метагеномики* – направлению биологии, посвященному получению и исследованию геномных последовательностей непосредственно из образцов среды. Совокупность таких последовательностей получила название *метагеном*.

При секвенировании метагеномов, все ДНК, которое находится в пробирке, используется для секвенирования. Это позволяет анализировать все сообщество организмов, находящихся в образце.

Можно выделить следующие отличия метагеномного секвенирования от обычного геномного:

- При секвенировании чтения «приходят» от разных организмов.
- Покрытие разных участков ДНК чтениями сильно варьируется, в том числе организмы в сообществе могут быть покрыты чтениями только частично.

Метагеномное секвенирование открыло возможность изучать сообщества организмов из разных сред обитания. Самыми распространенными средами считаются почва, водные ресурсы, кишечник человека и другие.

1.2. Сборка генома

Сборка геномной последовательности (*сборка генома*) – процесс получения больших фрагментов генома, выполняется с использованием компьютеров.

Задача разработки методов сборки геномных последовательностей является, в определенном смысле, центральной среди всех задач биоинформатики. Это объясняется тем, что без ее решения нельзя приступить к детальному изучению генома живого существа и его анализу с применением других алгоритмов биоинформатики.

Решением задачи сборки генома может являться набор контигов, набор скэффолдов либо завершённая референсная последовательность. *Контигом* называется непрерывная (без пропусков) последовательность нуклеотидов (часть исходного генома), которую обычно не дается расширить с использованием имеющихся данных. *Скэффолдом* называется набор контигов с заданным порядком их следования и оценками на расстояния между парами соседних контигов. *Референсная (эталонная) последовательность* – законченный собранный геном высокого качества (один контиг или один скэффолд).

В рамках данной работы будет рассматриваться так называемая задача *de novo сборки генома* – сборки генома живого существа, для которого геном еще не известен.

Сложность задачи сборки геномной последовательности обусловлена следующими факторами:

- большой объем входных данных;
- сложность структуры генома – наличие в нем повторов и полиморфизмов;
- наличие ошибок в исходных данных, полученных с устройств-секвенаторов.

Для того чтобы уменьшить влияние некоторых факторов, вызывающих ошибки, геномную последовательность покрывают чтениями

несколько десятков раз. При этом покрытие геномной последовательности чтениям оказывается достаточно равномерным – все позиции, независимо от их расположения в последовательности, покрыты чтениями примерно одинаковое число раз.

Процесс сборки генома обычно разбит на несколько этапов:

- исправление ошибок в исходных данных;
- восстановление фрагментов генома по парным чтениям, такие фрагменты называются *квазиконтигами*;
- сборка расширенных фрагментов (*контигов*) из квазиконтигов;
- построение *скэффолдов* – упорядоченных контигов с оценкой расстояния между ними.

1.2.1. Постановка задачи восстановления геномной последовательности

Таким образом, в процессе получения исходной последовательности ДНК организма возникает следующая задача. Имеются небольшие фрагменты геномной последовательности (иногда – несколько разных наборов с разными характеристиками). Необходимо восстановить исходную последовательность ДНК (или как можно большие её части) используя имеющиеся данные.

Для корректного решения задачи необходимо сформулировать математическую постановку задачи.

1.2.2. Задача о наименьшей общей надстроке

Задачу о наименьшей общей надстроке можно рассматривать как основополагающую задачу в процессе сборки генома.

Задача состоит в следующем. Дан набор строк $s_1, \ldots, s_m$. Необходимо найти наименьшую строку s, такую, что все s_i будут входить в строку s как подстроки. Иными словами, для каждого i ($i \in [1, m]$) строку s можно представить в виде $u_i s_i v_i$ для некоторых строк u_i и v_i.

Применительно к задаче сборки генома строки s_i обозначают непрерывные фрагменты генома, а строка s – исходный геном. Зная непрерывные фрагменты генома, необходимо восстановить исходный геном – найти строку s.

Задача в такой формулировке является *NP*-трудной [14, 15].

Доказательство *NP*-трудности задачи состоит в сведении задачи о гамильтоновом пути, которая является *NP*-полной, к рассматриваемой задаче. Такое сведение приведено в работе [14].

Из-за *NP*-трудности задачи, эффективного точного решения задачи, скорее всего, не существует. Однако ввиду важности задачи в областях секвенирования ДНК [16–18] и сжатии данных [19], эффективные приближенные алгоритмы ее решения были и остаются необходимыми. При этом для оценки оптимальности приближенных алгоритмов производится сравнение длины строки, являющейся результатом работы приближенного алгоритма, с длиной строки, которая получается оптимальным алгоритмом. Если удается доказать, что длины строк отличаются не больше, чем в константу раз, то говорят, что приближенный алгоритм является *линейной аппроксимацией* оптимального решения. При этом константу называют *коэффициентом аппроксимации*.

За время существования задачи было предложено множество приближенных алгоритмов. Для многих из них удалось найти коэффициент аппроксимации и доказать его. Сравнение таких алгоритмов выполнено в таблице 2.

Помимо этих алгоритмов давно известен жадный алгоритм решения задачи о наименьшей общей надстроке. Он состоит в повторе процедуры объединения двух различных строк с наибольшим перекрытием в одну. Если пар различных строк с максимальным перекрытием несколько, выбирается любая из них. Далее мы будем называть такой алгоритм *GREEDY*.

Однако, найти и доказать коэффициент аппроксимации этого алгоритма – сложная задача, которая не решена и по сей день. Известна гипотеза, что коэффициент аппроксимации равен двум [19, 30, 31]. А. Блум и его соавторы в 1991 году смогли показать, что жадный алгоритм аппроксимирует решение с коэффициентом четыре [20], Х. Каплан с соавторами в 2005 году улучшили этот коэффициент до $3\frac{1}{2}$ [32].

Таблица 2 – Сравнение приближенных алгоритмов с доказанным коэффициентом аппроксимации

Авторы алгоритма	Год	Коэффициент
Li [17]	1990	$O(\log(n))$
Blum, Jiang, Li, Tromp, Yannakakis [20]	1991	3
Teng, Yao [21]	1993	$2\frac{8}{9}$
Czumaj, Gasieniec, Piotr´w, Rytter [22]	1994	$2\frac{5}{6}$
Kosaraju, Park, Stein [23]	1994	$2\frac{50}{63}$
Armen, Stein [24]	1995	$2\frac{3}{4}$
Armen, Stein [25]	1996	$2\frac{2}{3}$
Breslauer, Jiang, Jiang [26]	1997	$2\frac{25}{42}$
Sweedyk [27]	1999	$2\frac{1}{2}$
Kaplan, Lewenstein, Shafrir, Sviridenko [28]	2005	$2\frac{1}{2}$
Paluch, Elbassioni, van Zuylen [29]	2012	$2\frac{1}{2}$

Рассмотрим задачу поиска наименьшей общей надстроки подробнее. Приведем разные интерпретации задачи, а также некоторые алгоритмы ее решения.

Дадим определения. Пусть имеется множество строк $S = \{s_1, \ldots, s_m\}$. Не уменьшая общности, можем считать, что множество S является «подстроково-независимым», что значит, что нет строки $s_i \in S$, которая является подстрокой другой строки $s_j \in S$. Иными словами, если есть $s_i \in S$, которая является подстрокой строки $s_j \in S$, то можно просто выкинуть s_i из множества S, и оптимальное решение от этого не поменяется. Это следует из того, что оптимальное решение задачи будет содержать строку s_j в качестве подстроки, и, следовательно, оно также будет содержать строку s_i в качестве подстроки, так как s_i – подстрока s_j.

Необходимо найти наименьшую строку s, такую, что для каждого i ($i \in [1, m]$) строку s можно представить в виде $u_i s_i v_i$ для некоторых строк u_i и v_i.

Рассмотрим пример. Предположим требуется найти наименьшую общую надстроку всех слов из следующей строки «*alf ate half lethal alpha alfalfa*». Заметим, что слово «*alf*» является подстрокой слов «*half*» и «*alfalfa*», поэтому можно его убрать. Тогда множество S = {*ate, half, lethal, alpha, alfalfa*}.

Тривиальное решение для нахождения надстроки (не обязательно минимальной) – простое сцепление всех слов подряд. Для множества S получим надстроку «*atehalflethalalphaalfalfa*» длины 25.

Оптимальное же решение – «*lethalphalfalfate*» длины 17, которое имеет выигрыш в восемь символов по отношению к предыдущему результату.

Рассмотрим работу алгоритма *GREEDY* для данного множества строк. Он будет действовать следующим образом. Сначала он объединит строки «*lethal*», «*half*» и «*alfalfa*» с максимальным перекрытием (три символа) в одну строку «*lethalfalfa*» (не важно в каком порядке он будет их

объединять). Множество S после этого шага будет следующим S = = {*lethalfalfa*, *alpha*, *ate*}. Далее перекрытие в один символ имеет каждая строка с каждой другой. Если алгоритм *GREEDY* сначала объединит первую строку со второй, или вторую с третьей, то в следующем шаге оставшиеся строки будут перекрываться на один символ и алгоритм получит в итоге строку «*lethalfalfalphate*» длины 17. Однако если он сначала объединит первую и третью строки, то получившаяся строка «*lethalfalfate*» не будет перекрываться с оставшейся строкой «*alpha*», и в конечном итоге получится строка длиной в 18 символов.

Из данного примера следует, что алгоритм *GREEDY* может достигать оптимального результата на некоторых примерах, если будет оптимально выбирать пару строк для объединения при нескольких вариантах.

Теперь рассмотрим работу алгоритма *GREEDY* для множества S = $\{c(ab)^k, (ba)^k, (ab)^k c\}$ для некоторого $k > 0$. Можно заметить, что сначала алгоритм объединит первую и третью строки, получив «$c(ab)^k c$». После этого получившаяся строка не будет перекрываться с оставшейся строкой «$(ba)^k$» и придется просто объединять эти две строки, получив строку длины $4k+2$, в то время как оптимальная строка «$c(ab)^{k+1}c$» имеет длину $2k+4$. При неограниченном увеличении параметра k данного примера соотношение результата алгоритма *GREEDY* к оптимальной строке будет стремиться к двум, что говорит о том, что при любом выборе пары строк из нескольких вариантов с максимальным перекрытием, алгоритм *GREEDY* не сможет гарантировать коэффициент аппроксимации меньший двух.

Приведем определение перекрытия строк.

Пусть имеется две строки s и t (не обязательно различных). Обозначим через ov(s, t) наибольшую строку v такую, что $s = uv$ и $t = vw$ для некоторых непустых строк u и w (см. рисунок 2). Через pref(s, t) обозначим строку u. Тогда наименьшая общая надстрока строк s и t будет uvw = pref(s, t)t.

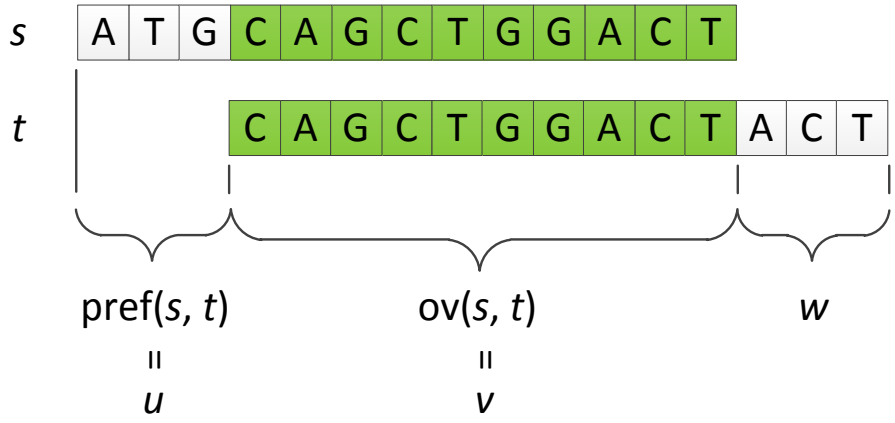


Рисунок 2 – Перекрытие строк

Обобщая сказанное, надстроку s для множества строк $S = \{s_1, \ldots, s_m\}$ можно получить, зная порядок следования строк s_i в надстроке s. Таким образом, надстрока $s = \text{pref}(s_{i_1}, s_{i_2})\text{pref}(s_{i_2}, s_{i_3}) \ldots \text{pref}(s_{i_{m-1}}, s_{i_m})s_{i_m}$ для некоторого заданного порядка строк $s_{i_1}, s_{i_2}, \ldots, s_{i_m}$. Это рассуждение верно для подстроково-независимого множества строк S, так как в этом случае порядок строк s_i однозначно определяет наименьшую надстроку s для выбранного порядка.

1.2.2.1. Графовое представление задачи о наименьшей общей надстроке

Имеется «подстроково-независимое» множество строк $S = \{s_1, \ldots, s_m\}$.

Графом перекрытий называется взвешенный ориентированный граф, каждой вершине которого сопоставлена строка s_i, с ребрами между каждыми двумя вершинами. Вес ребра между вершинами a и b – длина перекрытия двух строк s_a и s_b, соответствующих вершинам. На рисунке 3 изображен граф перекрытий для $S = \{ate, half, lethal, alpha, alfalfa\}$.

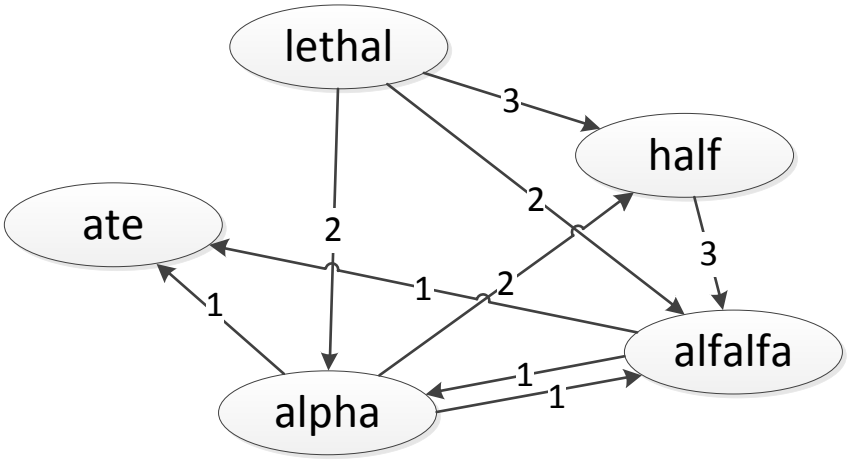


Рисунок 3 – Граф перекрытий (ребра с весом 0 не показаны)

Отметим, что любой путь в этом графе соответствует строке, составленной из строк, соответствующих вершинам в пути. Такая строка может быть получена путем объединения строк, соответствующих вершинам в пути, в порядке, в котором мы проходим эти вершины. При этом при объединении двух строк учитывается тот факт, что строки могут перекрываться. Например, для пути «*lethal → half → ate*» получится строка «*lethalfate*».

Определим вес пути как сумму весов всех ребер, по которым проходит путь. Неформально вес пути можно определить как выигрыш длины строки, соответствующей пути, по отношению к сумме длин строк, соответствующих вершинам в пути. Тогда оптимальной надстроке будет соответствовать путь с максимальным весом, который проходит по каждой вершине ровно один раз (гамильтонов путь с максимальным весом). Задача о гамильтоновом пути с максимальным весом является задачей коммивояжера (*travelling salesman problem, TSP*), которая является *NP*-трудной.

Графом расстояний называется взвешенный ориентированный граф, каждой вершине которого сопоставлена строка s_i, с ребрами между каждыми двумя вершинами. Вес ребра из вершины a в вершину b равен $|\mathrm{pref}(s_a, s_b)|$, где s_a и s_b – соответствующие вершинам строки. Если строки

s_a и s_b не перекрываются, то $|\text{pref}(s_a, s_b)| = |s_a|$. Граф расстояний изображен на рисунке 4 для S = {*ate*, *half*, *lethal*, *alpha*, *alfalfa*}.

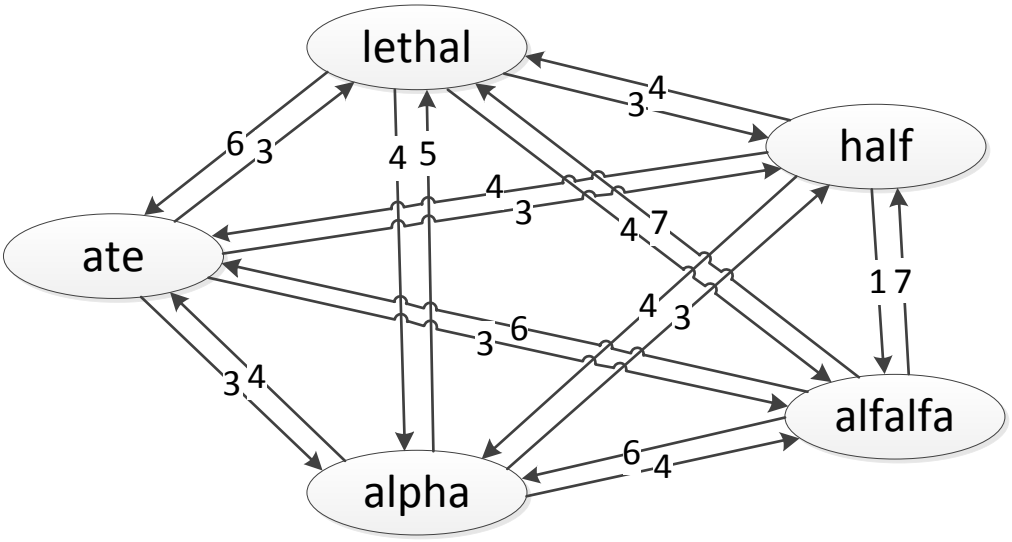


Рисунок 4 – Граф расстояний

Как и для графа перекрытий, пути в графе расстояний соответствует строка, получаемая объединением строк, соответствующих вершинам в пути, в порядке обхода вершин.

Определим вес пути в графе расстояний как сумму весов всех ребер и длины строки, соответствующей последней вершине в пути. Тогда оптимальной надстроке будет соответствовать минимальный по весу гамильтонов путь.

1.2.2.2. Точные алгоритмы решения

Как было отмечено выше, оптимальную надстроку s для множества строк S = {s_1, …, s_m} можно получить, зная порядок следования строк s_i в надстроке s. Основываясь на этом утверждении, можно предложить алгоритм решения задачи о наименьшей общей надстроке – перебрать все возможные варианты следования строк в надстроке, для каждого варианта построить надстроку и выбрать минимальную из них. Однако такое решение весьма неэффективно – затраченное время на решение задачи будет $O(m!)$, что очень медленно – современные компьютеры и кластеры

не смогут решить рассматриваемую задачу за приемлемое время даже для пятидесяти строк.

Такое элементарное решение также применимо для графового представления задачи – задачи коммивояжера: переберем все возможные варианты следования вершин в пути, проверим каждый путь и после этого выберем оптимальный.

Это решение можно улучшить, используя динамическое программирования по подмножествам. Пусть S – подмножество вершин графа, в котором решается задача коммивояжера, $d[S][i]$ – наименьший/наибольший вес пути, который проходит через все вершины из S и заканчивается в вершине i. Инициировав начальными значениями массив d (путями, состоящими из одной вершины) и организовав пересчет его (для заданного S и i перебираем следующую вершину в пути и проверяем такой вариант), можно решить задачу коммивояжера за время $O(2^n n^2)$ используя $O(2^n n)$ памяти. Однако, это все еще очень медленно – для графа из ста вершин задачу не решить.

1.2.2.3. Приближенные алгоритмы решения

Наиболее распространенным приближенным алгоритмом является алгоритм *GREEDY*, о котором уже было сказано выше. Он состоит в выборе двух различных строк, перекрытие которых наибольшее, объединении их и повторе этой процедуры, пока не останется только одна строка. Если пар различных строк с максимальным перекрытием несколько, то выбирается любая из них.

Гипотеза о коэффициенте аппроксимации этого алгоритма [19, 30, 31], утверждает, что этот коэффициент равен двум, однако лучший коэффициент, который удалось доказать, равен $3\frac{1}{2}$ [32].

Рассмотрим другие приближенные алгоритмы.

Как известно, задача поиска минимального гамильтонова пути в графе сложна. Необходимо перейти к более простой задаче для получения

приближенного алгоритма решения. Например, вместо поиска минимального гамильтонова пути можно искать минимальный гамильтонов цикл, а потом разрезать его в некотором месте, получив путь. Однако это никак не уменьшает сложность задачи. Следующая идея – искать не один гамильтонов цикл, а разрешить несколько циклов. Получается задача разбиения графа на циклы так, что каждая вершина встречается ровно в одном цикле и суммарный вес ребер всех циклов минимален. Эта задача уже имеет полиномиальное решение, так как она может быть сведена к *задаче о назначениях*, которая имеет полиномиальное решение.

Алгоритм, основанный на предыдущей идее, будет действовать следующим образом:

а) разбиение графа на циклы с минимальным суммарным весом;
б) разрезание всех циклов в произвольном месте, получая из них строки;
в) склеивание всех полученных строк в одну (порядок склеивания выбирается произвольно, строки склеиваются без сжатия по общим префиксам/суффиксам).

Такой алгоритм является приближенным алгоритмом для решения задачи о наименьшей общей надстроке. В работе [20] доказывается, что его коэффициент приближения не превосходит четырех.

Данный алгоритм можно улучшить. Для этого необходимо, во-первых, улучшить этап разрезания циклов: рассмотрим некоторый цикл; заметим, что он состоит из ребер с разным весом; выберем ребро с максимальным весом и исключим его – получим разрезание цикла. Во-вторых, заметим, что последний этап предыдущего алгоритма – склеивание строк без их сжатия – может быть улучшен путем объединения строк со сжатием по их перекрытиям. Получившийся улучшенный алгоритм имеет коэффициент аппроксимации, равный трем, что доказывается в работе [20].

1.2.2.4. Недостатки сборки генома через поиск наименьшей общей надстроки

Можно выделить следующие недостатки такого подхода:

а) не учитываются возможные ошибки в строках;

б) не учитывается отсутствие информации об ориентации каждой строки;

в) не учитывается степень покрытия частей генома чтениями;

г) не учитывается сложная структура генома.

Первый недостаток можно устранить путем поиска перекрытий между строками с учетом возможных ошибок. При оптимальном выборе параметров поиска можно полностью избавиться от этого недостатка, однако, скорее всего, исправить все ошибки не удастся, и наименьшая строка будет сильно увеличена в размерах из-за ошибок.

Проблему с незнанием ориентации можно решить с помощью добавления к множеству, как исходных строк, так и обратно-комплементарных к каждой из них. При этом в идеале от такого решения при поиске наименьшей надстроки получится две строки, которые будут обратно-комплементарны друг к другу, но в реальности из-за перекрытий строк с разным направлением, все может быть объединено в одну строку так, что разделить ее на две будет непросто.

Из-за того, что не учитывается степень покрытия генома чтениями, может возникнуть ситуация, при которой некоторые участки генома исходно не покрыты чтениями, и должно быть несколько наименьших надстрок в решении – по одной для каждой покрытой части. Однако из-за склеивания всех непересекающихся строк в одну в конце процесса поиска наименьшей надстроки или из-за странных перекрытий строк со строками из другой части, в результате может получиться одна строка, которую будет сложно интерпретировать.

И хотя от всех описанных выше недостатков можно с разной степенью успешности избавиться, главный недостаток состоит в том, что

задача о наименьшей общей надстроке не учитывает нетривиальную структуру генома. Это отражается, в первую очередь, в плохой работе с повторяющимися частями генома, которые часто встречаются в реальных геномах.

Рассмотрим один из таких примеров – пусть в исходном геноме есть две одинаковые части (рисунок 5).

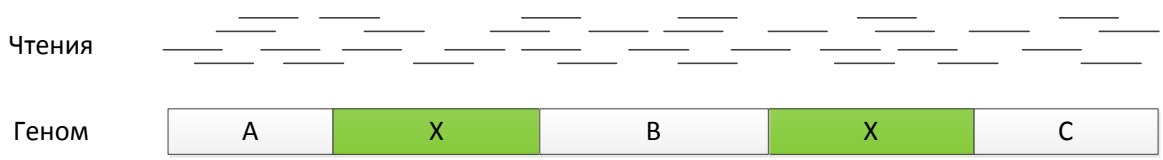


Рисунок 5 – Исходный геном и его покрытие чтениями

В этом случае алгоритм построения наименьшей общей надстроки из чтений может отнести чтения из второй повторяющейся части «X» к первой части. Возможные варианты результата алгоритма изображены на рисунке 6.

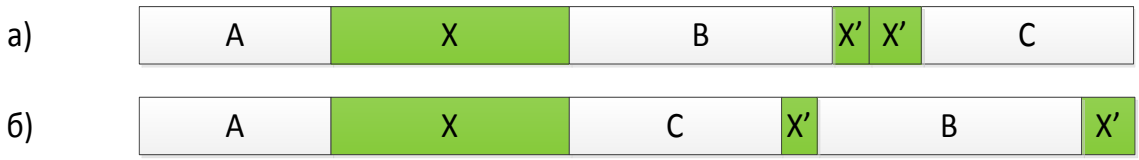


Рисунок 6 – Возможные наименьшие общие надстроки

1.2.3. Сборка генома на основе данных секвенирования с помощью гибридизации

В этом пункте рассматривается подход к секвенированию ДНК через гибридизацию и задача восстановления исходной ДНК, используя данные такого секвенирования.

Пусть имеется молекула ДНК D и натуральное число l. Процесс секвенирования молекулы ДНК через гибридизацию заключается в построении множества строк $S = \{s_1, \dots, s_n\} \subseteq \Sigma^l$, каждая из которых длины l, а также каждая строка встречается как подстрока в молекуле D.

Такое множество строк называется *l-спектром* молекулы ДНК, или просто *спектром*.

Отметим, что в идеальном случае (без ошибок) спектр молекулы содержит те и только те строки длины *l*, которые встречаются как подстроки в молекуле ДНК.

Задача построения молекулы ДНК по спектру *S* состоит в построении строки *w*, которая соответствует заданному *l*-спектру *S*. Будем считать, что *строка w соответствует спектру S*, если *w* содержит каждую строку из *S* как подстроку, и не содержит никакую другую подстроку длины *l*. При этом если *w* содержит каждую строку из *S* в точности один раз, то такое соответствие будем называть *простым*.

Данную задачу можно свести к задаче о наименьшей общей надстроке, рассматривая спектр как множество подстрок. В этом случае восстановить исходную ДНК можно будет с помощью методов построения наименьших общих надстрок, однако такая задача намного сложнее исходной. К тому же, при таком сведении мы отказываемся от дополнительной информации о том, что спектр содержит *все строки* длины *l*, которые присутствуют в молекуле, и только их.

Используя эту дополнительную информацию, можно предложить эффективный алгоритм, который описан далее.

Спектр молекулы ДНК можно представить в виде графа, который называется *l-спектральным графом*. Он состоит из вершин, которые соответствуют строкам длины *l*-1. Каждое ребро графа соответствует некоторой строке *e* длины *l* из спектра. Ребро проводится между двумя вершинами, первая из которых соответствует префиксу строки *e* длины *l*-1, вторая – суффиксу длины *l*-1. На каждом ребре ставиться метка с последней буквой ребра. Пример спектрального графа для 3-спектра *S*={*abc*, *bbc*, *bcc*, *bcd*, *ccd*, *cda*, *cdb*, *dbb*} изображен на рисунке 7.

Спектральный граф является частным случаем графа де Брейна степени *k*=*l*-1, определение которого будет дано в следующем разделе.

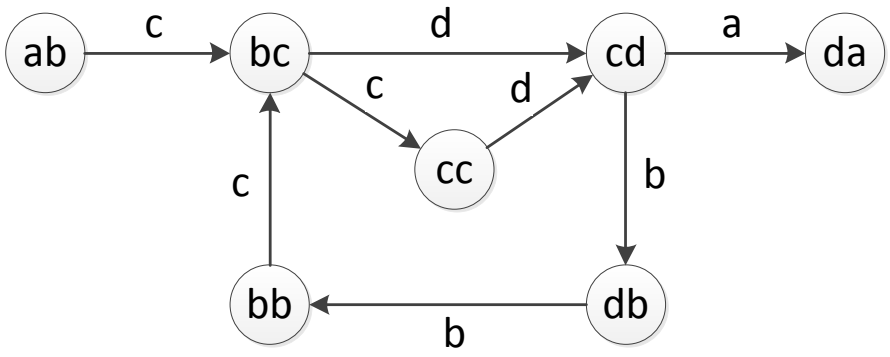


Рисунок 7 – Пример спектрального графа для S={*abc*, *bbc*, *bcc*, *bcd*, *ccd*, *cda*, *cdb*, *dbb*}

Любому пути в спектральном графе соответствует строка, получаемая объединением строки, соответствующей первой вершине, и всем меткам ребер, по которым проходит путь.

Для того чтобы по спектральному графу получить строку, которая включает в себя все строки спектра как подстроки и только их (и при этом только один раз), необходимо найти эйлеров путь в этом графе – путь, который проходит по каждому ребру ровно один раз.

Задача о поиске эйлерова пути является хорошо изученной задачей, и может быть решена за линейное относительно числа ребер время. При этом возможны случаи нескольких решений, что можно продемонстрировать на примере эйлерова пути для графа, изображенного на рисунке 7. Здесь имеется два пути: P_1=*abcdbbccda* и P_2=*abccdbbcda*.

Таким образом, задача восстановления молекулы ДНК через l-спектр молекулы может быть решена достаточно эффективно. Однако у такого решения имеются следующие недостатки:

- не учитываются возможные ошибки в спектре;
- восстанавливаются молекулы ДНК, в которых каждая строка из спектра встречается только один раз;
- возможность нескольких разных итоговых решений, которые сильно отличаются друг от друга;

- технические трудности: современные методы позволяют секвенировать молекулы ДНК на основе гибридизации только небольшой длины.

Из-за всех недостатков такой подход восстановления ДНК в настоящее время отдельно практически не используется. Однако в совокупности с другими методами он может быть весьма полезен.

1.2.4. Графовая постановка задачи о сборке генома

При постановке задачи сборки генома через поиск наименьшей общей надстроки возникают проблемы с учетом числа повторов, встречающихся в исходном геноме. Такие трудности могут быть частично решены при использовании графов. В графовой модели возможно несколько постановок задачи сборки генома.

Рассмотрим граф перекрытий. Как уже отмечалось в разделе 1.2.2.1, возможной формулировкой задачи сборки генома является поиск в этом графе гамильтонова пути минимального веса.

В статье [33] предлагается строить граф перекрытий, с размером перекрытия не меньше константы τ, а после этого упростить его путем удаления *транзитивных перекрытий*. Перекрытие $u \to w$ называется *транзитивным*, если существует вершина v, являющаяся подстрокой $\mathrm{pref}(u, w)w$, и существуют ребра $u \to v$ и $v \to w$.

Транзитивное ребро $u \to w$ удаляется, так как последовательность, соответствующую этому пути, можно получить через путь $u \to v \to w$. Для удаления транзитивных ребер в указанной выше статье предлагается алгоритм, время работы которого – $O(\sum_{v \in V} \mathrm{tr.deg}(w) \cdot \deg(v))$, где $\deg(v)$ – степень вершины в исходном графе, а $\mathrm{tr.deg}(v)$ – степень вершины в графе с удаленными транзитивными ребрами.

После удаления транзитивных ребер все вершины можно разбить на две группы: внутренние, имеющие входящую и исходящую степени равные единице, и узловые, которые не удовлетворяют этому требованию.

Будем называть *составными ребрами* пути между двумя узловыми вершинами, которые соединены последовательность внутренних вершин. После этого для каждого составного ребра определим вероятность того, что последовательность, образованная таким путем, встречается в геноме ровно один раз. Используя полученную вероятность для ребра, его можно отнести к одной из трех групп:

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

б) *Обязательные ребра* – такие ребра, которые встречаются в геноме **не меньше** одного раза.

в) *Необязательные ребра* – ребра, которые не подходят под предыдущие две категории.

Для того чтобы определить, сколько раз необходимо посетить некоторое ребро, необходимо решить задачу о потоке минимальной стоимости, удовлетворяющего ограничениям, в полученном графе. При этом для каждого ребра задаются границы потока по нему: для точных ребер границы потока равны $[1,1]$, для обязательных – $[1,+\infty)$ и $[0,+\infty)$ для необязательных. После решения задачи о потоке минимальной стоимости, все ребра, которые не участвуют в решении, удаляются из графа. Задача поиска потока может быть решена за полиномиальное время, следовательно, предлагаемый метод тоже работает за полиномиальное время.

Заметим, что результирующий граф может оказаться несвязным, даже если исходный геном был одной циклической хромосомой. Этот факт подтверждается теоремой о том, что поиск подходящего (удовлетворяющего ограничениям) обхода в таком графе является *NP*-трудной задачей (теорема доказана в работе [34]).

Со временем были предложены другие методы для задачи сборки генома с использованием графа де Брейна.

Графом де Брейна степени k над алфавитом Σ для множества строк $S=\{s_1, ..., s_n\}$ называется ориентированный граф, в котором вершинами являются все строки из Σ^k (состоит из всех подстрок длины k строк s_i), а ребра – строки e из Σ^{k+1}. При этом ребро e соединяет две вершины, соответствующие строкам $e[1..k]$ и $e[2..k+1]$.

При использовании графа де Брейна задачу сборки генома можно свести к задаче китайского почтальона (*Chinese Postman Problem*). В этой задаче требуется найти цикл, проходящий по всем ребрам хотя бы раз. Каждое ребро, которое присутствует в чтениях, должно присутствовать и в пути, соответствующему собранному геному. В статье [35] показывается, что данная задача может быть сведена к задаче о потоке минимальной стоимости и поиску эйлерова цикла, и решена за полиномиальное время. При этом показано, что для несвязного графа задача не имеет решения.

При этом описанное сведение не учитывает, что (k+1)-меры были получены из более длинных чтений. Для использования этой информации ставится задача поиска кратчайшего надпути в графе де Брейна [36]. Задача состоит в следующем. Пусть имеем множество чтений $R=\{r_1, ..., r_n\}$. Будем предполагать, что чтения не содержат ошибок. Построим граф де Брейна для некоторого k. Заметим, что каждому чтению r_i длиной не меньше k+1 в графе де Брейна соответствует путь p_i длины $|r_i| - k$. Не уменьшая общности, будем считать, что каждое из чтений не короче k+1. Тогда для множества чтений R можно построить множество путей $P=\{p_1, ..., p_n\}$. Задача поиска кратчайшего надпути состоит в том, чтобы найти в этом графе путь $\mathcal{P}$ наименьшей длины, проходящий через каждое ребро не меньше одного раза и содержащий все пути из P как подпути. В статье [34] доказывается, что описанная задача является *NP*-трудной, доказательство основано на сведении задачи о наименьшей общей подстроке к ней.

1.2.5. Методы учета двухцепочечной структуры ДНК

Особенностью сборки генома является то, что ДНК состоит из двух, обратно-комплементарных цепочек и для чтений неизвестно, из какой цепочки их получили. На первых этапах для учета двухцепочечной структуры в граф для каждой вершины добавлялась обратно-комплементарная ей. Поиск путей проходил одновременно для прямого пути и обратно-комплементарного.

Другим решением этой проблемы является использование двунаправленных графов. Граф G называется двунаправленным, если у ребра есть два варианта ориентации относительно каждого из его концов: положительная и отрицательная. Вершина и ребро называется положительно инцидентными, если ребро ориентированно положительно относительно вершины, и отрицательно инцидентными в противном случае. Путем в этом графе является последовательность вершин и ребер $v_1, e_1, \dots, v_{k-1}, e_{k-1}, v_k$, в которой для любой последовательной тройки e_i, v_i, e_{i+1} выполняется условие, что ребра e_i и e_{i+1} имеют разную ориентацию относительно вершины v_i. Так как указание вершин излишне, их обычно упускают.

Можно построить двунаправленный граф перекрытий. Пусть заданы два чтения r_1 и r_2 и соответствующие им вершины v_1 и v_2. Между вершинами v_1 и v_2 проводится ребро e в одном из четырех случаев:

а) суффикс r_1 совпадает с префиксом r_2, тогда e положительно инцидентно v_1 и отрицательно инцидентно v_2;

б) суффикс r_1 совпадает с префиксом r_2^{rc} (с помощью r^{rc} обозначается последовательность, обратно-комплементарная r), тогда e положительно инцидентно v_1 и положительно инцидентно v_2;

в) суффикс r_1^{rc} совпадает с префиксом r_2, тогда e положительно инцидентно v_1 и отрицательно инцидентно v_2;

г) суффикс r_1^{rc} совпадает с префиксом r_2^{rc}, тогда e отрицательно инцидентно v_1 и отрицательно инцидентно v_2.

Аналогичным образом можно построить и двунаправленный граф де Брейна.

Задача китайского почтальона в случае двунаправленных графов все также может быть решена за полиномиальное время [34].

1.2.6. Сборка генома с учетом покрытия

Глубокое покрытие, получаемое с помощью современных методов секвенирования, позволяет использовать метод максимального правдоподобия. В статье [37] был предложен алгоритм определения частот появления k-меров в геноме при наличии чтений одинаковой длины k. Пусть дан циклический геном G длины $N(G)$, а d_a – число позиций с k-мером a на данной позиции. Вероятность появления k-мера a в чтениях при равновероятном выборе позиции равна $\frac{d_a}{N(G)}$. Так как события выбора позиции независимы, то вероятность события появления имеющегося набора чтений равна:

$$P\left(\prod_a [X_a = x_a]\right) = \frac{n!}{\prod x_a} \prod_a \left(\frac{d_a}{N(G)}\right)^{x_a},$$

где X_a – случайная величина, равная числу появлений k-мера a в чтениях, x_a – конкретные значения этой случайной величины, n – число чтений. Эту вероятность можно считать правдоподобностью неизвестных параметров распределения d_a при условии известных значений случайных величин X_a. Если считать длину генома фиксированной (ее можно оценить заранее), логарифм правдоподобности распадается на сумму выпуклых функций $c_a(d_a)$. Построим двунаправленный граф перекрытий и удалим в нем транзитивные ребра. Затем преобразуем его, раздвоив каждую вершину v на вершины v_- и v_+ так, что входящие ребра в v будут направлены в v_-, а исходящие ребра из v будут выходить из v_+. Также проведем двунаправленное ребро между вершинами v_- и v_+ с минимальной

пропускной способностью, равной одному, и стоимостью, заданной выпуклой функцией c_a. Если в этом графе найти двунаправленный поток минимальной стоимости, то получится вектор частот появлений *k*-меров в геноме, максимизирующий правдоподобность.

В статье [38] модель была уточнена и расширена на случай чтений разной длины. В этой статье задача поиска вектора частот, максимизирующего функцию правдоподобия, сводилась к задаче оптимизации выпуклой функции с линейными ограничениями, которую предлагалось решать с помощью программного средства *MOSEK* [38].

При известных частотах чтений задача сборки генома может также формулироваться как поиск надпути в графе де Брейна с кратными ребрами (*De Bruijn Superwalk with Multiplicities*) – в графе, где для каждого ребра *e* указано число k_e (сколько раз результирующий путь должен пройти по этому ребру). Данная задача является *NP*-трудной [39].

1.2.7. Методы учета парной информации

С развитием методов сборки генома все большую важность приобретает задача правильной обработки повторов в геноме. Многие существующие сборщики основаны на идеях, никак не учитывающих наличие повторов в геноме. Например, методы, основанные на графах де Брейна, не способны правильно обрабатывать повторы достаточно большой длины.

Эта проблема может быть решена при помощи увеличения длины чтений, однако это сложно осуществить физически. Другой подход – использовать парные чтения. При этом подходе секвенаторы предоставляют данные в виде пар чтений. При этом известно вероятностное распределение расстояний между чтениями в паре. Это расстояние может быть сделано значительно большим, чем длины самих чтений. Таким образом, появляется возможность использовать парную информацию при сборке генома.

Было предложено несколько методов учета парной информации. Большинство из них используют отдельно обычный граф де Брейна и парную информацию. В подходе, описанном в работе [40], предлагается применять специально созданный *парный граф де Брейна* вместо обычного. Приведем описание данного представления.

Предположим, что геном представляет собой циклическую строку, а все чтения имеют одинаковую длину l. Предположим, что парные чтения прочитаны с некоторых позиций i и j соответственно. Следовательно, пары чтений находятся на расстоянии $d = j - i$ друг от друга. Будем также предполагать, что расстояния между всеми парами чтений одинаковы.

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

Введем следующие обозначения:

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

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

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

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

Построение парного графа де Брейна происходит следующим образом. Предположим, что необходимо построить такой граф для набора $(k+1, d)$-меров, обозначаемого C. Тогда выполняются следующие шаги:

а) Пусть G_0 – граф из $2|C|$ вершин. Для каждого $(k+1, d)$-мера $(a|b)$ из C в граф добавляются вершины u и v, а также ребро $u \to v$. Ребро помечается меткой $(a|b)$, вершина u – меткой $prefix(a|b)$, а вершина v – $suffix(a|b)$.

б) Производится «склеивание» вершин, имеющих одинаковые метки, в одну вершину. При этом все входящие и исходящие ребра переносятся в новую вершину.

Смысл склеивания состоит в том, чтобы объединить несколько разных вершин, соответствующих одному и тому же месту в геноме, в одну вершину.

В работе [40] также приводится пример такого парного графа. Он приведен на рисунке 8 (a – обычный граф де Брейна, b – парный граф де Брейна, графы построены по одной и той же строке). Для данного примера в парном графе де Брейна оказалось больше вершин, чем в обычном графе. Эта особенность является преимуществом данного графа перед обычным графом, так как в нем меньше вершин, полученных из разных частей генома, которые «склеились» в одну. Благодаря этому, вершина в парном графе де Брейна единственным образом задает место в исходном геноме, что может повысить качество его сборки.

Напомним, что путь в обычном графе де Брейна соответствовал подстроке генома. При использовании парного графа путь будет соответствовать двум подстрокам разной направленности. При этом для преобразования пути в последовательность можно использовать любую двух выделенных подстрок.

Главный недостаток данного представления данных состоит в том, что чтения, получаемые секвенаторами, обычно располагаются не на фиксированном расстоянии друг от друга, а в некотором диапазоне $[d - \Delta, d + \Delta]$. Из-за этого происходят сложности при склеивании двух вершин, задающих одно и то же место в геноме, в одну. Значения d и Δ обычно зависят от используемого секвенатора и известны заранее.

Для устранения описанных сложностей в работе [40] был также предложен *примерный парный граф де Брейна* (*approximate paired de Bruijn Graph, APBG)* для набора *(k+1, d, Δ*)-меров. Пара *(k+1)*-меров *a* и *b*,

называется *(k+1, d, Δ)*-мером, если они располагаются на расстоянии $[d - \Delta, d + \Delta]$ друг от друга в исходном геноме.

Построение примерного парного графа де Брейна происходит следующим образом. Первоначально чтения преобразуются в набор *(k+1, d, Δ)*-меров, обозначаемого *C*. После этого выполняются первый шаг построения обычного парного графа де Брейна (построение графа G_0 с $2|C|$ вершинами). Склеивание вершин *(α|β)* и *(α|β`)* в одну происходит в том случае, если существует путь из вершины *β* в вершину *β`* (или наоборот) длины не более *2Δ* в обычном графе де Брейна, построенному по множеству *C*. При склеивании все входящие и исходящие ребра переносятся в новую вершину.

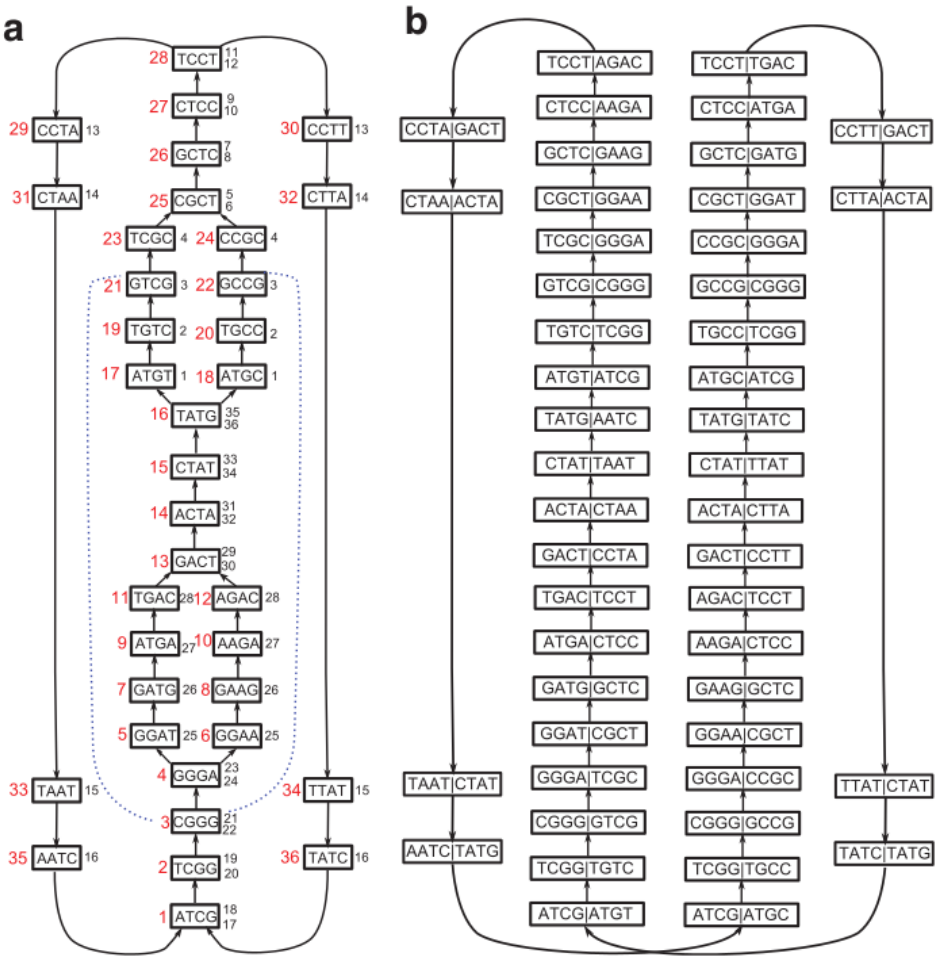


Рисунок 8 – Графы де Брейна, построенные по одинаковой строке:
(a) обычный граф де Брейна; (b) парный граф де Брейна.
Рисунок взят из статьи [40].

Другой подход к использованию парной информации был предложен в работе [41] – *графы множеств путей* (*Pathset Graph*). Их построение происходит следующим образом. Сначала строится обычный граф де Брейна из чтений без учета парной информации. После этого происходит сжатие графа де Брейна (замена каждого неветвящегося пути ребром, длина ребра равна числу ребер в первоначальном пути). Дальше используется парная информация: каждая пара чтений сначала разбивается на набор пар k-меров из левого и правого чтения, после чего строятся *парные реберные интервалы* (e_1, e_2, a, b), где e_1 – ребро, в котором находится некоторый k-мер из левого чтений, а e_2 – ребро с соответствующим ему k-мером из правого чтения. Расстояние между парами k-меров должно быть в отрезке $[a, b]$. Далее для построенного интервала строится *множество путей* (*pathsets*), которое состоит из всех путей, начинающихся с ребра e_1 и заканчивающихся ребром e_2 и с заданной длиной пути (в отрезке $[a, b]$). После выполнения нескольких шагов по упрощению полученных множеств, происходит построение графа, вершинами которого являются полученные множества. Поиск неветвящихся путей в нем позволяет построить контиги.

1.2.8. Подходы к уменьшению используемой памяти при хранении графа де Брейна

В 2012 году эксперты в области сборки геномов выделяли трудность в компактном хранении графа де Брейна как основную проблему при сборке больших геномов [48].

В том же году несколько групп ученых начали изучать вопросы возможности проведения сборки геномов при ограниченных системных ресурсах, а также предлагали алгоритмы и структуры данных для уменьшения необходимой памяти при сборке [49, 50]. В таких исследованиях регулярно ставятся вопросы теоретических оценок для минимальных объемов, необходимых сохранения основной структуры

данных при сборке – графа де Брейна. Например, в работе [50] были доказаны несколько нижних оценок на размер графа де Брейна.

Среди возможных подходов к уменьшению потребляемой памяти – использование фильтров Блума для хранения графа [48–50, 52], навигационных структур данных [50], *rank/select* словарей [51], использование минимизаторов (*minimizers*) для хранения максимальных простых путей графа [50], хранение разряженного графа де Брейна [58] и другие.

Фильтром Блума (*Bloom filter*) называется реализация вероятностного множества, предложенная Б. Блумом в 1970 году, которая позволяет компактно хранить элементы и проверять принадлежность элемента к множеству. *Вероятностное множество* (*probabilistic set*) – структура данных, способная добавлять элемент во множество, а также выполнять запросы проверки принадлежности элемента множеству. При этом возможно получение двух ответов на запрос проверки: во множестве такого элемента нет или элемент возможно присутствует в нем.

Фильтр Блума представляет собой битовый массив из m элементов (битов) и набор из k различных хеш-функций. Его работа происходит следующим образом. Первоначально массив m заполнен нулями. При добавлении элемента e вычисляются значения всех хеш-функции $h_1(e)$, $h_2(e)$, …, $h_k(e)$ и устанавливаются в единицы все значения $m[h_1(e)]$, $m[h_2(e)]$, и т. д. При проверке принадлежности элемента e множеству проверяются значения $m[h_1(e)]$, $m[h_2(e)]$, и т. д. – если везде установлены единицы, то элемент входит во множество, если хотя бы один ноль – элемента точно нет. Отличительной особенностью фильтра Блума является то, что он позволяет использовать любой объем памяти при хранении элементов, однако при слишком маленьких объемах число неверных положительных ответов сильно растет (структура говорит, что элемент во множестве, но на самом деле его там нет).

Таким образом, многие из предложенных алгоритмов для минимизации памяти имеют хорошие теоритические оценки по хранению графа де Брейна и работе с ним. Однако на практике такие подходы применяются достаточно редко. Среди возможных причин – не все из них имеют готовую реализацию в виде библиотек или законченных программ по сборке генома.

1.2.9. Существующие программы сборки генома

Все современные сборщики генома работают с графовым представлением задачи – это и удобно в представлении, и практично для написания алгоритмов. Существует две основных структуры для графового представления – *граф перекрытий* (*overlap graph*) и *граф де Брейна* (*de Bruijn graph*). Обычно сборщики работают с одним из этих двух графов, реализуя все шаги по сборке на нем.

Несмотря на разницу в представлении данных в графе де Брейна и графе перекрытий, все основные шаги по сборке (алгоритмы) могут с небольшими изменениями работать на каждом из них. Разница состоит в том, насколько удобно и компактно хранятся исходные данные в этих структурах. Например, граф де Брейна рекомендуется использовать при большом покрытии – это поможет уменьшить используемую память, а граф перекрытий – при длинных чтениях. Это позволяет более полно использовать всю имеющуюся информацию. Автором были проанализированы зависимости в требуемой памяти для хранения данных в представленных графах (при простых реализациях данных структур). Результаты описаны в разделе 2.1.

Для анализа существующих программ сборки генома *de novo* была составлена сводная таблица распространенных сборщиков вместе с их характеристиками и особенностями (таблица 3).

Таблица 3 – Сводная таблица сборщиков

Название	**Годы разработки**	**Авторы / Компания**	**Возможность собирать большие геномы**	**Основан на**	**Поддерживаемые технологии секвенирования**	**Поддерживаемые операционные системы**	**Дополнительная информация**
ABySS [43]	2008-2015	*Simpson J.* et al.	Да	Граф де Брейна	*Illumina*/*Solexa*, *SOLiD*	*Linux*, *Mac OS*	
Allpaths-LG [53]	2009-2015	*Broad Institute*	Да	Граф де Брейна	*Illumina*/*Solexa*, *PacBio*	*Linux*	
Celera WGS Assembler (CABOG) [79]	1999-2015	*Myers, G.* et al.; *Miller G.* et al.	Да	Граф перекрытий	*Sanger*, 454, *Illumina*/*Solexa*	*Linux*	
CLC Genomics Workbench [95]	2008-2016	*CLC bio*	–	Граф де Брейна	*Sanger*, 454, *Illumina*/*Solexa*, *SOLiD*, *IonTorrent*	*Linux*, *Mac OS*, *Windows*	**Коммерческий продукт**, пробная версия на 14 дней. Возможность запуска через

							графический интерфейс.
ITMO Genome Assembler [102, 103, 108] (предлагаемое решение)	2010-2016	*Alexandrov A., Kazakov S., Melnikov S., Sergushichev A., Tsarev F., Shalyto A.*	Да	Граф де Брейна, граф перекрытий	*Sanger, Illumina/Solexa, IonTorrent*	*Linux, Mac OS, Windows*	Возможность запуска через графический интерфейс.
MaSuRCA [54]	2012-2015	*Zimin, A.* et al.	Да	Граф де Брейна, граф перекрытий	*Sanger*, 454, *Illumina/Solexa*	*Linux*	
Minia [49]	2012-2016	*R. Chikhi, G. Rizk.*	Да	Граф де Брейна	*Sanger, Illumina/Solexa*	*Linux, Mac OS*	
MIRA [55]	1997-2015	*Chevreux B.*	–	Граф перекрытий	*Sanger*, 454, *Illumina/Solexa, IonTorrent, PacBio*	*Linux, Mac OS*	

Newbler (*GS De Novo Assembler*)	2009-2013	*454 Life Sciences*/*Roche*	–	Граф перекрытий	454, *Sanger*	*Linux*	Возможность запуска через графический интерфейс.
SOAPdenovo2 [57]	2009-2013	*Luo R.* et al.	Да	Граф де Брейна	*Illumina*/*Solexa*	*Linux*, *Mac OS*	
SPAdes [56]	2012-2016	*Bankevich A.*, *Nurk S.* et al.	Нет	Граф де Брейна	*Illumina*/*Solexa*, *IonTorrent*, *PacBio*, *Oxford Nanopore*, *Sanger* , 454	*Linux*, *Mac OS*	
SparseAssembler [58]	2012-2015	*Ye C.*, et al.	Да	Граф де Брейна	*Sanger*, *Illumina*/*Solexa*	*Linux*	
Velvet [42]	2007-2013	*Zerbino D.*, *Birney E.*	–	Граф де Брейна	*Illumina*/*Solexa*, *Sanger*, 454	*Linux*, *Mac OS*	

1.2.10. Анализ программ по возможности их использования при обучении

Для анализа возможностей использования существующих программ сборки генома *de novo* при обучении, были выбраны следующие критерии, по которым было произведено сравнение известных сборщиков:

1. Возможность свободного использования.
2. Возможность работы на персональном компьютере (при небольшом объеме оперативной памяти).
3. Кроссплатформенность.
4. Наличие графического интерфейса пользователя.
5. Возможность простого запуска (одной командой, без выбора дополнительных параметров сборки).

Критерий «возможность работы на персональном компьютере» подразумевает возможность работы сборщика на реальных данных секвенирования бактериального генома с использованием менее 2 Гб оперативной памяти.

Сравнение существующих сборщиков по выбранным критериям приведено в таблице 4.

Таблица 4 – Сравнение существующих программ сборки генома

Название сборщика	**1**	**2**	**3**	**4**	**5**
ABySS	+	–	–	–	+
Allpaths-LG	+	–	–	–	–
CLC Genomics Workbench	–	+	+	+	+
MaSuRCA	+	–	–	–	–
Minia	+	+	–	–	–
Newbler	±	–	–	+	–
SPAdes	+	–	–	–	+
Sparse Assembler	+	+	–	–	–
Velvet	+	–	–	–	–

Из данной таблицы следует, что наиболее подходящим для образовательного процесса является сборщик *CLC Genomics Workbench*, который, однако, **не распространяется свободно, является платным** и весьма дорогим. Остальные сборщики обычно не являются кроссплатформенными, не имеют графического интерфейса или требуют «специальных» знаний для их запуска. Эти особенности приводят к усложнению их использования при обучении геномной биоинформатике.

1.2.11. Использование результатов сборки

Обычно результаты сборки далее анализируются для получения информации об исследуемом организме. Например, большинство работ в журнале *Genome Announcements* [96] после непосредственной сборки исходных данных производят аннотирование полученной сборки и анализируют представленные и отсутствующие гены в ДНК. Регулярно выполняется сравнение полученной сборки с геномами родственных организмов для получения отличительные особенности генома, что может помочь с пониманием тех или иных особенностей конкретного организма, его поведения и условий обитания. Имея собранные геномы организма и близких видов можно построить филогенетическое дерево, отражающее эволюционные взаимосвязи между ними. Также в последнем случае можно попытаться восстановить историю популяции, проследить общего предка и оценить изменения численности популяции со временем (например, в статье [46] был проведен такой анализ для популяции гепардов).

Для некоторых классов задач анализ итоговой сборки вместо непосредственно исходных чтений является большим недостатком – в основном из-за уменьшения доступной информации для анализа. К таким задачам можно отнести генотипирование (определение индивидуальных особенностей генома) и исследование числа копий определенных сегментов генома (*copy number variation*, *CNV*). В метагеномных

исследованиях анализ сборок вместо чтений может привести к значительному уменьшению разнообразия микроорганизмов, представленных в исследуемой среде.

1.3. Анализ данных метагеномного секвенирования

При метагеномном секвенировании ученые изучают сообщество организмов (метагеном).

При изучении конкретного метагенома существует несколько главных вопросов, на которые пытаются ответить:

- Какие организмы присутствуют в среде, или определение *таксономического состава*.
- Какие функции могут выполнять присутствующие в среде организмы, или определение *функционального состава*. При этом определяется набор генов, которые содержатся в геномах организмов.
- Насколько велико разнообразие сообщества? Разнообразие может быть измерено численно за счет подсчета числа различных организмов, входящих в сообщество (*species richness*).

При изучении набора метагеномов также исследуют такой вопрос, как степень сходства и различия разных метагеномов (иногда обозначаемое, как *бета-разнообразие сообществ*). Более точно, ученых интересуют изменения в составе сообщества при переходе от одного метагенома к другому. Область знаний, которая пытается ответить на этот вопрос, со временем получила название – *сравнительная метагеномика*.

Как уже было сказано, метагеномное секвенирование отличается от геномного тем, что достоверно не известно, из какого организма пришло каждое чтение (в случае геномного секвенирования все чтения принадлежат одному образцу). Это, в свою очередь, накладывает дополнительные условия при анализе таких чтений.

Можно выделить следующие основные трудности при анализе метагеномов:

- Большая доля некультивированных микроорганизмов (нет референсных последовательностей).
- Сложная структура сообщества (в том числе регулярно встречается большое число самых разнообразных организмов в метагеноме).
- Большие объемы данных секвенирований.

Изучение микробных сообществ играет важную роль в биологии. С помощью анализа микробиоты люди смогут ответить на множество фундаментальных вопросов, в том числе вопросы о работе бактерии в кишечнике человека, о вреде, который антибиотики и другие лекарства могли нанести микрофлоре в течение последнего века, о роли тех или иных бактерии в жизни человека и других млекопитающих. Все это, в свою очередь, может помочь в разработке средств персонализированной медицины, при поиске и анализе редких микроорганизмов из пока неизученных сред и для других прикладных задач.

1.3.1. Сравнительный анализ в метагеномике

Сравнительная метагеномика – область метагеномики, которая занимается вопросами соотношения между разными метагеномами. При этом сравнивать метагеномы можно как из разных сред обитания, так и из одной. Примером задачи с одной средой обитания может быть, например, анализ изменений состава метагенома некоторого озера в разные моменты времени.

В сравнительной метагеномике ученых может интересовать не только вопрос сходства или различия двух метагеномов, а также вопросы выделения конкретных признаков, которые отличают одно сообщество от другого. Такими признаками могут быть, например, конкретные организмы (или их ДНК), которые присутствуют в одном метагеноме и

отсутствуют в другом. Также признаком может быть разная представленность некоторых микроорганизмов в разных сообществах.

1.3.2. Существующие подходы для сравнительного анализа метагеномов

Появившиеся возможности быстрого и относительно недорогого секвенирования привели к большому числу проектов по всему миру, целью которых являлось секвенирование и изучение геномов отдельных организмов. Это, в свою очередь, стало основой для быстрого развития так называемых референсных методов (*reference-based methods*, также называемых традиционными методами), основанных на выравнивании исходных метагеномных данных на каталог известных геномов. Впоследствии были придуманы более изощренные методы использования каталога референсных геномов, чем прямое выравнивание каждого чтения на каждый референс. Примеры таких методов – *Kraken* [71], *CLARK* [72], *FOCUS* [65], *MetaPhlAn2* [73] и т. д. Однако, большой проблемой для таких подходов стали сообщества микроорганизмов из ранее неизученных ниш, которые содержат большое число неизученных бактерий. Соответственно, отсутствуют эталонные последовательности геномов для многих видов таких бактерий и вирусов. Данная проблема является значительной и для сред, которые уже изучаются десятилетиями: например, микробиота кишечника человека. По оценкам экспертов, больше половины бактерий микробиоты до сих пор не изучены и не имеют референсных последовательностей [60].

Для тог чтобы справиться с напором увеличившегося объема информации, были разработаны, в том числе, методы, основанные на адаптивной выборке данных [61].

Также, ввиду отсутствия представительной каталогизации, для исследователей метагеномов становятся привлекательными так называемые безреференсные подходы (*alignment-free methods*). Среди них,

с одной стороны, имеются абстрактные методы, основанные на разложении (*composition-based methods*), к которым относятся анализ *k*-мерного спектра [62–67], нейронные сети [68], марковские модели [69] и т. д. Такие методы высокоскоростные и параллелизуемые. Однако к недостатку такого подхода относится тот факт, что в результате анализа отдельные признаки (идентифицированные как различающие две или несколько групп метагеномов, либо коррелирующие с каким-либо экспериментальным фактором) являются либо сокрытыми внутри метода, либо малоинформативными.

Из данной группы методов можно отдельно выделить методы, основанные на построении *k*-мерных групп (*k-mer binning methods*). Данные методы также являются абстрактными методами, основанными на разложении. Примеры таких методов – *AbundanceBin* [63], *CompostBin* [64], *MaxBin* [66], *MetaCluster* [70] и т. д. Благодаря использованию групп *k*-меров вместо отдельных *k*-меров устойчивость таких подходов к малым изменениям исходных данных значительно увеличивается, однако смысл и информативность таких признаков все еще остается под большим вопросом.

Альтернативным подходом к вычислению попарного расстояния между метагеномами является совместная сборка метагеномов *de novo* (по аналогии с тем, как это делают для отдельных геномов) с последующим анализом полученных контигов (классификация, дифференциальный анализ покрытия контигов чтениями). Такие методы получили название – методы, основанные на совместной сборке (*assembly-based methods*). Несмотря на осмысленность каждого полученного признака, сборка метагенома затруднена ввиду широкого диапазона степени представленности разных бактерий и внутривидовой геномной вариабельностью. Специальные алгоритмы для таких целей были разработаны – *Meta-IDBA* [74], *MetaVelvet* [75], *Ray Meta* [76], *MetAMOS* [77]. В частности, специально для вычисления попарного

различия путем анализа совместной сборки чтений был разработан метод *crAss* [78]. Однако полная сборка от чтений до контигов является вычислительно сложной задачей, особенно для метагеномных данных больших объемов.

Таким образом, можно разделить все имеющиеся методы для сравнительного анализа на несколько классов в зависимости от реализуемых подходов:

- Традиционные методы, основанные на картировании на каталог известных геномов (reference-based methods) – *Kraken, CLARK, FOCUS, MetaPhlAn2*.
- Основанные на совместной сборке исходных данных (assembly-based methods) – *Meta-IDBA, MetaVelvet, Ray Meta, MetAmos, crAss.*
- Основанные на абстрактном разложении данных (composition-based methods) – анализ *k*-мерного спектра, нейронные сети, марковские модели.
- Основанные на построении *k*-мерных групп (*k*-mer binning methods) – *AbundanceBin, CompostBin, MaxBin, MetaCluster*.

В таблице 5 приводится более детальное описание выделенных классов, их преимуществ и недостатков.

Из данной таблицы следует, что все описанные методы имеют существенные недостатки, которые ограничивают область их применимости. Среди них – требование наличия репрезентативной базы геномов, необходимость больших вычислительных ресурсов для работы, малоинформативность получаемых признаков и сложный запуск имеющихся программ. Указанные недостатки приводят к необходимости искать другие решения для анализа набора метагеномов, а требование больших вычислительных ресурсов и сложный запуск программ усложняют их использование в учебном процессе.

Таблица 5 – Существующие подходы для сравнительного анализа метагеномов

Класс методов	**Основные шаги**	**Использование базы референсных геномов**	**Требуемое время работы / память**	**Другие преимущества и недостатки**
Традиционные методы, основанные на выравнивании чтений на каталог известных геномов (*reference-based methods*)	1. Профилирование состава 2. Вычисление попарного сходства	Да	Зависит от метода	**Преимущества:** • Выявляется таксономический состав • Позволяют определить бактерии, отличающие два разнопредставленных метагенома **Недостатки:** • Требуют репрезентативной базы геномов
Методы, основанные на	1. Совместная сборка данных	Нет	Много/	**Преимущества:**

сборке исходных данных (*assembly-based methods*)	2. Выравнивание исходных чтений на контиги 3. Подсчет представленности каждого контига в исходных метагеномах 4. Вычисление попарного сходства		много	• Позволяют восстановить геномы бактерий, представленные в метагеномах (контиги сборки) **Недостатки:** • Вычислительно трудоемкий процесс сборки
Методы, основанные на абстрактном разложении данных (*composition-based methods*)	1. Извлечение большого числа абстрактных признаков из исходных данных 2. Подсчет попарного сходства по этим признакам	Нет	Мало/ мало	**Преимущества:** • Работают очень быстро **Недостатки:** • Малоинформативность получаемых признаков
Методы, основанные на построении *k*-мерных групп (*k-mer binning*	1. Построение *k*-мерного спектра для каждого метагенома отдельно	Нет	Среднее/ среднее	**Недостатки:** • Малоинформативность получаемых признаков

methods)	2. Построение *k*-мерных групп (*k-mer bins*) по какому-либо параметру сходства 3. Подсчет представленности каждой *k*-мерной группы в исходных метагеномах 4. Вычисление попарного сходства			

1.4. Задачи, решаемые в диссертационной работе

Как отмечалось в разделе 1.1.2, технологии секвенирования развиваются стремительно – появляются новые технологии, уже известные методы оптимизируются и снижается их стоимость. При падении стоимости секвенирования доступность секвенаторов заметно возрастает, что ведет к значительному увеличению скорости производства данных по всему миру. Для обработки получаемых данных необходимы как вычислительные мощности и эффективные алгоритмы для их анализа, так и квалифицированные специалисты, способные анализировать и интерпретировать результаты обработки таких данных. Подготовка студентов и магистрантов по биоинформатическим специальностям ведется по всему миру.

При этом, как уже было отмечено в разделах 1.2.10 и 1.3.2, существующие программы сборки генома и сравнительного анализа метагеномов плохо подходят для такого обучения, так как их установка, настройка и запуск на компьютерах обучающихся, которые обычно являются персональными, плохо осуществимы.

Также были выявлены следующие недостатки существующих подходов:

- Большинство программ по сборке генома и сравнительному анализу метагеномов работают только под одной операционной системой *Linux*.
- Подавляющее большинство сборщиков не имеет возможности указывать объем памяти для использования, и поэтому их запуск требует больших вычислительных ресурсов, которые имеются только на серверах или кластерах.
- Часть сборщиков довольно сложно запускать: они требуют разные сторонние средства для своей работы, а для запуска всего

процесса сборки бывает необходимым запускать несколько команд последовательно.

- Большинство сборщиков генома имеют много параметров для ручной настройки, которые меняют качество итоговой сборки. При этом настройка таких параметров занимает довольно большое время и не всегда очевидна для конечного пользователя.

На основании проведенного обзора сформулируем задачи, решаемые в диссертационной работе:

1. Разработать автоматизированный метод сборки генома *de novo* на основе совместного применения графа де Брейна и графа перекрытий, оптимизированный по объему используемой памяти и применимый при обучении геномной биоинформатике.
2. Разработать автоматизированный метод сравнительного анализа метагеномов на основе анализа компонент связности в графе де Брейна, оптимизированный по вычислительным ресурсам и применимый в учебном процессе.
3. Произвести экспериментальные сравнения разработанных программ с существующими программами по сборке генома и сравнительному анализу метагеномов. Сравнение необходимо производить по метрикам качества получаемых результатов и необходимым вычислительным ресурсам.

Выводы по главе 1

1. Приведены основные понятия и определения, относящиеся к области биоинформатики.
2. Выполнен обзор существующих методов секвенирования генома, составлена сводная таблица с их характеристиками.
3. Приведен обзор существующих подходов сборки генома.
4. Выполнен обзор существующих методов сравнительного анализа метагеномов.
5. Сформулированы основные недостатки существующих методов сборки генома и сравнительного анализа метагеномов.
6. Произведен анализ существующих программ по возможности их использования для обучения геномной биоинформатике.
7. Сформулированы задачи, решаемые в диссертационной работе.
8. Результаты обзора, описанные в данной главе, были опубликованы в работах автора [101, 110, 111], отчетах по государственным контрактам [1, 97–99] и в магистерской диссертации автора [2].

ГЛАВА 2. АВТОМАТИЗИРОВАННЫЙ МЕТОД СБОРКИ ГЕНОМА *DE NOVO* НА ОСНОВЕ СОВМЕСТНОГО ПРИМЕНЕНИЯ ГРАФА ДЕ БРЕЙНА И ГРАФА ПЕРЕКРЫТИЙ

Настоящая глава посвящена разработанному методу сборки генома *de novo*.

В ней приводится описание предлагаемого метода, даются оценки на время работы этапов метода, описывается реализация предложенных подходов, а также приводятся результаты экспериментального исследования, целью которого было сравнение предложенного метода с существующими.

Результаты и методы, описанные в данной главе, были опубликованы в статьях соискателя [101–104, 106–108, 110, 111, 114–119], отчетах по государственным контрактам [1, 98, 99] и в магистерской диссертации автора [2]. Оценки сложности алгоритма *meet-in-the-middle*, описанного в главе 2.2.2, были взяты из работы [100].

2.1. Анализ затрат памяти на хранение графов

Для анализа требуемых ресурсов на хранение исходных данных автором были получены следующие зависимости объема памяти (в байтах) для хранения графа перекрытий (*ov-graph*) и графа де Брейна (*db-graph*):

$$M_{ov-graph} = N \cdot L \cdot \frac{2}{8} + N \cdot \left\lfloor \frac{N(L-L_{ov})}{G} \right\rfloor \cdot 6; \quad (1)$$

$$M_{db-graph} = \left(K_g + \lfloor N \cdot L \cdot p_{er} \rfloor \cdot \frac{k(L-k+1)}{L}\right) \cdot 8 \cdot \frac{4}{3}, \quad (2)$$

где N – число чтений, L – средняя длина чтения, G – длина генома, L_{ov} – длина перекрытия, k – размер k-мера, K_g – число безошибочных k-меров, p_{er} – вероятность ошибки в одной позиции чтения.

Данные зависимости были получены для случая упрощенной модели получения данных, которая состояла в следующем. Будем считать, что

исходные данные состоят из N чтений, каждое из которых имеет длину L. При этом все чтения равномерно покрывают исходный геном длины G. Также для упрощения формул (1) и (2) предполагалось, что не существует повторов в геноме длины большей или равной, чем размер k-мера, поэтому безошибочные k-меры из разных частей генома не совпадают. Предполагается, что в чтениях могут встречаться ошибки: на каждой позиции с вероятностью p_{er} могла быть совершена ошибка, а события появления ошибок в разных позициях чтений являются независимыми.

При получении зависимостей (1) и (2) считалось, что хранение графов происходит следующим образом. Граф перекрытий хранится в виде двух структур данных: массива чтений (которые являются вершинами) и списков смежности для хранения ребер. Кодирование одного нуклеотида занимает два бита, чтение длины L будет занимать $L \cdot \frac{2}{8}$ байт памяти. Перекрытия сохраняются в виде номеров вершин (четыре байта на номер) и сдвига (два байта). Граф де Брейна хранится в виде набора k-меров в хеш-таблице с открытой адресацией. На сохранение k-мера используется восемь байт (для $k \leq 32$), коэффициент заполнения хеш-таблицы $\frac{3}{4}$.

В случае уникальности каждого k-мера в исходном геноме и $G \gg k$ также выполняется следующее соотношение:

$$K_g = G - k + 1 \approx G. \quad (3)$$

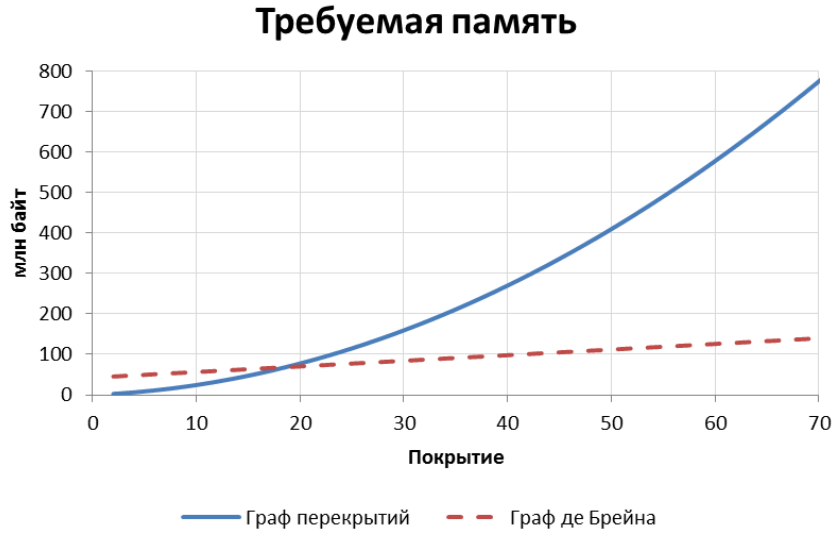
Покрытие генома чтениями, обозначаемое c, определяется как среднее число чтений, покрывающих некоторую позицию в геноме, и может быть вычислено по формуле:

$$c = \frac{N \cdot L}{G} = \frac{\sum_{i=1}^{N} L_i}{G}. \quad (4)$$

Полученные зависимости (1) и (2) отображаются на двух графиках, представленных на рисунке 9. На первом из них фиксируется длина чтения

$L = 100$ нукл., а переменными являются число чтений N и покрытие генома чтениями c. На втором графике при фиксированном покрытии $c = 20$ переменными являются средняя длина чтения L и число чтений N (с сохранением равенства $c = 20$).

А)



Б)

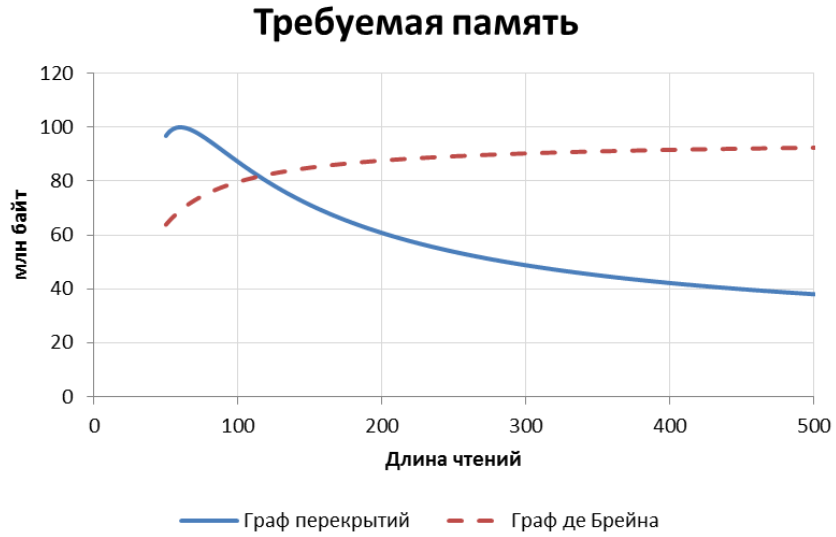


Рисунок 9 – Зависимости требуемой памяти для хранения графов перекрытий и де Брейна при А) изменении покрытия генома чтениями, Б) изменении длины чтений

Из представленных графиков следует, что при большом покрытии генома чтениями ($c > 40$) граф перекрытий требует значительно больше памяти, чем граф де Брейна. При небольшом покрытии ($c = 20$) и длине

чтений $L = 100$ нукл. оба графа используют примерно одинаковый объем памяти, однако при увеличении длины чтения граф перекрытий экономнее хранит имеющиеся чтения.

Из представленных данных следует, что для достижения лучших результатов на разных этапах сборки целесообразно использовать разные структуры данных. Автором было предложено применять граф де Брейна на этапе сборки квазиконтигов из парных чтений, где данных может быть много, а длина чтения небольшой (обычно 35–100 нукл.), и граф перекрытий на этапе сборки контигов из квазиконтигов, где исходные данные (квазиконтиги) обычно достаточно длинные (в среднем по 500 нукл.).

2.2. Метод сборки генома

Предлагаемый подход состоит в совместном применении графа де Брейна и графа перекрытий на разных этапах сборки, что позволяет использовать преимущества обеих структур данных.

Сборку генома *de novo* предлагается проводить в несколько этапов:

- исправление ошибок секвенирования (основано на анализе *k*-мерного спектра);
- сборка квазиконтигов из парных чтений (выполняется на графе де Брейна);
- сборка контигов из квазиконтигов (выполняется на графе перекрытий).

Архитектура предлагаемого решения показана на рисунке 10.

Метод принимает на вход парные чтения в формате *FASTQ* [13, 92], однако также может работать и с чтениями в формате *FASTA* [91]. Результатом работы программы являются собранные контиги, которые записываются в файл формата *FASTA*.

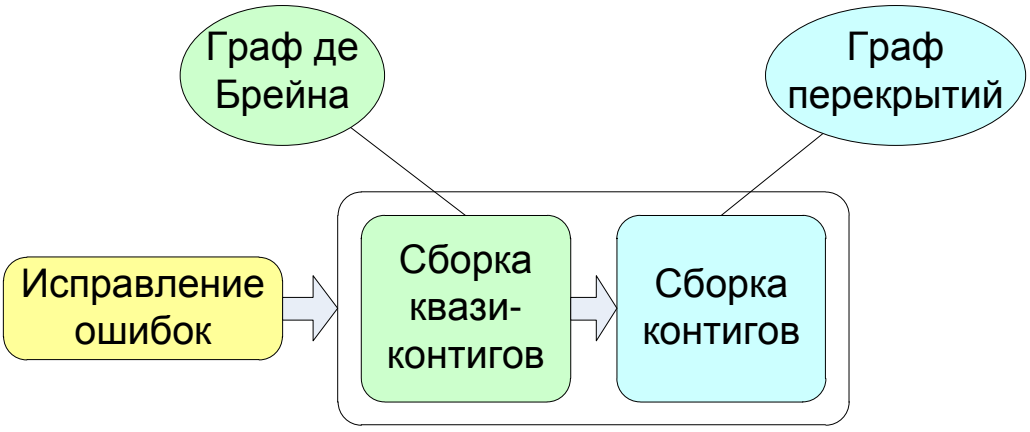


Рисунок 10 – Архитектура предлагаемого решения

Отличительной особенностью предлагаемого решения является то, что на разных этапах сборки используются разные графы – на этапе сборки квазиконтигов используется граф де Брейна, а на этапе сборки контигов – граф перекрытий. Данное решение обусловлено тем, что используемые графы при разном числе чтений и их длине используют разный объем памяти для их хранения. Также известно, что граф перекрытий лучше подходит для сборки длинных чтений, так как использует чтения более полно (использует всю длину чтения, не «нарезая» их на небольшие *k*-меры). Сравнительный анализ затрат на хранение разных исходных данных приведен в разделе 2.1.

Далее подробнее рассматривается каждый из этапов предлагаемого решения. В процессе разработки пробные реализации предлагаемых этапов тестировались в рамках проектов *de novo Genome Assembly Assessment Project workshop (dnGASP)* [93] и *Assemblathon 2* [94].

2.2.1. Исправление ошибок

Приведем описание метода исправления ошибок в данных секвенирования.

Метод основан на частотном анализе содержащихся в чтениях *k*-меров (подстрок длины *k*) и не использует граф де Брейна.

Для эффективного исправления ошибок необходимо, чтобы каждая позиция генома была прочитана несколько раз. Тогда одинаковые *k*-меры без ошибок будут встречаться несколько раз в исходных чтениях, тогда как *k*-меры с ошибками будут встречаться достаточно редко. Данное наблюдение основано на том факте, что ошибки в чтениях возникают на случайных позициях, и, следовательно, вероятность того, что найдется два *k*-мера с одинаковой ошибкой, достаточно мала.

Метод исправления ошибок состоит в следующем. Сначала для каждого *k*-мера вычисляется частота его присутствия в исходных чтениях. На основании этих данных все *k*-меры разделяются на две группы – «надежные» *k*-меры (встречаются не реже некоторого эвристически выбранного порога) и «подозрительные» (редко встречаемые). «Надежные» *k*-меры считаются безошибочными. «Подозрительные» же *k*-меры вызваны либо плохим покрытием тех частей генома, откуда они были прочитаны, либо наличием в них одной или нескольких ошибок.

Пример распределения числа *k*-меров от частоты присутствия *k*-мера в чтениях для реальных данных показан на рисунке 11. Для представленного распределения порог был выбран равным четырем.

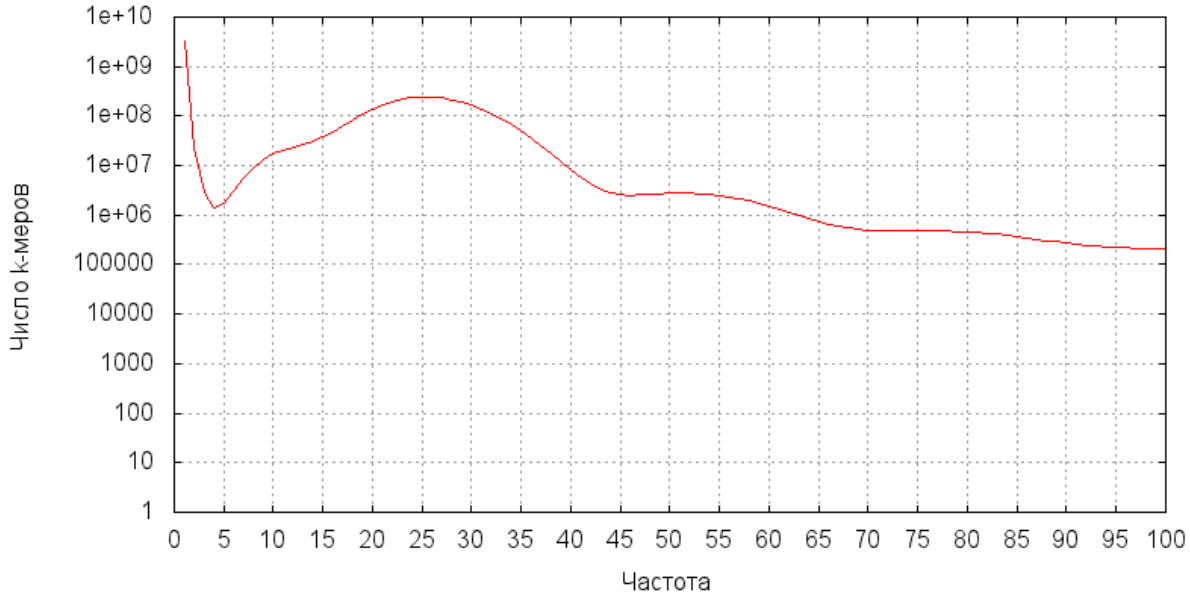


Рисунок 11 – Распределение числа *k*-меров от частоты присутствия *k*-мера в чтениях

Исправление ошибок состоит в замене каждого «подозрительного» *k*-мера на «близкий» надежный *k*-мер, который может быть получен из подозрительного путем изменений небольшого числа нуклеотидов в нем. Для этого выполняется перебор всех позиции *k*-мера и нуклеотида, на который нужно заменить нуклеотид на выбранной позиции. Если получаемый при этом *k*-мер попадает в группу «надежных», то, вероятно, первоначальный *k*-мер является результатом ошибочного прочтения полученного надежного *k*-мера. 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 применяется. Если было найдено несколько возможных вариантов исправления ошибочного *k*-мера, то исправление не производятся из-за неопределенности выбора. Если не было найдено ни одного подходящего исправления для подозрительного *k*-мера, такой *k*-мер тоже не исправляется.

Псевдокод описанного выше метода исправления ошибок приведен на листинге 1.

Листинг 1 − Псевдокод алгоритма исправления ошибок (без разбиения на корзины)

```
function CORRECTERRORS(R, N, K, T)
    R – исходные чтения
    N – число чтений
    K – размер k-мера
    T – порог частоты k-мера, начиная с которой он считается «надежным»
    G – ассоциативный массив для пар <k-мер, частота присутствия в чтениях>
    F – ассоциативный массив для пар <позиция, правильный нуклеотид>
    G ← COUNTKMERS(R, N, K)
    for i ← 1…N do
        F ← {}
        for startPos ← 1…(LENGTH(R_i) – K + 1) do
```

```
            kmer ← GETKMER(R_i, startPos, K)
            if G.GETVALUE(kmer) < T then        – если k-мер «подозрительный»
                found ← 0
                for j ← 1…K do
                    for rightNuc ← {A,C,G,T} do
                        newKmer ← REPLACENUC(kmer, j, rightNuc) – исправить
                        if G.GETVALUE(newKmer) >= T then
                            found ← found + 1
                            lastJ ← j
                            lastRightNuc ← rightNuc
                        end if
                    end for
                end for
                if found == 1 then
                    F.ADD(startPos + lastJ – 1, lastRightNuc)   – добавить исправление
                end if
            end if
        end for
        R_i ← APPLYFIXES(R_i, F)
    end for
    return R      – вернуть исправленные чтения
end function
```

При этом алгоритм разработан с учетом возможного исправления не только одной ошибки, но и большего их числа. Число возможных замен является параметром алгоритма (по умолчанию параметр равен одному). В случае если число возможных замен больше одного, выполняется схожая процедура исправления, при которой перебираются несколько позиций в *k*-мере и несколько нуклеотидов, на которые необходимо исправить выбранные нуклеотиды в *k*-мере. Если после исправления получается «надежный» *k*-мер, исправление запоминается. Если было найдено несколько возможных исправлений, то ни одно из них не применяется

из-за неопределенности выбора. Псевдокод данного алгоритма **не приводится** в тексте диссертации из-за его громоздкости.

Отличительной особенностью предлагаемого алгоритма является то, что в случае нехватки оперативной памяти для выполнения данного алгоритма, все исходные чтения разбиваются на несколько групп, и процесс исправления ошибок выполняется отдельно для каждой из групп. При этом на такое выполнение требуется меньше памяти, чем для всего этапа в целом, позволяя исправить ошибки в меньшем объеме памяти, однако, с увеличением времени работы алгоритма.

Предлагаемое решение, направленное на уменьшения используемой памяти, оказывается возможным благодаря разбиению всех *k*-меров на группы по некоторому небольшому префиксу (в группе оказываются *k*-меры с одинаковым префиксом). При этом исправление ошибок в *k*-мере не выводит исправленный *k*-мер из обрабатываемой групп, если не исправляются ошибки в префиксе этого *k*-мера. Следует отметить, что в случае, если ошибка была совершена в префиксе *k*-мера, она может быть исправлена при обработке обратно-комплементарного *k*-мера, который соответствует данному *k*-меру. В этом случае префикс *k*-мера станет суффиксом, и требуемое исправление будет найдено.

Псевдокод алгоритма исправления ошибок с разбиением *k*-меров на группы по префиксу приведен на листинге 2.

Листинг 2 − Псевдокод алгоритма исправления ошибок с разбиением *k*-меров на группы по префиксу

function CORRECTERRORSWITHPARTITIONING(R, N, K, T, P)

R – исходные чтения

N – число чтений

K – размер *k*-мера

T – порог частоты *k*-мера, начиная с которой он считается «надежным»

P – длина префикса, по которому нужно разбить на корзины

G – ассоциативный массив для пар <*k*-мер, частота присутствия в чтениях>

F – множество троек <номер чтения, позиция, правильный нуклеотид>

```
    CountKmersByGropsAndSaveToDisk(R, N, K, P)
    for p ← GetAllPossiblePrefixes(P) do     – обрабатываем k-меры с префиксом p
        G ← LoadKmersWithPrefix(p)
        for i ← 1…N do
            for startPos ← 1…(Length(R_i) – K + 1) do
                kmer ← GetKmer(R_i, startPos, K)
                if kmer.startsWith(p) AND G.getValue(kmer) < T then
                    found ← 0
                    for j ← (p+1)…K do
                        for rightNuc ← {A,C,G,T} do
                            newKmer ← ReplaceNuc(kmer, j, rightNuc) – исправить
                            if G. getValue(newKmer) >= T then
                                found ← found + 1
                                lastJ ← j
                                lastRightNuc ← rightNuc
                            end if
                        end for
                    end for
                    if found == 1 then
                        F.add(<i, startPos + lastJ – 1, lastRightNuc>)
                    end if
                end if
            end for
        end for
    end for
    R ← ApplyFixes(R, F)
    return R      – вернуть исправленные чтения
end function
```

2.2.2. Сборка квазиконтигов

Приведем описание алгоритма сборки квазиконтигов.

В методе сборки квазиконтигов используется граф де Брейна, в котором множество вершин состоит только из «надежных» *k*-меров.

Заметим, что если некоторый участок нуклеотидной последовательности покрылся чтениями при секвенировании достаточно хорошо – все входящие в него *k*-меры надежные, то в графе де Брейна будет существовать путь между первым и последним *k*-мерами участка. Ввиду того, что парные чтения были получены с концов некоторого фрагмента генома, поиск такого фрагмента в графе де Брейна и производится при сборке квазиконтигов.

Более формально, метод сборки квазиконтигов состоит в поиске пути в графе де Брейна между *k*-мерами из левого и правого чтения. Ввиду того, что распределение длин фрагментов, из которых были получены парные чтения, обычно задано, из всех путей интересны только те из них, которые укладываются в априорные границы длин фрагментов. Поэтому слишком короткие и слишком длинные пути можно отбросить. Если нашелся единственный подходящий путь из всех найденных путей, то он преобразуется в квазиконтиг и записывается в файл. Если не нашлось такого пути, или нашлось несколько подходящий путей, не один из них не используется из-за неопределенности выбора.

Пример уникального подходящего пути в графе де Брейна для $k = 3$ показан на рисунке 12. Пример двух подходящих путей для того же графа показан на рисунке 13.

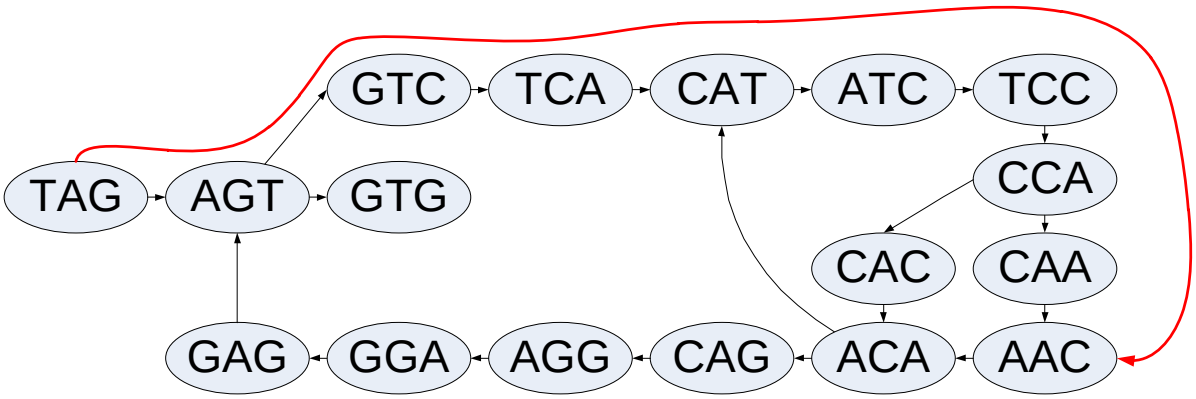


Рисунок 12 – Пример уникального пути в графе де Брейна

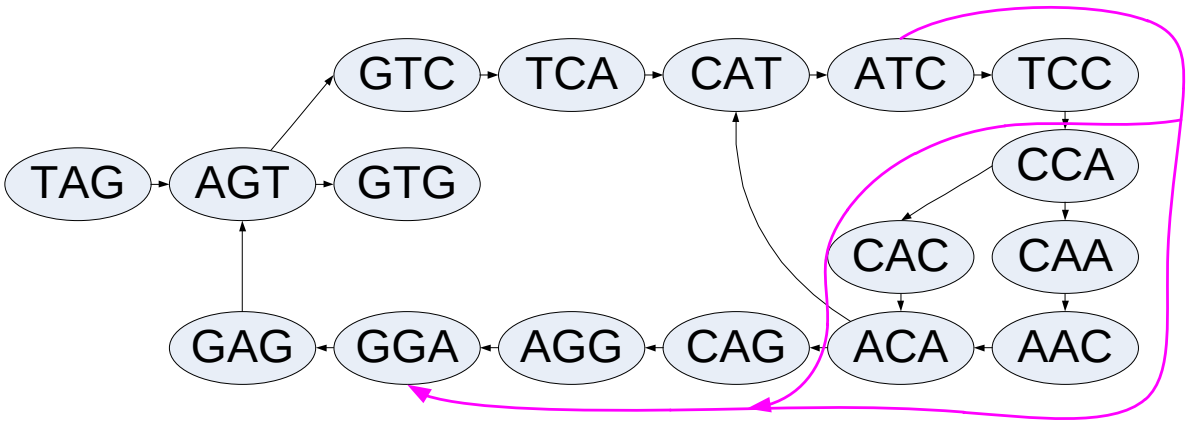


Рисунок 13 – Пример двух подходящих путей в графе де Брейна

Следует отметить, что существует частный случай, когда при условии нахождения нескольких подходящих путей следует вывести хотя бы один из них. Этот случай объясняется тем, что в геноме встречаются однонуклеотидные полиморфизмы (замена одного нуклеотида другим, *single nucleotide polymorphism*, *SNP*), из-за которых соответствующий путь может раздвоиться. В этом случае следует вывести любой из них, чтобы данная часть генома была представлена в полученных квазиконтигах.

Для поиска путей в графе де Брейна применяется подход *meet-in-the-middle*, который состоит в том, что происходит поиск нужного по длине пути одновременно с двух сторон, с помощью двух одновременных обходов в ширину. Первый обход идет по прямым ребрам графа де Брейна, начиная от первого *k*-мера из левого чтения. Второй обход идет по обратным ребрам, начиная с последнего *k*-мера из правого чтения. Таким образом, после l_1 шагов первого обхода строится множество вершин V_1, удаленных от начальной вершины не более чем на l_1 ребер, а после l_2 шагов во втором обходе – множество вершин V_2, удаленных от конечной вершины не более, чем на l_2 ребер. Если в какой-то момент множества V_1 и V_2 пересекаются, то существует путь между начальной и конечной вершиной длины $l_1 + l_2$. Этому пути соответствует нуклеотидная последовательность длины $l_1 + l_2 + k$. Используя деревья обходов в ширину, этот путь можно восстановить.

Данный подход имеет меньшую временную сложность, чем стандартные подходы обхода в ширину или глубину. Пусть необходимо найти путь с длиной в промежутке $[l_{\min};l_{\max}]$. Тогда для поиска такого пути обычным обходом в ширину нужно выполнить $l_{\max}$ шагов обхода от начальной вершины, посетив все вершины, удаленные от начальной не более чем на $l_{\max}$ ребер. Обозначим за d среднюю полустепень исхода вершин графа (среднее число ребер, выходящих из вершины). Тогда поиск пути стандартным методом обхода в ширину потребует $O(d^{l_{max}})$ операций (для случая, когда $d^{l_{max}} < N$, где N – число вершин в графе). При одновременном поиске подходящего пути с двух сторон, каждый из обходов будет искать пути, удаленные от начальной вершины не более чем на $l_{max}/2$ ребер. Для такого поиска потребуется $O(d^{l_{max}/2})$ операций. Итого для поиска пути с длиной в промежутке $[l_{\min};l_{\max}]$ между начальной и конечной вершиной подход *meet-in-the-middle* потребует $O(d^{l_{max}/2})$ операций, что в $O(d^{l_{max}/2})$ раз меньше, чем при поиске пути стандартным обходом в ширину или глубину.

Псевдокод алгоритма поиска подходящего пути приведен на листинге 3.

Листинг 3 − Псевдокод алгоритма поиска подходящего пути в графе де Брейна с помощью подхода *meet-in-the-middle*

function FINDPATH(G, R_{left},R_{right},l_{min},l_{max})

G – граф де Брейна

R_{left}, R_{right} – левое и правое чтение

l_{min},l_{max} – границы длин подходящих путей

V_i – множество вершин, удаленных от начальной вершины не более, чем на *i* ребер (вычисляются итеративно)

U_i – множество вершин, удаленных от конечной вершины не более, чем на *i* ребер (вычисляются итеративно)

P – множество найденных подходящих путей

firstKmer ← GETFIRSTKMER(R_{left})

lastKmer ← GETLASTKMER(R_{right})

```
    V_0.ADD(firstKmer)
    U_0.ADD(lastKmer)
    P ← {}
    for i ← 1…⌈l_max/2⌉ do                  – итерация номер i
        V_i.ADDALL(V_i-1)
        for v in (V_i−1 \ V_i−2) do          – итерация первого обхода в ширину
            for nv in GETNEXTNODES(v) do
                if nv ∉ V_i−1 then
                    V_i.ADD(nv)
                end if
            end for
        end for
        U_i.ADDALL(U_i-1)
        for u in (U_i−1 \ U_i−2) do          – итерация второго обхода в ширину
            for nu in GETPREVNODES(u) do
                if nu ∉ U_i−1 then
                    U_i.ADD(nu)
                end if
            end for
        end for
        for v in (V_i \ V_i−1) do            – проверить пересечения множеств
            if v ∈ U_i then
                path ← RESTOREPATH(firstKmer, v, V) + RESTOREPATH(v, lastKmer, U)
                if LENGTH(path) in [l_min,l_max] then
                    P.ADD(path)
                end if
            end for
        end for
    end for
    if SIZE(P) == 1 OR (SIZE(P) > 1 AND CHECKPATHSAREEQUAL(P)) then
        return {P_1}                         – найден подходящий путь
    end if
    return {}                                – подходящий путь не найден
end function
```

2.2.3. Сборка контигов

Сборка контигов из квазиконтигов основана на подходе *overlap-layout-consensus* (*OLC*) [45] и выполняется с использованием графа перекрытий.

Подход *OLC* заключается в последовательном выполнении трех этапов: поиск перекрытий и построение графа перекрытий (*overlap stage*), выстраивание чтений (*layout stage*) и нахождение консенсуса для них (*consensus stage*). Схематично подход *OLC* можно изобразить так, как показано на рисунке 14.

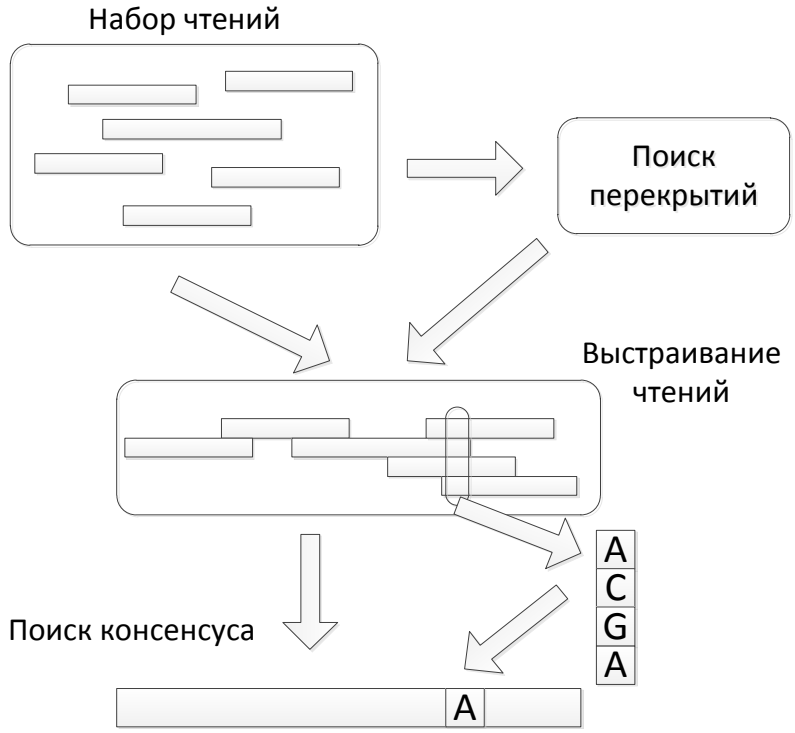


Рисунок 14 – Схематичное изображение стандартного подхода *OLC*

Предложенный метод сборки контигов из квазиконтигов основан на таком подходе, однако, в отличие от него, имеет несколько дополнительных шагов, а также все реализованные шаги изменены для работы при небольшом объеме доступной оперативной памяти.

Предложенный подход состоит из следующих этапов:

- поиск перекрытий между квазиконтигами (*overlap stage*);
- поиск и удаление покрываемых квазиконтигов;

- поиск ненайденных перекрытий с помощью найденных;
- удаление транзитивных перекрытий;
- построение графа перекрытий;
- упрощение графа перекрытий;
- поиск и вывод неветвящихся путей в графе (*layout stage*);
- нахождение консенсуса для путей (*consensus stage*).

Отметим, что перед сборкой контигов для каждого квазиконтига добавляется его обратно-комплементарная копия, что позволяет упростить алгоритм. При этом фактически копирование квазиконтигов не производится, а при загрузке чтений программно устанавливается, что каждый нечетный квазиконтиг является обратно-комплементарной копией квазиконтига, находящегося на предыдущей позиции.

Далее подробнее рассмотрим каждый из этапов.

2.2.3.1. Поиск перекрытий между квазиконтигами

На данном этапе необходимо найти перекрытия между всеми квазиконтигами C_1, C_2, …, C_n.

Для начала дадим определение перекрытию. Пусть имеется квазиконтиг A и квазиконтиг B. Перекрытием называется ситуация, при которой часть квазиконтига A совпадает с частью квазиконтига B при фиксированном их расположении относительно друг друга (рисунок 15).

При этом строка, по которой перекрываются квазиконтиги, не должна быть слишком маленькой (в противном случае перекрытий будет очень много). Минимальное значение длины, по которой могут перекрываться квазиконтиги, является параметром алгоритма (по умолчанию порог равен 40 нуклеотидам).

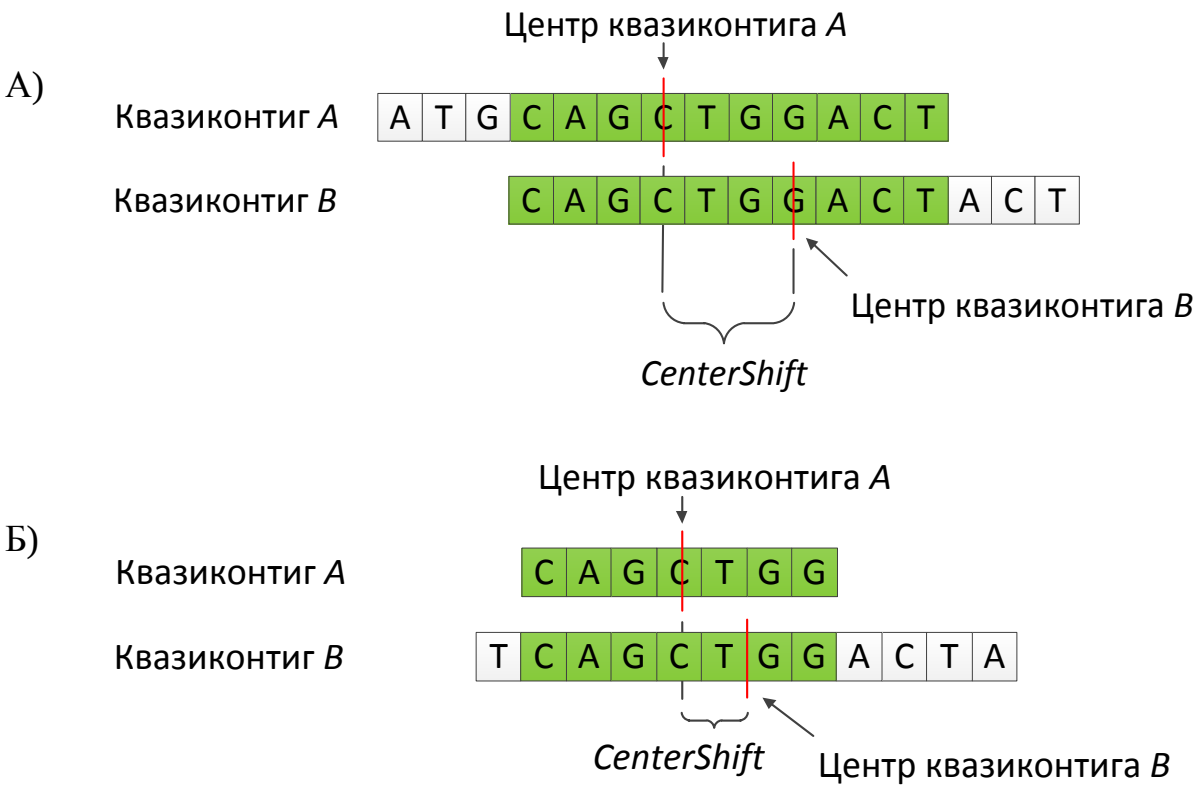


Рисунок 15 – Перекрытие двух квазиконтигов: А) случай частичного перекрытия; Б) случай покрытия одним квазиконтигом другого

Величина *CenterShift* – сдвиг между центрами двух перекрывающихся квазиконтигов, принимает только неотрицательные значения. Из-за этого любое перекрытие с ненулевым сдвигом однозначно задается тройкой: <номер первого квазиконтига (левее другого), номер второго квазиконтига, величина *CenterShift*>. Если центры квазиконтигов совпадают (*CenterShift* = 0), то перекрытие записывается так: <квазиконтиг с наименьшим номером, квазиконтиг с наибольшим номером, 0>. Также в последующих этапах используется величина сдвига между началами квазиконтигов *BeginShift*, которая легко может быть получена из величины *CenterShift*.

Теперь рассмотрим подробнее сам процесс поиска перекрытий.

Для эффективного поиска перекрытий квазиконтигов используется суффиксный массив строк, соответствующих квазиконтигам. *Суффиксным массивом* $a[i]$ называется лексикографически отсортированный массив

всех суффиксов строки. Значение $a[i]$ равно номеру суффикса (позиции начала суффикса в исходной строке), который идет i-тым в отсортированном списке всех суффиксов. Его построение происходит следующим образом. Сначала строится строка «$C_1\$C_2\$C_3\$...C_n\$$», где C_i – квазиконтиг с номером i. После чего создается массив всех суффиксов этой строки – массив $a[i]$, где при создании устанавливается $a[i]=i$ (суффикс с номером i первоначально расположен на i-той позиции). После этого производится сортировка суффиксов в массиве $a[i]$ в лексикографическом порядке.

С помощью построенного суффиксного массива можно быстро искать заданную подстроку во всех квазиконтигах. Это можно сделать, например, с помощью бинарного поиска в суффиксном массиве всех суффиксов, которые начинаются с заданной строки. Они будут располагаться рядом за счет сортировки.

Далее процесс поиска перекрытий выглядит просто. Для каждого квазиконтига B рассматриваются его префиксы в порядке увеличения длины, начиная с минимального порога перекрытия. Для того, чтобы некоторый квазиконтиг A частично перекрывался с квазиконтигом B по выбранному префиксу (не в случае полного покрытия одним квазиконтигом другого), необходимо, чтобы квазиконтиг A оканчивался на него. Таким образом, необходимо найти все суффиксы в суффиксном массиве, которые начинаются со строки «<зафиксированный префикс B>$» и понять, какой квазиконтиг завершается на этом месте. Таким способом можно найти все перекрытия, кроме случаев, когда один квазиконтиг полностью покрывает другой. Такой случай можно обработать отдельно: для каждого квазиконтига производится поиск суффиксов, начинающихся с него.

При таком подходе будут находиться только точные перекрытия – те перекрытия, при которых часть строки A полностью совпадает с частью строки B. Однако в квазиконтигах бывают ошибки, и имеет смысл искать

неточные перекрытия. Для этого необходимо после фиксирования префикса (или всего квазиконтига), по которому будут перекрываться квазиконтиги, изменить некоторое небольшое число нуклеотидов в нем. Понятно, что имеет смысл изменять только небольшое число символов префикса. Поэтому одним из параметров алгоритма данного этапа является максимальное число ошибок в перекрывающейся строке (по умолчанию равно двум на окно в 100 нуклеотидов).

Как и в случае с методом исправления ошибок, в случае нехватки оперативной памяти для выполнения всего этапа, происходит выполнение этапа «по частям». Для этого происходит разбиение всех суффиксов в суффиксном массиве и квазиконтигов по префиксу длины p, образуя 5^p групп (для алфавита из четырех нуклеотидов и знака «$»), и этап выполняется отдельно для каждой из них. При этом на такое выполнение требуется меньше памяти, чем для всего этапа в целом.

Приведем оценки сложности предложенного алгоритма. Он состоит из двух частей.

Первая часть – построение суффиксного массива. Разбиение неотсортированного массива на части происходит за $O(N \cdot p + 5^p)$, где N – общее число квазиконтигов, p – длина префикса, по которому разбиваем (является константой). Сортировка части суффиксного массива в среднем осуществляется за $O(n \cdot log_5 n)$, а в худшем случае – за $O(n \cdot L)$, где n – число квазиконтигов в рассматриваемой группе, а L – средняя длина квазиконтига.

Вторая часть с исправлением двух ошибок выполняется в худшем случае за $O(n \cdot C_L^2 \cdot log_5 n)$. Данная оценка верна для небольших L. Для больших же L эта оценка неверна из-за того, что в суффиксом массиве остается мало префиксов и для каждого из них число ошибок подсчитывается простым сравнением. Также применяется несколько дополнительных оптимизаций, которые делают процесс поиска перекрытий более быстрым.

2.2.3.2. Поиск и удаление покрываемых квазиконтигов

Все квазиконтиги, которые полностью покрываются более длинными квазиконтигами, не используются при дальнейшей работе предложенного сборщика. Это связанно с тем, что они не добавляют никакой новой информации (при условии существования покрывающего его квазиконтига), а, напротив, усложняют все последующие этапы. При отсутствии полностью покрываемых квазиконтигов последующие шаги становятся проще, так как не надо отдельно обрабатывать случаи с их существованием.

Определение покрываемых квазиконтигов производится сравнительно просто по результатам вычисления всех перекрытий. Для этого необходимо по каждому перекрытию определить, является ли один из квазиконтигов, участвующий в перекрытии, покрываемым другим квазиконтигом (путем сравнения длин квазиконтигов и сдвига при перекрытии).

Вычислительная сложность этого этапа – $O(N + E)$, где E – общее число найденных перекрытий.

2.2.3.3. Поиск ненайденных перекрытий с помощью найденных

После выполнения предыдущих этапов возможны ситуации, когда некоторые перекрытия не будут найдены. Это может случиться, например, при условии ошибок в префиксе, по которому были разбиты квазиконтиги и суффиксный массив, или если число ошибок в перекрывающейся части не на много больше исправляемого числа ошибок. Поэтому необходимо запустить поиск ненайденных перекрытий с помощью найденных.

Для этого для каждого квазиконтига используются все квазиконтиги, перекрывающиеся с ним, после этого производится поиск новых перекрытий среди них. Ввиду того, что квазиконтигов, между которыми производится поиск перекрытий, теперь намного меньше, чем в предыдущем этапе, используется другой алгоритм поиска.

Рассмотрим этот алгоритм. Предположим, что некоторый квазиконтиг *A* выбран. Выпишем все квазиконтиги, из которых есть ребро в квазиконтиг *A*. Тогда возможно несколько ситуаций, некоторые из которых показаны на рисунке 16.

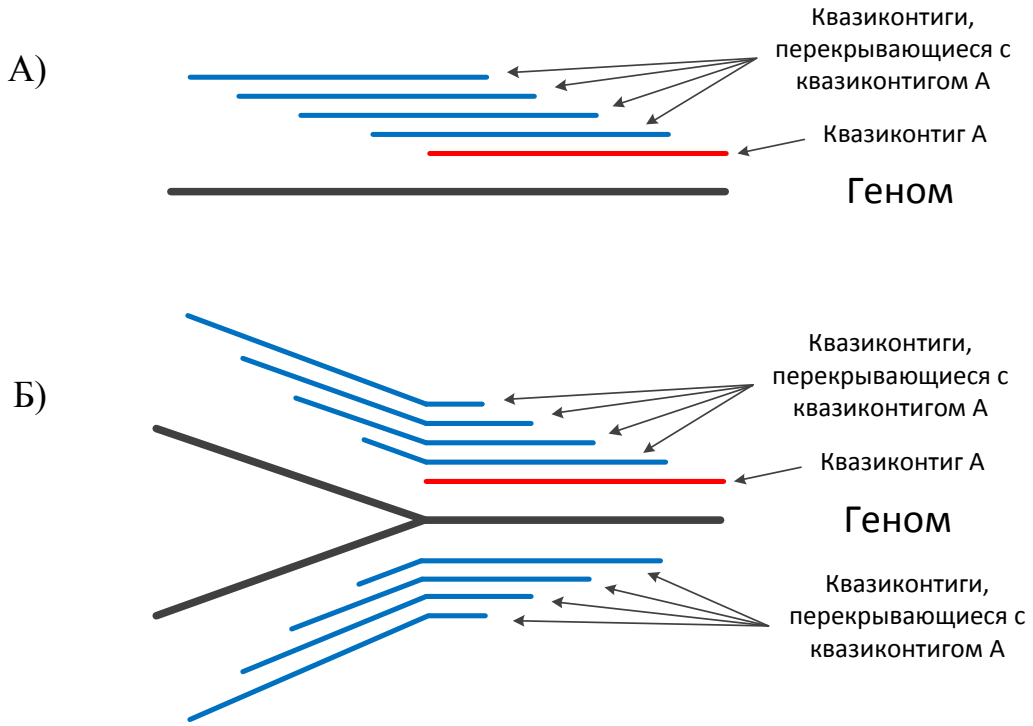


Рисунок 16 – Схематичное изображение возможных перекрытий квазиконтигов: А) случай части генома без развилок; Б) случай части генома с развилкой (повторяющимся фрагментом)

Алгоритм делает следующее. После выбора всех квазиконтигов, из которых есть ребро в выбранный квазиконтиг *A*, производится их сортировка по сдвигу относительно квазиконтига *A*. После этого выполняется последовательная обработка квазиконтигов, начиная с самого левого и кончая самым правым. По мере обработки, сохраняются последние квазиконтиги из разных веток – квазиконтиги, которые не перекрываются между собой. Обработка состоит в том, что обрабатываемый квазиконтиг проверяется на возможность перекрытия с последними квазиконтигами из разных веток. Если такое перекрытие найдено, то считается, что обрабатываемый квазиконтиг должен

находиться в той же ветке, что и квазиконтиг, с которым он перекрывается. Добавляется новое ребро между этими квазиконтигами, если оно ещё не существовало. Если же не получается найти перекрытие с существующими квазиконтигами разных веток, то добавляется новая ветка, а обрабатываемый квазиконтиг сохраняется как последний квазиконтиг новой ветки.

В этом алгоритме не производится поиск перекрытий между квазиконтигами по разную сторону от квазиконтига A из-за того, что в транзитивных перекрытиях нет необходимости.

Приведем оценки сложности предложенного алгоритма. Алгоритм в среднем выполняется за $O(N \cdot e \cdot \log e + N \cdot e \cdot L)$, в худшем случае – за $O(N \cdot e^2 \cdot L)$, где e – среднее число перекрытий на один квазиконтиг, L – средняя длина квазиконтига.

2.2.3.4. Удаление транзитивных перекрытий

После нахождения всех перекрытий обычно оказывается, что их достаточно много (для реальных данных каждый квазиконтиг имеет в среднем 20–80 перекрытий). Поэтому для уменьшения их числа сначала выполняется поиск и удаление транзитивных перекрытий.

Перекрытие <*A, C, CenterShift*$_{AC}$> между квазиконтигами *A* и *C* будем называть *транзитивным*, если существуют перекрытия <*A, B, CenterShift*$_{AB}$> и <*B, C, CenterShift*$_{BC}$> такие, что:

- *CenterShift*$_{AC}$ = *CenterShift*$_{AB}$ + *CenterShift*$_{BC}$;
- *CenterShift*$_{AB}$ < *CenterShift*$_{AC}$;
- *CenterShift*$_{BC}$ < *CenterShift*$_{AC}$.

Пример транзитивного перекрытия <*A, C, CenterShift*$_{AC}$> изображен на рисунке 17.

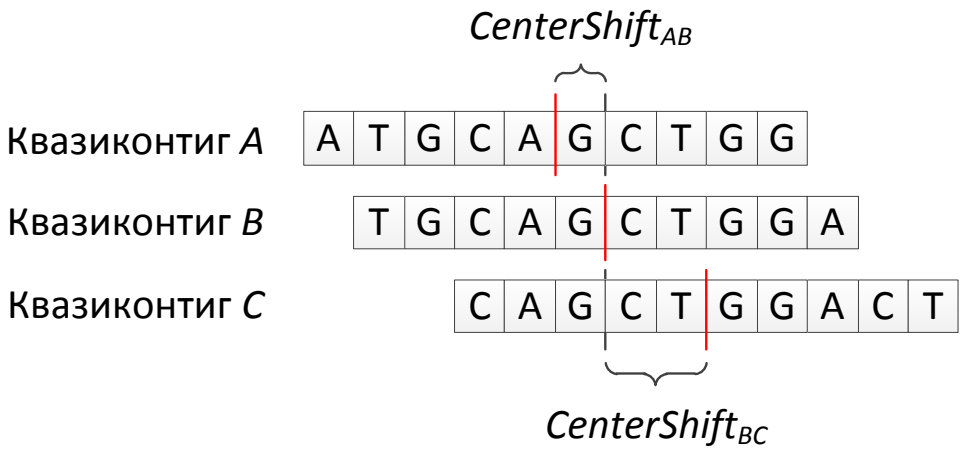


Рисунок 17 – Транзитивное перекрытие

Заметим, что в транзитивных перекрытиях нет никакой необходимости – мы всегда можем их получить, используя меньшие по сдвигу перекрытия. Более того, транзитивные перекрытия увеличивают сложность алгоритмов на следующих шагах. Поэтому желательно удалить все такие перекрытия.

Поиск транзитивных перекрытий осуществляется следующим образом. Сначала для каждого квазиконтига A все его перекрытия упорядочиваются по сдвигу. После этого перебираются два квазиконтига B и C, которые перекрываются с ним, и проверяется существование перекрытия непосредственно между ними. В случае его существования, перекрытие AB или AC с самым длинным сдвигом удаляется.

Вычислительная сложность этапа – $O(N \cdot e \cdot \log e)$, где e – среднее число перекрытий на один квазиконтиг.

2.2.3.5. Построение графа перекрытий и его упрощение

Напомним, что *графом перекрытий* называется граф, вершины которого – квазиконтиги, а две вершины соединяются ребром, если соответствующие квазиконтиги перекрываются. При этом на ребре пишется центральный сдвиг перекрытия и вес ребра. Пример графа перекрытий изображен на рисунке 18.

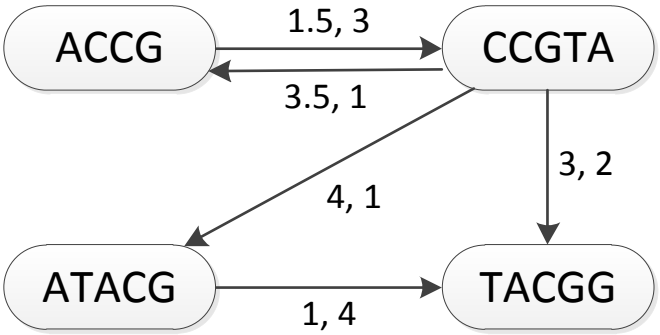


Рисунок 18 – Граф перекрытий

Вес ребра является числовой характеристикой надежности ребра. Чем она больше, тем больше вероятность того, что данное перекрытие действительно присутствует в геноме, а не образовалось в результате совпадения небольшой части двух квазиконтигов. Вес рассчитывается по длине перекрывающейся части, ввиду того, что чем больше перекрывающаяся часть, тем больше уверенности в том, что это не случайны повтор.

Граф перекрытий довольно легко построить, зная все перекрытия между квазиконтигами. Он удобен для последующей работы над перекрытиями и квазиконтигами благодаря наглядному отображению ситуации.

Для лучшего понимания далее будем писать на ребрах графа только длину соответствующего перекрытия. Пример такого графа изображен на рисунке 19.

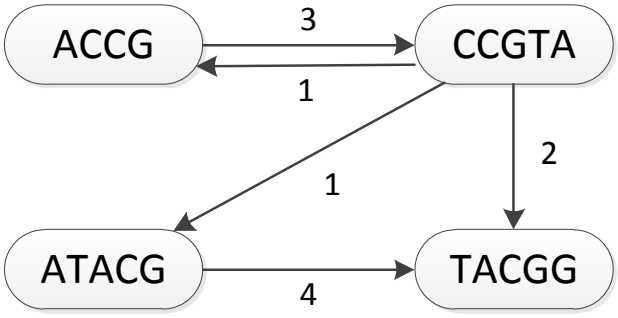


Рисунок 19 – Граф перекрытий со значениями длины перекрытий на ребрах графа

После построения графа перекрытий обычно оказывается, что он сильно запутан. Такая запутанность вызвана как ошибками в исходных данных, так и сложной структурой генома с большим числом повторов и полиморфизмов. При этом на следующем этапе происходит поиск и вывод путей без ветвлений (*layout stage*), и необходимо иметь максимально простой (желательно линейный) граф. Для этого применяются шаги по упрощению графа перекрытий: объединение схожих путей, удаление «отростков» (*tips*) в графе, анализ развилок и их упрощение.

При объединении схожих путей рассматриваются два пути, которые имеют одинаковую длину, а также одну и ту же начальную и конечную вершину. При условии схожести нуклеотидного состава образующих их квазиконтигов (консенсусы квазиконтигов в пути должны отличаться в небольшом числе нуклеотидов), происходит объединение рассматриваемых путей. Сам процесс объединения происходит путем выстраивания всех вершин в двух путях в одну цепочку, учитывая их сдвиги относительно соседних вершин. Простой пример объединения путей изображен на рисунке 20.

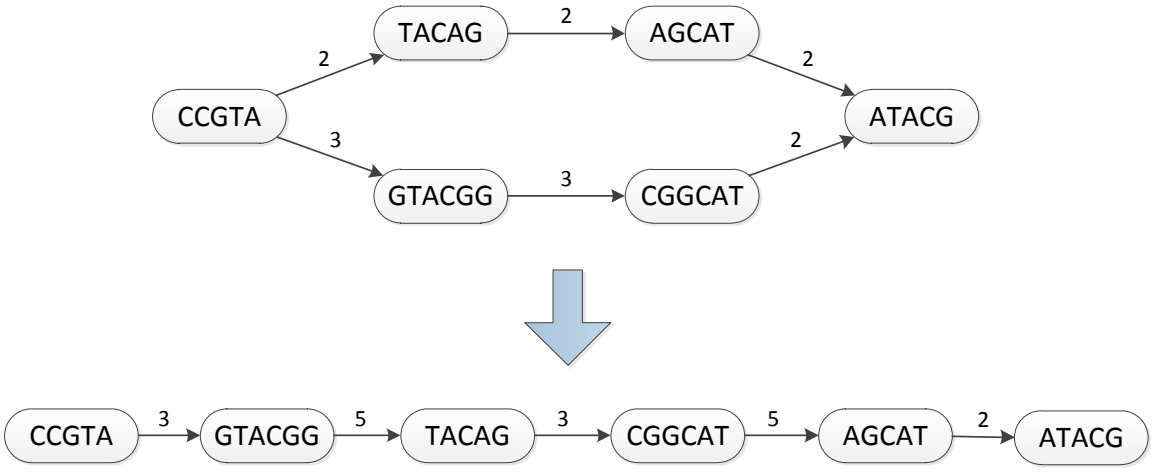


Рисунок 20 – Объединение путей

Отростком (*tip*) в графе перекрытий называется небольшая часть графа, обычно ветвящаяся и не имеющая исходящих ребер. При этом важно не удалять значимые части графа, поэтому отростками признаются

только те части, которые имеют небольшую максимальную глубину (по умолчанию не большее пяти) и при условии наличия альтернативной части с большей глубиной (отношение глубин должно быть не меньше двух). Такие отростки удаляются из графа из-за того, что они, скорее всего, возникли в результате ошибки и мешают продолжению контига. Пример такого упрощения показан на рисунке 21.

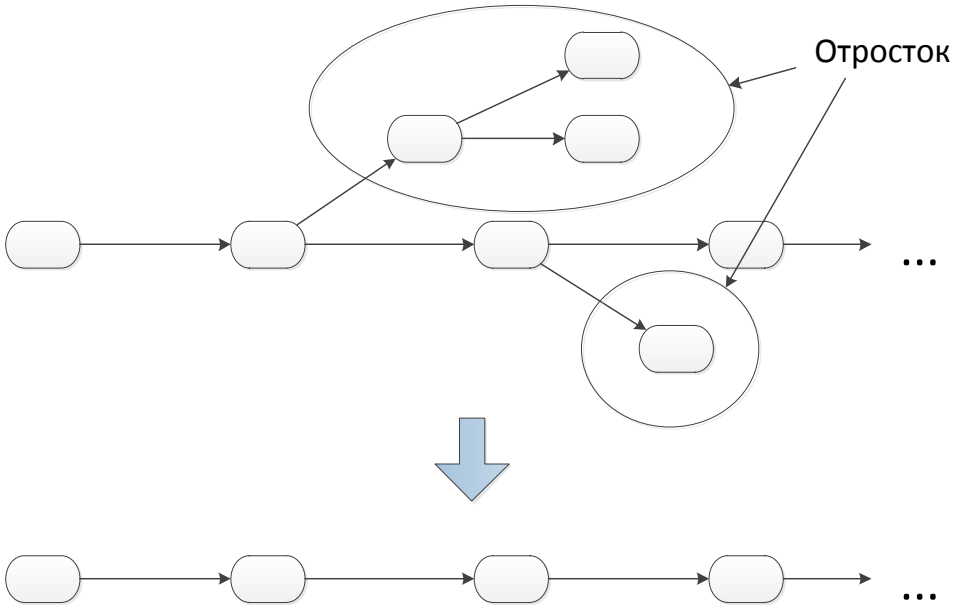


Рисунок 21 – Удаление отростков

На этапе анализа развилок происходит поиск вершин в графе, имеющих несколько исходящих или входящих ребер. Примеры возможных развилок приведены на рисунке 22.

Ситуация для каждой вершины анализируется отдельно. В случае наличия ребер с небольшим весом, они исключаются из рассмотрения ситуации. Если при этом остается больше одного ребра (рассматривая только входящие или исходящие), из них выбирается одно ребро с наибольшим весом и помечаются как наиболее правдоподобное. После рассмотрения всех вершин непомеченные ребра у вершин с несколькими входящими или исходящими ребрами удаляются. Такой процесс помогает избавиться от неправдоподобных ребер в графе перекрытий.

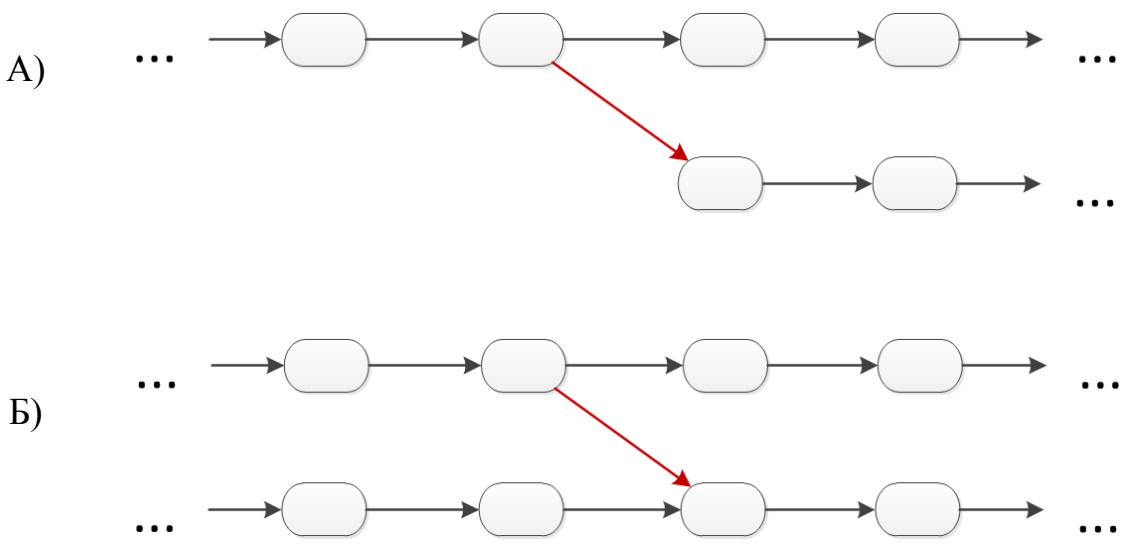


Рисунок 22 – Схематичное изображение возможных развилок в графе перекрытий А) случай появления дополнительного пути; Б) случай двух путей с ребрами между ними

Вычислительная сложность каждого из этапов упрощения графа перекрытий – $O(N + E)$, где N – число вершин в графе, E – число ребер.

2.2.3.6. Поиск и вывод путей в графе

На данном этапе выводятся непересекающиеся пути с вершинами, имеющими входящую и исходящую степени равными единице (пути без ветвлений). Такой путь сохраняется в виде последовательности номеров квазиконтигов и смещений между двумя соседними квазиконтигами. На следующем этапе каждый из таких путей преобразуется в контиг путем нахождения консенсуса для квазиконтигов, участвующих в пути.

Отметим, что вершины, имеющие входящую или исходящую степень больше единицы, участвуют только в качестве начальных или конечных вершин пути. Это эквивалентно разбиению всего графа на простые пути с помощью удаления всех имеющихся развилок.

Данный этап выполняется за $O(N + E)$ действий.

2.2.3.7. Нахождение консенсуса для путей

На данном этапе рассматриваются все пути, которые записаны в виде последовательности номеров квазиконтигов, и выводится одна последовательность нуклеотидов для каждого из них – консенсус всех перечисленных квазиконтигов. Полученная нуклеотидная последовательность считается результирующим контигом.

Нахождение консенсуса происходит путем выбора наиболее часто встречаемого нуклеотида в квазиконтигах отдельно для каждой позиции. Схематично процесс можно представить так, как показано на рисунке 23.

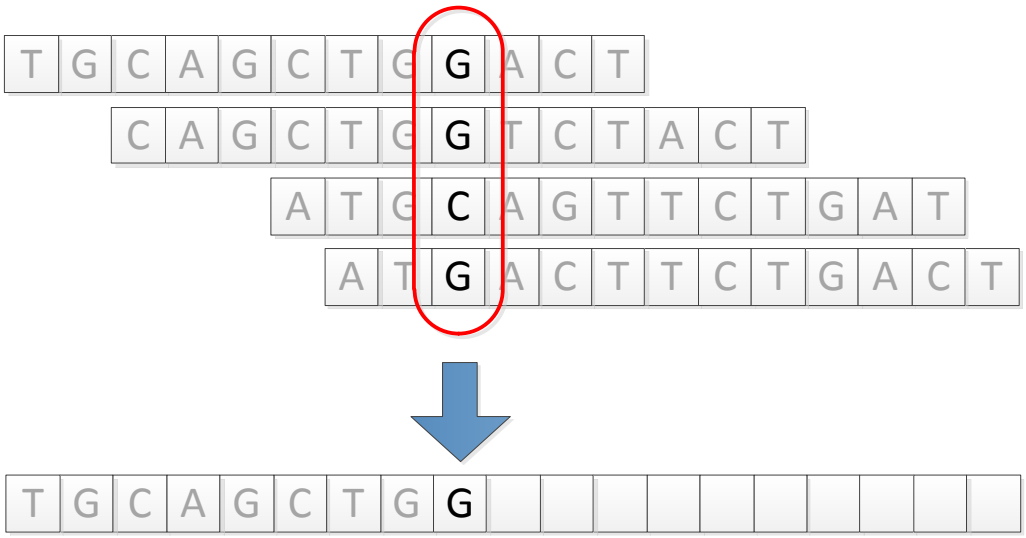


Рисунок 23 – Нахождение консенсуса

Вычислительная сложность этапа – $O(N \cdot L)$, где N – общее число квазиконтигов во всех рассматриваемых путях, а L – средняя длина квазиконтига.

2.3. Реализация предложенных подходов

Предложенные методы сборки генома были реализованы на языке программирования *Java*. Благодаря этому разработанная программа является кроссплатформенной и может быть запущена на операционных системах *Windows*, *maxOS/OS X* и *Linux*. Для запуска необходима установленная *Java* версии 1.6 или выше.

Предложенный метод *de novo* сборки генома и все необходимые для его работы структуры данных были реализованы в виде библиотеки *itmo-assembler.jar*, которая свободно доступна вместе с исходным кодом в сети Интернет по адресу `http://genome.ifmo.ru/ru/assembler`.

Разработанная программа имеет графический интерфейс пользователя (рисунок 24). С его помощью можно запускать процесс сборки генома, изменять параметры сборки и следить за ходом её выполнения. Благодаря графическому интерфейсу, сборку могут производить и люди, которые плохо владеют работой с командой строкой.

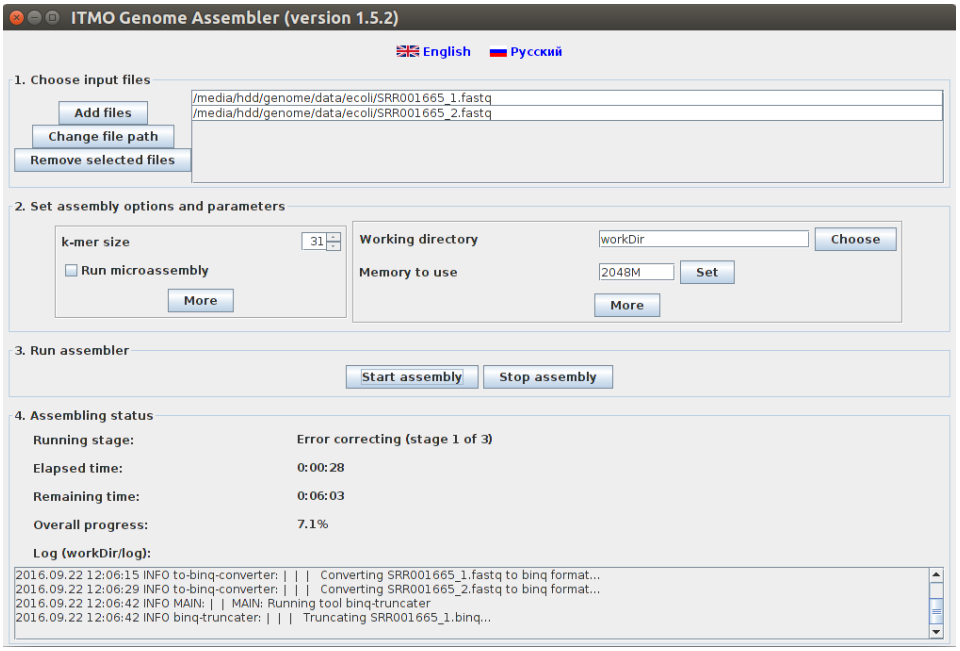


Рисунок 24 – Графический интерфейс разработанной программы

Программа также имеет возможность простого запуска процесса сборки генома, указав только исходные файлы (доступно как из командной строки, так и из графического интерфейса). Для этого были выбраны значения по умолчанию для многих параметров сборки, при этом некоторые из них определяются автоматически во время работы программы. Также остается возможность ручной настройки параметров – любой из них можно менять в пределах разрешенных значений.

Реализация также имеет возможность задавать используемый объем памяти компьютера и число ядер процессора.

Автоматизация сборки генома состоит в том, что все этапы и подэтапы, необходимые для проведения сборки, выполняются автоматически (как единый процесс). При этом программа имеет модульную архитектуру, что позволяет независимо изменять отдельные модули, не меняя и не перестраивая остальные части. Таким образом, запуск всех этапов сборки осуществляется одной командой, однако также остается возможность запуска модулей по отдельности.

2.4. Экспериментальное исследование

Для оценки эффективности разработанной программы было выполнено экспериментальное исследование, целью которого было сравнение разработанной программы сборки генома с существующими.

2.4.1. Использованные наборы данных

Для сравнения программ сборки генома были использованы три набора данных, свободно доступных в сети Интернет. Информация о них представлена в таблице 6.

Таблица 6 – Используемые наборы данных

№	Название организма	Платформа секвениро-вания	Размер генома, млн нукл.	Дополнительная информация о наборе
1.	*Escherichia coli* (Кишечная палочка)	*Illumina Genome Analyzer*	4,6	Парные чтения, размер фрагмента – 200 нукл., длина чтения – 36 нукл. Покрытие исходного генома – 161,5x. Размер исходных данных – 4,0 Гб.

2.	*Escherichia coli* (Кишечная палочка)	*Illumina Genome Analyzer* II	4,6	Парные чтения, размер фрагмента – 215 нукл., длина чтения – 100 нукл. Покрытие исходного генома – 492х. Размер исходных данных – 5,0 Гб.
3.	*Human chromosome* 14 (14-ая хромосома человека)	*Illumina HiSeq* 2000	107,3	Парные чтения, размер фрагмента – 155 нукл., длина чтения – 101 нукл. Покрытие исходного генома – 34,3х. Размер исходных данных – 7,8 Гб.

Для каждого из предложенных наборов были скачены исходные файлы с парными чтениями, референсная последовательность и файл с известными генами в референсе. Последние два файла использовались для вычисления дополнительных характеристик при сравнении результатов сборок. Сами сравниваемые сборщики использовали при своей работе только исходные файлы с чтениями.

2.4.2. Методология экспериментов

Сравнение работы разработанной программы сборки генома производилось со следующими сборщиками: *SPAdes*, *MaSuRCA*, *Velvet*, *Newbler*, *Minia* и *SparseAssembler*. Сборщики *SPAdes*, *Velvet*, *Minia* и *SparseAssembler* основаны на графе де Брейна, сборщик *Newbler* использует только граф перекрытий, а сборщик *MaSuRCA* – оба графа при работе.

Сравнение работы программ производилось следующим образом. Каждый сборщик запускался на каждом наборе данных. При этом использовались параметры сборки, установленные по умолчанию, либо они задавались согласно руководству по запуску сборщика для тех из них, которые требовали ручного задания параметров. В случае если программа имела возможность указывать объем памяти для использования, она последовательно (в разных экспериментах) запускалась на наборе данных с возможностью использования всей памяти в 16 Гб, 4 Гб памяти, 3 Гб памяти, 2 Гб, 1 Гб и 0.5 Гб. В случае успешности запуска (программа смогла отработать и получить результат), информация об этом запуске заносилась в таблицу сравнения.

Тестирование сборщиков проводилась на компьютере с 16 Гб оперативной памяти и шестиядерным процессором *AMD PhenomTM II X6 1090T* под управлением *OS Linux 3.13.0 x86_64*. Эта операционная система использовалась для проведения экспериментов, так как большая часть сборщиков не могут работать под другими операционными системами. Всем сборщикам предлагалось использовать шесть ядер процессора для работы.

Для каждого эксперимента производилось измерение вычислительных ресурсов, необходимых для работы, – затраченного времени и максимального объема оперативной памяти во время работы. Все измерения производительности производились стандартными средствами операционной системы *Linux*. В случае успешности запуска, получаемая сборка (итоговые контиги или скэффолды, в случае их наличия) оценивалась программой *QUAST* 3.1 [59] с использованием референсной последовательности и файла с известными генами. Наиболее значимые характеристики сборки, вычисленные *QUAST*, записывались в таблицу сравнения и использовались для последующего анализа.

2.4.3. Результаты экспериментов

Результаты запусков сборщиков на наборе данных № 1 (*Escherichia coli*, размер генома – 4,6 млн нукл.) представлены в таблице 7. Лучшие значения для каждого из столбцов выделены жирным шрифтом. Сборщик *Newbler* не смог произвести сборку данном наборе данных, так как предоставляемые чтения были слишком короткими для него.

Таблица 7 – Результаты запусков для первого набора данных (*Escherichia coli*, размер генома – 4,6 млн нукл., число известных генов – 4518)

Сборщик	Использованная память, Гб	Время работы, мин:сек	Число контигов/ скэффолдов	N50, тыс. нукл.	Процент собранного генома	Собрано генов
SPAdes	2.90	19:35	127	82.4	98.0	**4378 + 36 part**
MaSuRCA	3.75	11:08	166	54.3	97.9	4355 + 91 part
Velvet	2.13	7:32	**110**	**95.4**	97.6	4360 + 39 part
Minia	1.21	**1:18**	461	16.3	97.2	4163 + 172 part
Sparse-Assembler	**0.15**	3:51	561	12.7	96.6	4098 + 217 part
ITMO Assembler (2 Гб)	2.30	24:11	246	37.1	**98.2**	4350 + 116 part
ITMO Assembler (1 Гб)	1.17	12:11	247	35.8	**98.1**	4347 + 120 part
ITMO Assembler (0.5 Гб)	0.60	6:40	270	32.3	**98.1**	4344 + 119 part

Результаты запусков сборщиков на наборе данных № 2 (*Escherichia coli*, размер генома – 4,6 млн нукл.) представлены в таблице 8. На данном наборе данных все сборщики смогли произвести сборку при ограничении памяти в 16 Гб, однако сборщики *MaSuRCA*, *Velvet* и *Newbler* потребовали на это больше 10 Гб памяти. Сборщик *Newbler* к тому же выполнял сборку более 12 часов, что на порядок больше, чем время работы остальных программ. Сборщики *Velvet* и *SparseAssember* получили относительно плохую итоговую сборку, которая покрывает не более 60% исходного генома.

Таблица 8 – Результаты запусков для второго набора данных (*Escherichia coli*, размер генома – 4,6 млн нукл., число известных генов – 4518)

Сборщик	Использованная память, Гб	Время работы, ч:мин	Число контигов/ скэффолдов	N50, тыс. нукл.	Процент собранного генома	Собрано генов
SPAdes	3.90	0:59	**92**	**133.3**	98.2	**4395 + 32 part**
MaSuRCA	14.91	2:28	798	10.2	**98.3**	3955 + 520 part
Velvet	11.62	0:39	2939	0.9	55.6	804 + 2590 part
Newbler	12.81	12:14	**91**	89.2	97.0	4306 + 48 part
Minia	4.24	**0:08**	531	13.8	96.7	4085 + 239 part
Sparse-Assembler	**0.55**	0:29	76	0.6	1.0	11 + 88 part
ITMO Assembler (4 Гб)	4.57	1:05	156	80.7	**98.4**	**4390 + 97 part**
ITMO Assembler (2 Гб)	2.33	0:34	149	82.5	**98.4**	**4388 + 96 part**

Результаты запусков сборщиков на наборе данных № 3 представлены в таблице 9. Сборщики *SPAdes*, *MaSuRCA*, *Velvet* и *Newbler* не смогли произвести сборку из-за нехватки оперативной памяти в 16 Гб.

Таблица 9 – Результаты запусков для третьего набора данных (14-ая хромосома человека, размер генома – 107,3 млн нукл., число генов – 2244)

Сборщик	Использованная память, Гб	Время работы, ч:мин	Число контигов/ скэффолдов	N50, тыс. нукл.	Процент собранного генома	Собрано генов
Minia	4.11	**0:26**	37017	2.3	60.3	682 + 1164 part
Sparse-Assembler	**2.19**	0:55	37126	2.3	60.4	671 + 1179 part
ITMO Assembler (4 Гб)	4.63	1:51	37012	**2.9**	**71.9**	**886 + 1184 part**
ITMO Assembler (3.3 Гб)	3.85	1:40	41371	2.4	**70.2**	845 + 1211 part

Из представленных таблиц можно сделать следующие выводы:

- По числу найденных генов качественнее всего собирает *SPAdes* (для бактериальных геномов). На втором месте – *ITMO Genome Assembler* (разница в числе найденных генов по сравнению со *SPAdes* – от **0.1%** до **0.6%**). При этом *ITMO Genome Assembler* является лидером по числу собранных генов для среднего по размеру генома (14-ой хромосомы человека).
- Сборщик *ITMO Genome Assembler* собирает лучше остальных по проценту покрытия исходного генома. На втором месте – *MaSuRCA* и *SPAdes* (разница – от **0.1%** до **0.3%**, для бактериальных геномов).
- Сборщик *SPAdes* собирает лучше остальных по значению параметра N50 (минимальной длине контига для покрытия 50% сборки, для

бактериальных геномов). Сборщик *ITMO Genome Assembler* собирает лучше других по этому показателю для среднего по размеру генома.

- Сборщик *Minia* работает быстрее остальных сборщиков, однако показатели качества сборки (N50, число найденных генов) остаются низкими.
- Сборщик *SparseAssembler* использует меньше всех оперативной памяти, однако показатели качества сборки (N50, число найденных генов) остаются низкими. На втором месте – *ITMO Genome Assembler*.
- Сборщики *Velvet*, *Masurca* и *Newbler* могут потребовать от двух до 15 Гб оперативной памяти при сборке бактериального генома в 4,6 млн. нукл.
- Не один из известных сборщиков *SPAdes*, *Velvet*, *MaSuRCA* и *Newbler* не может произвести сборку среднего по размеру генома при ограничении памяти в 16 Гб.
- Сборщик *ITMO Genome Assembler* показал лучше результаты по многим параметрам сравнения при сборке среднего по размеру генома (14-ой хромосомы человека).

Таким образом, предлагаемое решение *ITMO Genome Assembler* по объему используемой памяти и времени работы уступает только сборщикам *Minia* и *SparseAssembler*, которые, однако, получают результаты относительно низкого качества (по числу найденных генов в сборке, параметру N50, проценту покрытия исходного генома).

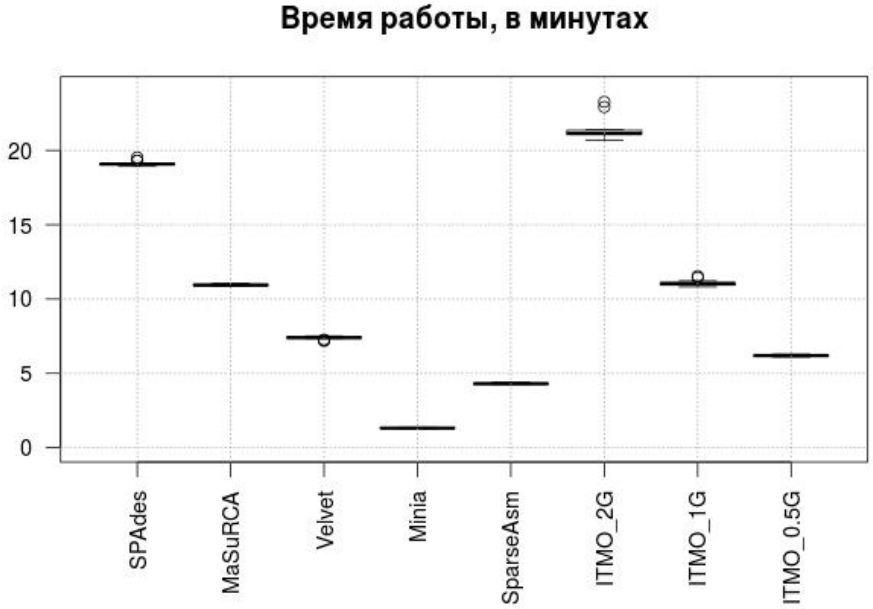
2.4.4. Дополнительные эксперименты по анализу требуемых вычислительных ресурсов

Для подтверждения времени работы и требуемой памяти у сравниваемых сборщиков были произведены многократные подтверждающие запуски. По результатам измерений были построенные

ящичные диаграммы распределений запусков сборщиков по времени работы и объему используемой памяти.

Ящичные диаграммы распределений запусков на наборе данных № 1 показаны на рисунке 25.

А)



Б)

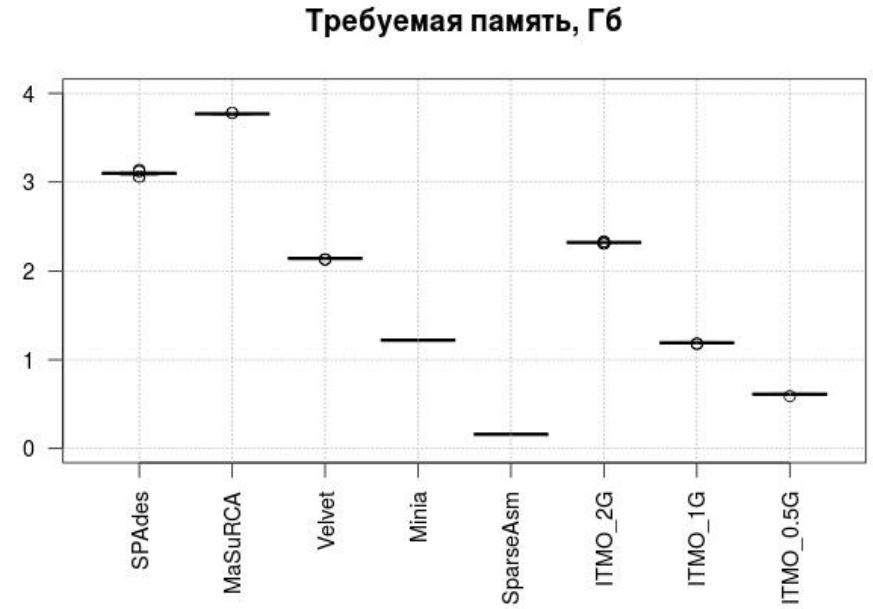
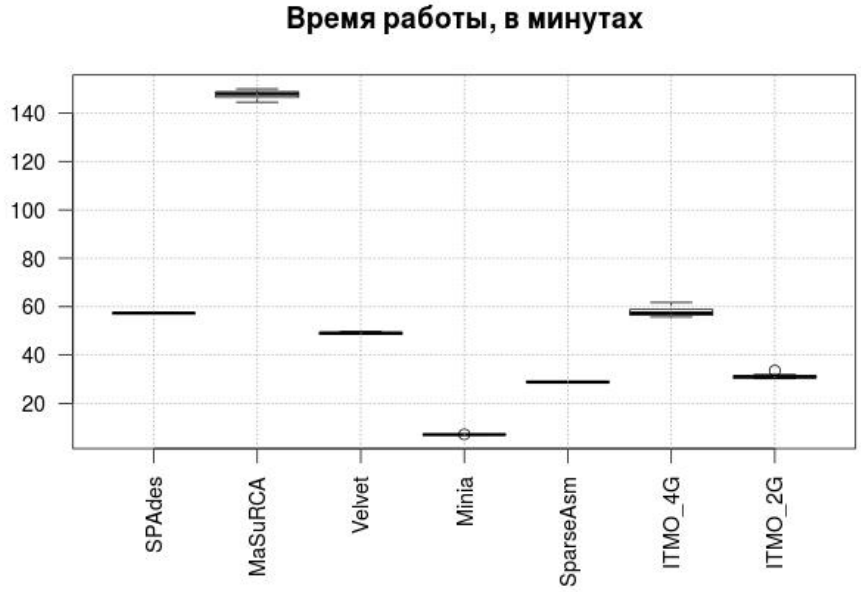


Рисунок 25 – Ящичные диаграммы распределений А) времени работы; Б) требуемой памяти при запуске сборщиков на наборе данных № 1

Ящичные диаграммы распределений запусков на наборе данных № 2 показаны на рисунке 26.

А)



Б)

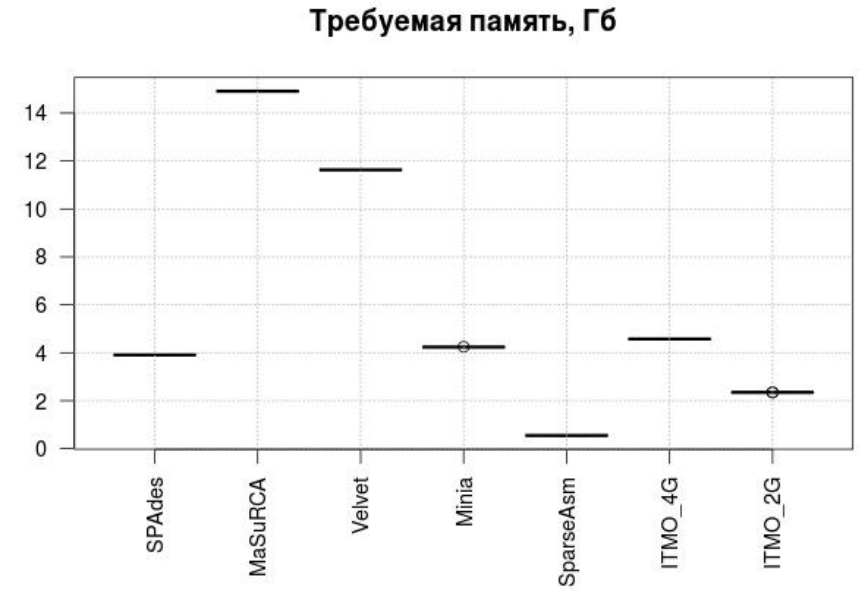
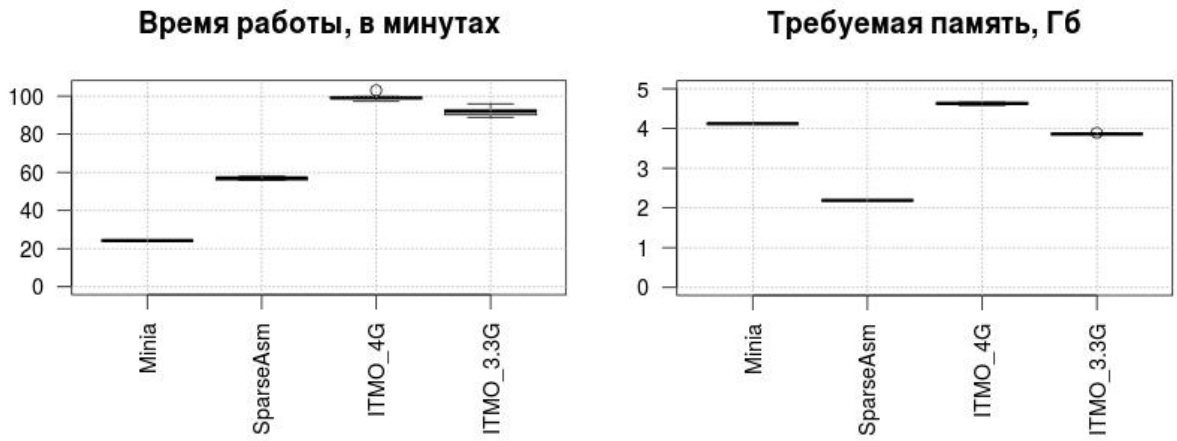


Рисунок 26 – Ящичные диаграммы распределений А) времени работы; Б) требуемой памяти при запуске сборщиков на наборе данных № 2

Ящичные диаграммы распределений запусков на наборе данных № 3 показаны на рисунке 27.



А) Б)

Рисунок 27 – Ящичные диаграммы распределений А) времени работы; Б) требуемой памяти при запуске сборщиков на наборе данных № 3

Из представленных графиков можно сделать следующие выводы:

- Сборщик *SparseAssembler* на всех трех наборах данных использует меньше памяти, чем предлагаемое решение *ITMO Genome Assembler*. Остальные сборщики используют больше памяти, чем предлагаемое решение.
- Сборщики *Minia* и *SparseAssembler* являются единственными, кто работает быстрее предлагаемого решения *ITMO Genome Assembler* на всех трех набора данных. Остальные сборщики работают дольше.

Выводы по главе 2

1. Разработан автоматизированный метод сборки генома de novo на основе совместного применения графа де Брейна и графа перекрытий.
2. Было проведено экспериментальное исследование предложенного метода и существующих решений на трех наборах данных.
3. Экспериментальное исследование продемонстрировало эффективность и применимость предложенного подхода.
4. Результаты и методы, описанные в данной главе, были опубликованы в статьях соискателя [101–104, 106–108, 110, 111, 114–119], отчетах по государственным контрактам [1, 98, 99] и в магистерской диссертации автора [2].

ГЛАВА 3. АВТОМАТИЗИРОВАННЫЙ МЕТОД СРАВНИТЕЛЬНОГО АНАЛИЗА МЕТАГЕНОМОВ, ОСНОВАННЫЙ НА АНАЛИЗЕ КОМПОНЕНТ СВЯЗНОСТИ В ГРАФЕ ДЕ БРЕЙНА

Настоящая глава посвящена разработанному методу сравнительного анализа метагеномов.

В ней приводится описание предлагаемого метода, даются оценки на время работы этапов метода, описывается реализация предложенных подходов, а также приводятся результаты экспериментального исследования, целью которого было сравнение предложенного метода с существующими.

Результаты и методы, описанные в данной главе, были также опубликованы в работах [101, 105, 109, 112, 113].

3.1. Метод сравнительного анализа метагеномов

Предлагаемый подход состоит в выполнении «упрощенной» сборки для каждого метагенома отдельно, объединением полученных последовательностей в один граф де Брейна и анализом компонент связности в нем. Используемая «упрощенная» сборка частично основана на описанном в граве 2 методе *de novo* сборки генома.

Предлагаемый подход состоит из следующих этапов.

1. **Подсчет** и сохранение **надежных *k*-меров** для каждого метагенома отдельно.
2. **Выполнение «упрощенной» сборки** из чтений для каждого метагенома отдельно (выполняется на графе де Брейна). Получаемые последовательности (также называемые *унитигами*, *unitigs* [79]), являются аналогами контигов, однако являются в разы короче их.
3. **Объединение полученных последовательностей** от всех метагеномов в один граф де Брейна **и выделение компонент связности** в нем. Большие компоненты при этом разбиваются на части, используя итеративный метод выделения основной части.

Каждая полученная компонента применяется далее как единичный признак.

4. **Вычисление характеристического вектора** для каждого метагенома. Для каждой компоненты подсчитывается сколько раз *k*-меры, которые образуют компоненту, присутствуют в чтениях рассматриваемого метагенома. Вычисленные значения для всех компонент образуют характеристический вектор.
5. **Вычисление попарного расстояния** между метагеномами на основе полученных характеристических векторов (с использованием индекса Брея-Кертиса). Значения записывают в так называемую «матрицу расстояния».
6. **Выполнение кластеризации образцов** по полученной матрице расстояний **и построение графических результатов**: тепловой карты (*heatmap*) сходства исследуемых метагеномов и дендрограммы.

Схематичное изображение выполняемых шагов приведено на рисунке 28. Для лучшего понимания ситуации чтения от разных бактерий изображены разным цветом.

Метод принимает на вход чтения в формате *FASTQ* [13, 92], однако также может работать и с чтениями в формате *FASTA* [91]. Главные результаты работы метода – выделяемые признаки (*features*) и матрица расстояний между метагеномами.

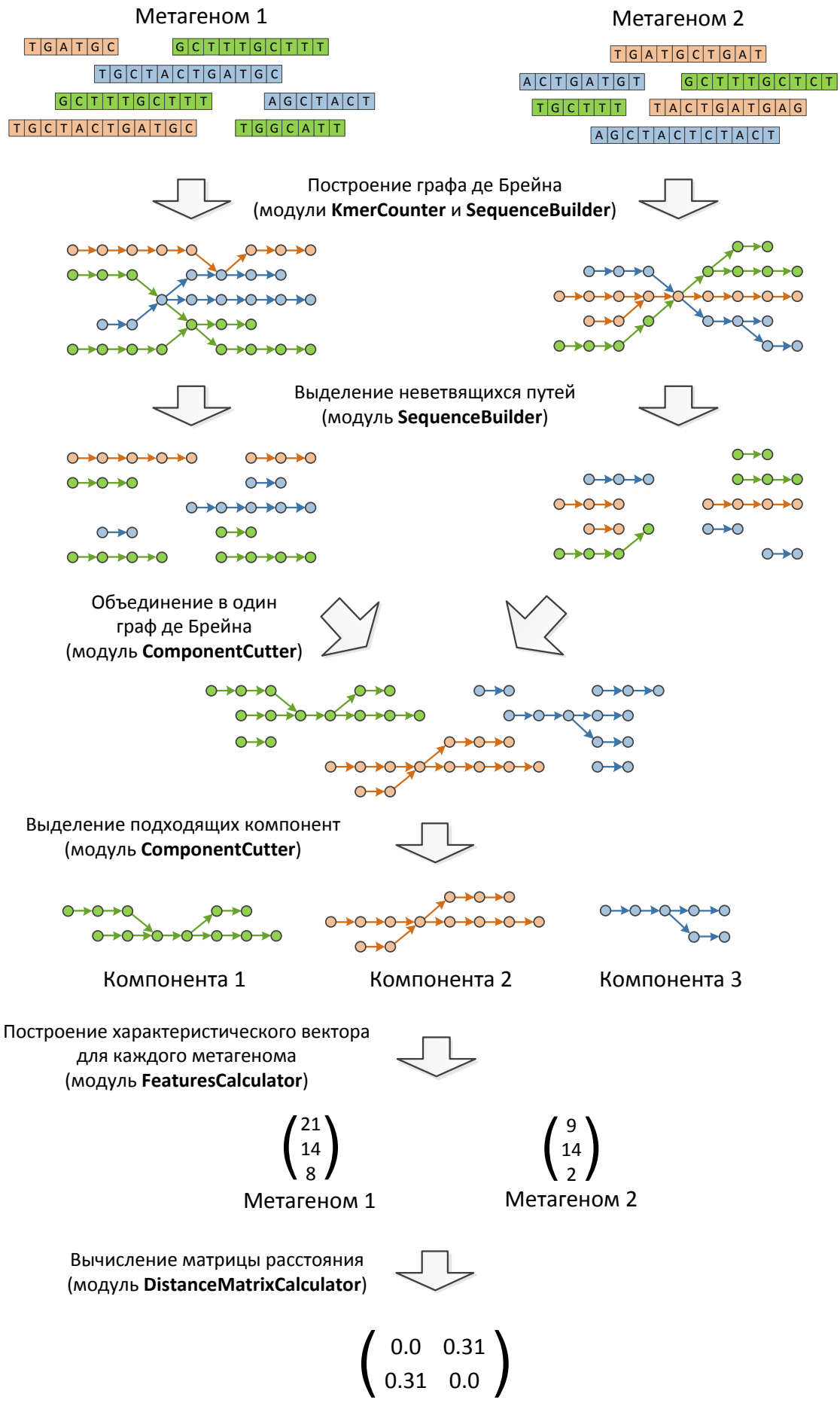


Рисунок 28 – Основные шаги работы алгоритма *MetaFast*

Отличительной особенностью предлагаемого решения является то, что происходит выполнение «упрощенной» сборки вместо стандартной. Это позволяет разработанному методу использовать меньше оперативной памяти и процессорного времени при работе, сохраняя при этом разнообразие в геномных последовательностях исследуемых организмов. Отметим, что в отличие от абстрактных подходов, таких, как анализ *k*-мерного спектра, получаемые признаки являются информативными (могут быть уплощены в длинную ДНК последовательность и исследованы).

Далее подробнее рассматривается каждый из этапов предлагаемого решения.

3.1.1. Подсчет надежных *k*-меров

Согласно исследованиям, опубликованных в работах [63, 70], ошибки секвенирования, также как и мало представленные штаммы бактерий, ведут к появлению «зашумленных» редких *k*-меров, которые мешают анализу данных и ухудшают качество получаемых результатов. Для предлагаемого подхода (как и для обычных сборщиков) такие *k*-меры приводят к появлению пузырей, отростков и усложняют структуру графа де Брейна. Для предотвращения этого, стандартным шагом для большинства методов является удаление (неиспользование) редких *k*-меров, присутствующих в чтениях. Для этого используется порог, который разделяет все *k*-меры на две группы – «надежные» (присутствуют в чтениях не реже значения порога) и «подозрительные» (редко встречаемые). Выбор конкретного значения порога обычно зависит от свойств обрабатываемого набора данных – в том числе от глубины покрытия.

В предлагаемом подходе модуль **KmerCounter** выполняет подсчет частоты присутствия *k*-меров в чтениях метагенома (для каждого метагенома отдельно). При этом производится фильтрация: чтения, в

которых присутствуют неопределенные нуклеотиды (кодируются как «*N*»), не используются. По результатам подсчета частоты присутствия те *k*-меры, которые встречаются не чаще, чем выбранный порог (по умолчанию равен одному), считаются ошибками секвенирования и не используются в последующем анализе. Надежные *k*-меры сохраняются в файле и на следующем этапе образуют вершины графа де Брейна. Вычислительная сложность этапа – $O(S)$, где S – суммарная длина всех чтений.

3.1.2. Выделение неветвящихся путей

Для каждого метагенома модуль **SequenceBuilder** строит граф де Брейна из надежных *k*-меров, и после этого выполняет упрощенную сборку в нем. При этом производится поиск протяженных линейных (неветвящихся) путей в нем, и сохраняются только те их них, которые длиннее некоторого выбранного значения порога (по умолчанию 100 нуклеотидов).

Данный шаг, по сути, является упрощенной сборкой *de novo*. В отличие от стандартной сборки, он не выполняет шаги по упрощению и распутыванию графа де Брейна для получения длинных контигов. Полученные при этом короткие унитиги являются аналогами контигов, однако в разы короче их.

Вычислительная сложность этапа – $O(M \cdot N)$, где M – число метагеномов в исследовании, а N – среднее число вершин в графе де Брейна для одного метагенома (число уникальных «нажедных» *k*-меров). Теоретическая оценка в $O(N)$ для обработки графа де Брейна из N вершин является минимально возможной; более сложные алгоритмы зачастую имеют бóльшую сложность.

3.1.3. Выделение компонент связности

Модуль **ComponentCutter** объединяет линейные пути от всех используемых метагеномов в один граф де Брейна, после этого он разбивает граф на компоненты связности определенного размера

($b_1 \leq size \leq b_2$, значения по умолчанию $b_1 = 1000$ k-меров, $b_2 = 10000$ k-меров). Значения по умолчанию выбраны таким образом, чтобы компоненты в среднем соответствовали длинам генов бактерий. Каждая такая компонента на следующих шагах используется как единичный признак, при этом каждый признак используется для сравнения метагеномов.

Компоненты, которые по размеру меньше b_1, исключаются. Из компонент, которые по размеру больше b_2, выделяется «основная часть» путем отбрасывания k-меров, которые встречаются в малом числе метагеномов. Более формально, производится поиск такой границы f, что компонента, составленная только из k-меров, которые встречаются не меньше чем в f метагеномах, составляют связную компоненту подходящего размера. Поиск происходит путем итеративного увеличения порога f, начиная со значения «единица», с проверкой получившихся компонент. Обычно оказывается, что уже для небольших значений границы f подавляющая часть больших компонент разбивается на компоненты требуемого размера.

Верхняя граница вычислительной сложности этапа – $O(N \cdot M)$, где N – число вершин в объединенном графе де Брейна, а M – число метагеномов (а также максимальное значение границы f). На практике число выполняемых операций близко к $O(N)$.

3.1.4. Построение характеристических векторов

Модуль **FeaturesCalculator** строит характеристический вектор x для каждого метагенома. Характеристический вектор для метагенома $\boldsymbol{M_i}$ состоит из числовых значений $x_1..x_K$ (значений признаков), каждое из которых вычисляется по следующей формуле:

$$x_j = \sum_{k-mer \in \boldsymbol{C_j}} w(k-mer, \boldsymbol{M_i}),$$

где $\boldsymbol{C_j}$ – компонента с номером j, а функция $w(k-mer, \boldsymbol{M_i})$ возвращает частоту присутствия заданного *k*-мера в чтениях метагенома $\boldsymbol{M_i}$.

Следует отметить, что вычисление значений x_j происходит с использованием уже подсчитанных частот присутствий *k*-меров в исходных чтениях, вычисленных на первом этапе. Благодаря этому получается возможным избежать вычислительно-трудоемкого этапа выравнивания чтений на длинные нуклеотидные последовательности.

Вычислительная сложность этапа – $O(M \cdot N_C)$, где M – число метагеномов, а N_C – число *k*-меров во всех используемых компонентах.

3.1.5. Построение матрицы расстояний

Далее модуль **DistanceMatrixCalculator** вычисляет матрицу попарного различия между метагеномами. Матрица вычисляется на основе характеристических векторов для каждого метагенома, используя *индекс Брея-Кертиса* (*Bray-Curtis dissimilarity*), широко распространенный в сравнительной метагеномике. Индекс Брея-Кертиса $BC(X, Y)$ между двумя метагеномами X и Y использует нормализованные значения признаков $x_1..x_K$ и $y_1..y_K$ и вычисляется следующим образом:

$$BC(X,Y) = 1 - 2\frac{\sum_{i=1}^{K}\min(x_i, y_i)}{x_i + y_i},$$

где K – число признаков. Значение индекса Брея-Кертиса равно нулю, если значения представленности признаков эквивалентны в двух метагеномах, и единице, если нет общих признаков, представленных в обоих метагеномах. Поэтому значение индекса Брея-Кертиса трактуется как расстояние между двумя метагеномами, хотя оно не является расстоянием, так как не удовлетворяет неравенству треугольника.

3.1.6. Выполнение кластеризации и отображение графических результатов

Основываясь на полученной матрице расстояния, для наглядного отображения результатов строится тепловая карта (*heatmap*) расстояний между образцами (рисунок 29). Также производится иерархическая кластеризация образцов, используя метод средней связи (*average linkage*) и индекс Брея-Кертиса, а результаты кластеризации выводятся в виде дендрограммы (рисунок 29, слева).

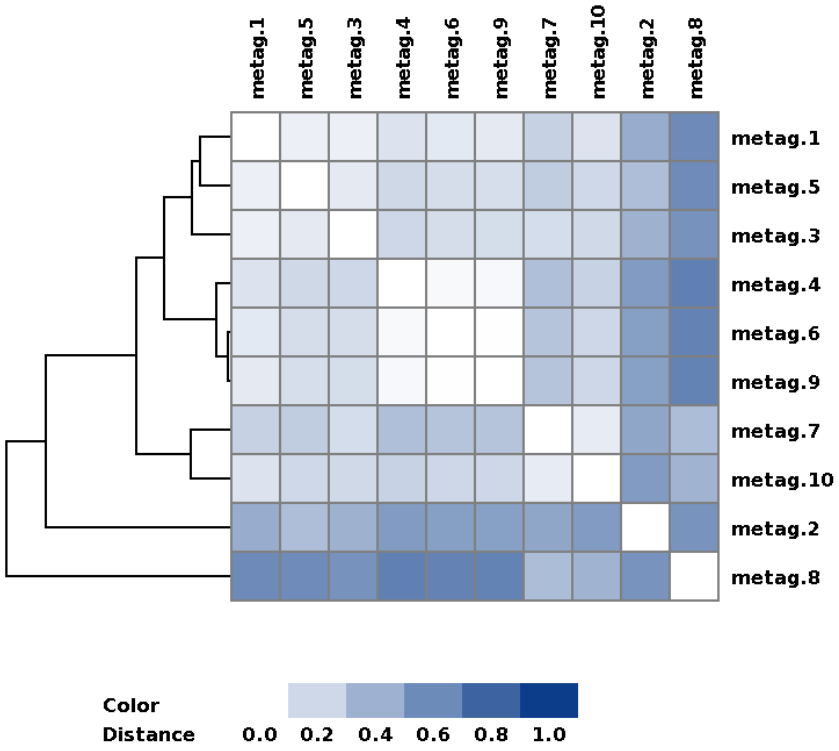


Рисунок 29 – Пример вывода предложенного метода: тепловая карта и дендрограмма

3.2. Реализация предложенного подхода

Предложенный подход был реализован на языке программирования *Java*. Благодаря этому, разработанная программа является кроссплатформенной и может быть запущена на операционных системах *Windows*, *maxOS/OS X* и *Linux*. Для запуска необходима установленная *Java* версии 1.6 или выше.

Разработанная программа и исходные коды свободно доступны в сети Интернет по адресу `https://github.com/ctlab/metafast`. Реализация предложенного подхода была выполнена с использованием библиотеки *itmo-assembler.jar*.

Разработанная программа имеет графический интерфейс пользователя. С его помощью можно запускать процесс анализа метагеномов, изменять параметры анализа и следить за ходом его выполнения.

Программа также имеет возможность простого запуска одной командой, указав только исходные файлы (доступно как из командной строки, так и из графического интерфейса). Реализация также позволяет указывать объема оперативной памяти для использования и число ядер процессора.

При этом, как и в случае с программой сборки генома, все этапы и подэтапы, необходимые для проведения сравнительного анализа, выполняются автоматически (как единый процесс). Программа имеет модульную архитектуру и позволяет независимо изменять отдельные модули, не изменяя и не перестраивая остальных частей. По умолчанию производится запуск всех этапов разработанного подхода *MetaFast* последовательно. При этом сохраняется также и возможность запуска этапов по отдельности.

При реализации предложенного подхода были выбраны значения по умолчанию для многих параметров, которые, так или иначе, влияют на итоговый результат. При этом также остается и возможность для ручной настройки параметров: можно устанавливать размер *k*-мера (от 1 до 31, по умолчанию 31), порог частоты надежного *k*-мера (единица по умолчанию), порог длин унитигов для использования (100 нуклеотидов по умолчанию), минимальный и максимальный размер компоненты b_1 и b_2 (по умолчанию $b_1 = 1000$ *k*-меров, $b_2 = 10000$ *k*-меров) и другие параметры.

3.3. Экспериментальное исследование

Было проведено экспериментальное исследование по сравнению разработанного подхода с существующими.

3.3.1. Использованные метагеномные наборы данных

Для проведения исследований было использовано четыре метагеномных набора данных с разными особенностями – один из них был искусственно сгенерирован на основании упрощенной модели реальных данных (набор № 1), остальные три были реальными данными, полученными с помощью секвенирования метагеномов и свободно доступными в сети Интернет.

Информация об используемых наборах данных приведена в таблице 10.

Таблица 10 – Использованные метагеномные наборы данных

№	Описание набора	Число метаге-номов	Число чтений на мета-геном, млн	Технология секвенирования и длина чтения, нукл.
1.	Симулированные метагеномы микробиоты кишечника человека	100	1	N/A (100)
2.	Метагеномы микробиоты метро Нью-Йорка [80]	29	2.3 ± 0.6	*Illumina MiSeq* (300)
3.	Реальные метагеномы микробиоты кишечника человека [81]	157	47 ± 11	*Illumina HiSeq* (90)
4.	Метагеномы виром озер [82–84]	31	5 ± 4	*Illumina HiSeq* (100–150)

Для первого набора данных были сгенерированы 100 искусственных метагеномов микробиоты кишечника человека путем нарезки в случайных пропорциях геномов 10 наиболее часто встречающихся бактерий в кишечнике человека. Относительные величины представленности каждого генома в метагеноме были случайно сгенерированы из нормального распределения с параметрами, которые были получены из реальных данных (по таксономическому составу из исследования [81]). Основываясь на этих величинах представленности каждого генома в метагеноме, суммарно один миллион безошибочных чтений по 100 нуклеотидов был сгенерирован для каждого метагенома. Для генерации чтений использовалось программное обеспечение *MetaSim* [86]. Всего для первого набора были получены сто метагеномов с разным таксономическим профилем.

Набор данных № 2 состоял из метагеномов микробиоты метро Нью-Йорка, которые были взяты из недавнего проекта по их секвенированию [80]. Образцы для секвенирования были получены с разных поверхностей метро (турники, места для сидения, поручни). Секвенирование производилось на платформе *Illumina MiSeq*, получившиеся чтения имели длину в 300 нуклеотидов.

Набор данных № 3 состоял из большой выборки ($M = 157$) реальных метагеномов кишечника жителей Китая. Исходные данные были впервые получены и опубликованы в работе [81]. Последующий анализ по получению функционального и таксономического состава для них был описан в работе [85].

Набор данных № 4 состоял из метагеномов виром озер, объединенных с трех проектов, ориентированных на анализ водных ресурсов – Арктических озер (озеро Линневатнет, пруд Боргдаммане, озера Тунжоен, Нордаммен и Тенндаммен) [84], соленого озера Тиррелл, расположенного в штате Виктория в Австралии [82], и озер Эри и Онтарио, расположенных в Канаде и США [83].

3.3.2. Методология экспериментов

Для проведения исчерпывающего исследования применимости предложенного подхода были выбраны следующие важные характеристики для оценки подходов сравнительного анализа метагеномов:

1. Точность получаемого результата.
2. Скорость работы решения.
3. Расход памяти.
4. Зависимость от референсной базы геномов.

Точность получаемого результата является одной из самых важных характеристик при сравнении любых решений. Для методов сравнительного анализа метагеномов точность может быть определена путем сравнения матрицы расстояний между образцами, полученной с помощью некоторого решения, с истинной матрицей расстояний. Однако для получения истинной матрицы расстояний необходимо иметь точные значения присутствия и распространенности каждой из бактерий в метагеноме. Для реальных метагеномов получить такие значения практически нельзя – в том числе из-за того, что в большинстве микробных сообществ присутствует до 50–70% бактерий с неизвестными до сих пор геномами [60]. Из-за этой особенности был сгенерирован набор данных № 1 с априорно известным составом. Благодаря этому можно было получить истинную матрицу расстояний и вычислить точность других матриц расстояний на данном наборе данных. Для оценивания точности получаемых результатов для реальных данных был использован таксономический состав метагеномов (при его наличии) для вычисления истинной матрицы расстояний. Такой подход был использован для набора данных № 3.

Несмотря на то, что для некоторых реальных метагеномов истинную матрицу расстояний посчитать не удавалось, точность результатов работы также оценивалась путем сравнения получаемых матриц расстояний

предлагаемого решения и традиционных подходов, основанных на выравнивании последовательностей на каталог известных геномов. В случае, если число известных бактерий в метагеноме достаточно велико, традиционные подходы должны давать в целом точную матрицу расстояний между метагеномами.

Набор данных № 2 состоял из небольшой группы ($M = 29$) метагеномов микробиоты метро Нью-Йорка. На этом наборе предложенный метод сравнивался с существующими решениями в скорости работы, используемой памяти и зависимости от референсной базы геномов. Ввиду отсутствия данных о таксономическом составе каждого из метагеномов, включенных в данный набор, точность получаемых матриц расстояний оценивалась с помощью сравнения с подходами, основанными на картировании на каталог известных геномов.

Для оценки работоспособности методов на больших метагеномных исследованиях был использован набор данных № 3 с метагеномами микробиоты кишечника человека. Для данного набора сравнение производилось с использованием характеристик скорости работы решения, используемой памяти, а также точности получаемых матриц расстояния (ввиду наличия данных о таксономическом составе).

Набор данных № 4 с метагеномами виром озер был использован как один из возможных случаев данных при отсутствии референсных последовательностей для значительной части микроорганизмов, входящий в состав метагеномов. Этот набор позволял более глубоко изучить возможности и зависимость существующих решений от наличия референсных последовательностей в базах данных. Скорость работы и использованная память решения тоже записывались для последующего сравнения.

Сравнение предложенного подхода *MetaFast* производилось с традиционными методами, основанными на выравнивании чтений на каталог референсных геномов, – *FOCUS* [65], *Kraken* [71], *CLARK* [72],

MetaPhlAn2 [73], а также с методом, основанном на анализе совместной сборки метагеномов, – *crAss* [78]. Совместная сборка обычно производилась с помощью сборщика *Newbler* v2.6 компании *Roche*, Швейцария, однако также были проведены эксперименты с использованием сборщиков *SPAdes* [56] и *Velvet* [42] для этих целей на наборе данных № 2. Методы запускались с установленными параметрами по умолчанию. Базы геномов для *Kraken* и *CLARK* были сгенерированы из базы данных *NCBI/RefSeq* (загружена 8 декабря 2015 г.).

3.3.3. Эксперименты с симулированными метагеномами: сравнение точности получаемых матриц

Набор данных № 1 с симулированными метагеномами микробиоты кишечника человека с априорно известным составом в них использовался для сравнения точности получаемых матриц расстояний.

На основании долей каждого генома в метагеноме, истинная матрица различия (таксономического состава) между ними была посчитана с помощью индекса Брея-Кертиса.

На данном наборе данных были запущены предложенный подход *MetaFast* и метод *crAss* (вместе с совместной сборкой, выполненной *Newbler*). Сравнение матрицы расстояния, посчитанной *MetaFast*, с истинной матрицей показало высокую корреляцию между ними (по тесту Мантеля: корреляция Спирмена $r = 0.96$, $p\text{-}value = 0.001$). Графическое отображение сравнений значений в двух матрицах показано на рисунке 30. Данный график построен путем отображения точки на графике для каждого из расстояний между парой метагеномов, где по оси X отложено значение в истинной матрице расстояний для них, а по оси Y – значение в матрице расстояний, посчитанной *MetaFast*. Данные результаты говорят об общей верности значений в матрице расстояний, полученных предложенным подходом.

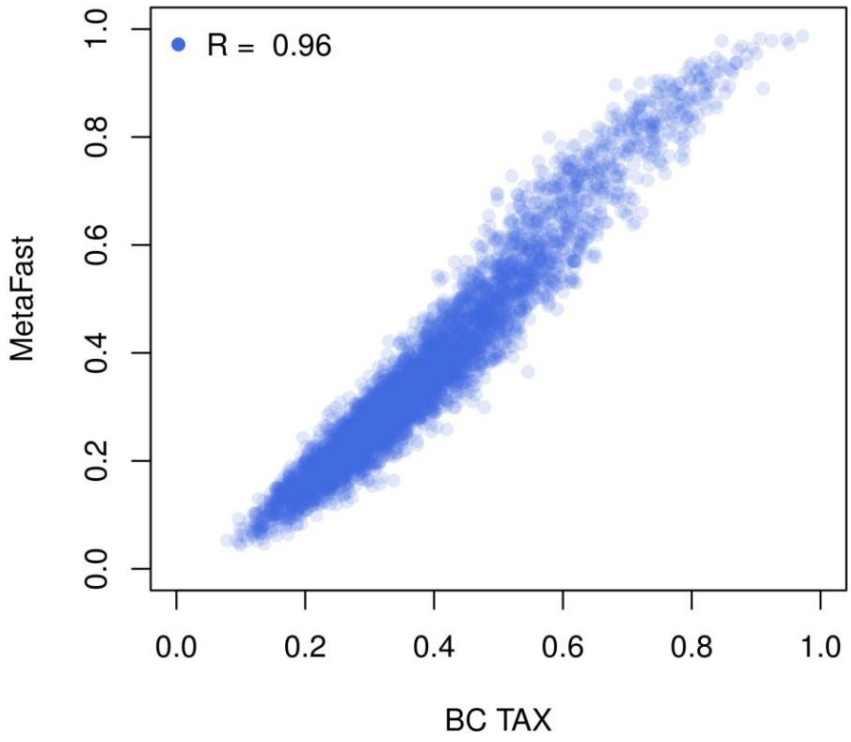


Рисунок 30 – График распределения попарных расстояний между образцами, рассчитанных *MetaFast* и по таксономическому составу. По осям координат отложены значения соответствующих расстояний

Матрица расстояний, полученная *crAss* путем анализа совместной сборки, выполненной *Newbler*, тоже была проанализирована. Корреляция с истинной матрицей расстояний оказалась очень высокой (корреляция Спирмена $r = 0.99,\ p\text{-}value = 0.001$). Таким образом, результаты *crAss* оказались более точными, чем результаты *MetaFast*. При графическом отображении сравнений матриц с истинной матрицей расстояний такой вывод также подтверждается (рисунок 31).

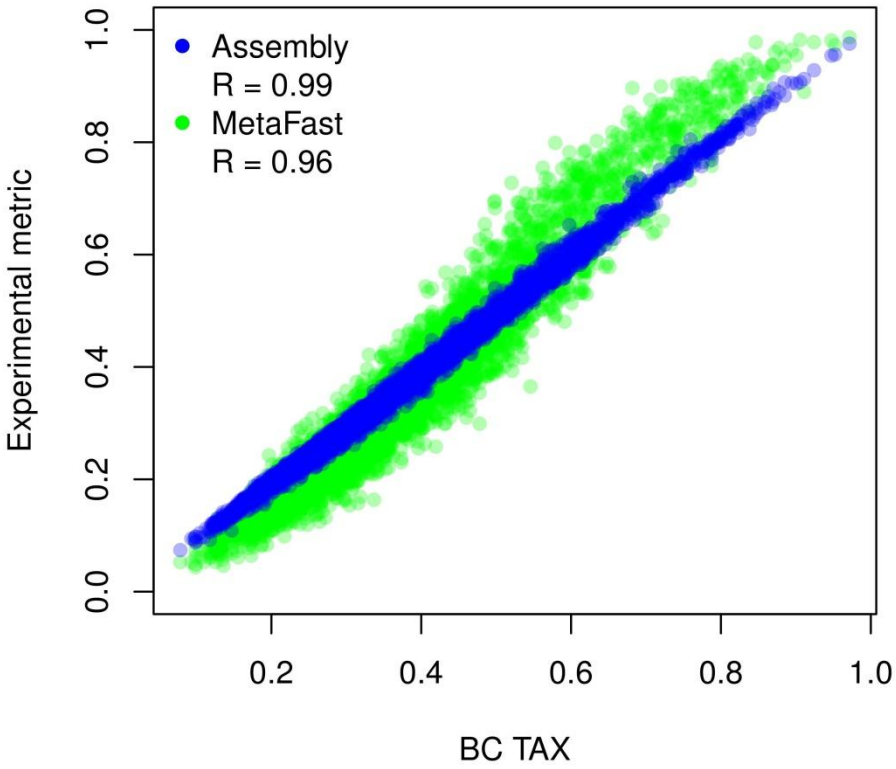


Рисунок 31 – График распределения попарных расстояний между образцами, рассчитанных *MetaFast*, *crAss* и по таксономическому составу

Также были произведены запуски *MetaFast* с параметрами размера k-мера $k = 20$ и $k = 15$. Результаты работы оказались в целом схожи с результатами для значения по умолчанию $k = 31$ (корреляция Спирмена с истинной матрицей $r = 0.96–0.98$).

Таким образом, результаты работы *MetaFast* на симулированном наборе данных оказались весьма хорошими. Полученная матрица расстояния оказалась высоко коррелирующей с истинной матрицей расстояния, что говорит об общей верности результатов, получаемых предложенным подходом на данном наборе данных.

3.3.4. Эксперименты с метагеномами микробиоты метро Нью-Йорка: сравнение работы существующих решений

Набор данных № 2 с метагеномами микробиоты метро Нью-Йорка был использован для общего сравнения существующих методов сравнительного анализа метагеномов: подход *MetaFast* сравнивался с традиционными методами профилирования таксономического состава (*Kraken*, *CLARK*, *FOCUS*, *MetaPhlAn2*) и с подходом *crAss*, основанным на анализе совместной сборки метагеномов.

Результаты сравнений работы методов приведены в таблице 11 и на рисунке 32.

В таблице 11 приводятся сравнение необходимых ресурсов для работы методов – времени вычислений и необходимого объема памяти, а также другие параметры, которые используют алгоритмы. Запуски производились на компьютере с 48-ядерным процессором *4x AMD Opteron Processor 6176 12 cores* 2.3 GHz и с объемом ОЗУ в 256 Гб.

Таблица 11 – Детальное сравнение времени работы, используемой памяти и параметров, используемых алгоритмами

Метод	Время работы, минут	Исполь-зуемая память, Гб	Число записей в референсной базе данных	Размер *k*-мера	Процент исполь-зованных чтений
MetaFast	82	14.0	–	31	89 ± 6
FOCUS	33	5.1	2766	6–8	100
Kraken	33	76.5	NCBI/RefSeq	31	61 ± 13
CLARK	16	107.4	NCBI/RefSeq	21–31	43 ± 15
MetaPhlAn2	89	3.3	~ 17000	255	–
crAss (*Newbler*)	75 (3190)	18.4 (70.8)	–	–	–

Поле «число записей в референсной базе данных» может содержать «–», что означает, что референсная база данных не использовалась. *FOCUS* использовал базу данных с полными геномами от *SEED* серверов. *NCBI/RefSeq*-база данных содержит большое число полных геномов бактериальных, простейших, вирусных и человеческих геномов. *MetaPhlAn2* использовал примерно один миллион филогенетически-специфических маркерных генов (~ 17’000 референсных геномов). *crAss* запускался на результатах работы сборщика *Newbler* (параметры работы *Newbler*’а приведены в скобках). *crAss* и *Newbler* не используют *k*-меры для анализа. *MetaPhlAn2* и *crAss* не выводят информацию о числе использованных чтений для выполнения анализа.

Рисунок 32 содержит значения корреляций Спирмена между матрицами расстояния, полученными разными методами. Значения корреляции были получены с помощью теста Мантеля ($p\text{-}value = 0.001$). Обозначение «BC MetaFast sp» применяется для матрицы расстояний, полученной методом *MetaFast*, при котором шаг «упрощенной» сборки был заменен настоящей сборкой с помощью сборщика *SPAdes*, «BC MetaFast nb» – сборка с помощью *Newbler*.

После анализа полученных данных можно сделать следующие выводы. *FOCUS* был единственным средством, которое превзошло *MetaFast* по обоим параметрам скорости работы и необходимой памяти. *MetaPhlAn2* показал схожие результаты по времени работы, но меньше использование памяти по сравнению с *MetaFast*. Оба средства *CLARK* и *Kraken*, заявленные как сверхбыстрые программы для таксономического профилирования, используя *k*-мерный спектр, требовали значительно большее памяти, чем *MetaFast.* Это связано с тем, что они хранили чтения и базу данных в памяти для ускорения анализа.

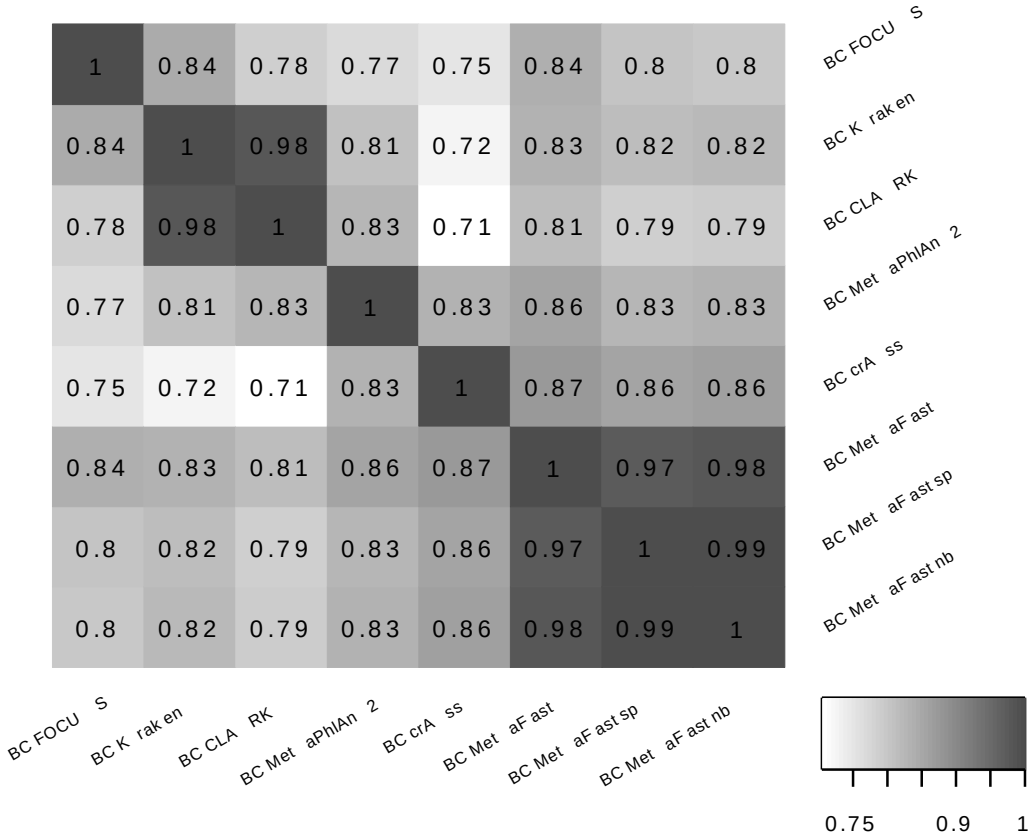


Рисунок 32 – Корреляция Спирмена между матрицами расстояний, полученными разными методами

Все четыре средства, упомянутые выше, имеют один и тот же недостаток – зависимость от полноты базы геномов. Единственный подход, близкий по принципу работы с *MetaFast*, – это *crAss*, который тоже использует совместную сборку для анализа метагеномных чтений. Однако ввиду необходимости производить совместную сборку сторонними средствами, анализ с помощью *crAss* и *Newbler* требовал почти в 40 раз больше времени и в пять раз больше памяти. В том числе, из-за больших требований на используемую память, только небольшое число метагеномов может быть включено в совместный анализ одновременно. На данном наборе данных совместная сборка *Newbler*'ом потребовала 70.8 Гб памяти (для 29 метагеномов), на наборе данных № 3 со 157 метагеномами *Newbler*'у не хватило 256 Гб оперативной памяти. Для сравнения была также произведена совместная сборка *Velvet*'ом этих же данных: сборка

заняла 24 часа 35 минут (*Newbler* потребовал 53 часа 10 минут) и использовала 238 Гб ОЗУ (*Newbler* потребовал 70.8 Гб). Однако результаты работы *crAss*'а заметно ухудшились со сборкой *Velvet* (корреляция Спирмена с таксономическими методами уменьшилась до $r = 0.50 - 0.56$, первоначально $r = 0.71 - 0.83$ при анализе *crAss* + *Newbler*).

При анализе данного набора данных также уделялось внимание числу чтений, которые были использованы средствами для анализа (таблицу 11). Хотя *crAss* и *MetaFast* используют почти все чтения для анализа, ожидалось, что методы с использованием референсной базы геномов будут использовать только часть данных. Результаты показывают, что три метода из четырех использовали достаточно большое число чтений (43–100%), что свидетельствует о том, что микробиота содержала большое число образцов с известными геномами.

Анализ корреляций итоговых матриц расстояний, полученных разными методами, говорит о том, что в целом все методы выдают схожие результаты для исследуемого набора данных. Отметим, что *MetaFast* выдает результаты, более схожие с методами таксономического профилирования, чем *crAss* (корреляция $r = 0.81 - 0.86$ для *MetaFast* против $r = 0.71 - 0.83$ для *crAss*). Хотя точные данные таксономического разложения микробиоты метро Нью-Йорка не известны, полученные значения говорят о том, что результаты *MetaFast* в целом хорошо отражают данные таксономического разложения.

Для анализа преимуществ предложенной упрощенной сборки, были проведены эксперименты с модифицированной версией метода *MetaFast*, где шаг упрощенной сборки был заменен настоящей сборкой либо с помощью сборщика *SPAdes* (основан на графе де Брейна) [56], либо с помощью *Newbler* (*Roche*, основан на подходе *overlap-layout-consensus*). Результаты экспериментов показали, что в обоих случаях вместе с заметным увеличением необходимых вычислительных ресурсов, корреляция получающихся матриц расстояний с матрицами алгоритмов

таксономического профилирования незначительно уменьшилась (отличие в корреляции $dr = 0.01 - 0.04$, рисунок 32). Вероятно, это связано с особенностью алгоритмов стандартной сборки, которые из-за шагов по увеличению длины получаемых контигов также увеличивают число ошибочных контигов (химерные контиги и т. п.) и значительно уменьшают истинное разнообразие геномных последовательностей.

Таким образом, предложенный подход показал свою применимость на данном наборе данных, работая значительно быстрее, чем стандартная сборка. Он также имеет хорошую корреляцию с методами таксономического профилирования. Требуемая память для обработки данных у предлагаемого подхода немного выше, чем у традиционных методов *FOCUS* и *MetaPhlAn2*, однако меньше, чем у методов *Kraken*, *CLARK* и *crAss*.

3.3.5. Эксперименты с метагеномами микробиоты кишечника человека: оценка работоспособности *MetaFast* на больших наборах данных

После этого, предложенный подход был протестирован на большой выборке (157 образцов) кишечных метагеномов жителей Китая из недавнего исследования [81]. Этот набор данных характеризуется разнообразным бактериальным составом, который был хорошо описан в работе [85]. В той работе также было выполнено таксономическое и функциональное разложение всех образцов. Благодаря этому можно было оценить точность и сходство работы *MetaFast* с полученными разложениями.

Исходные данные состояли из данных секвенирования *Illumina HiSeq* 2000. Всего 7.8 миллиарда чтений со средней длиной чтения в 90 нуклеотидов, суммарно 580 Гб сжатых *FASTQ*-файлов.

Полный обсчет *MetaFast* с установленными по умолчанию параметрами занял 34 часа, используя 20 ядер процессора и 90 Гб ОЗУ.

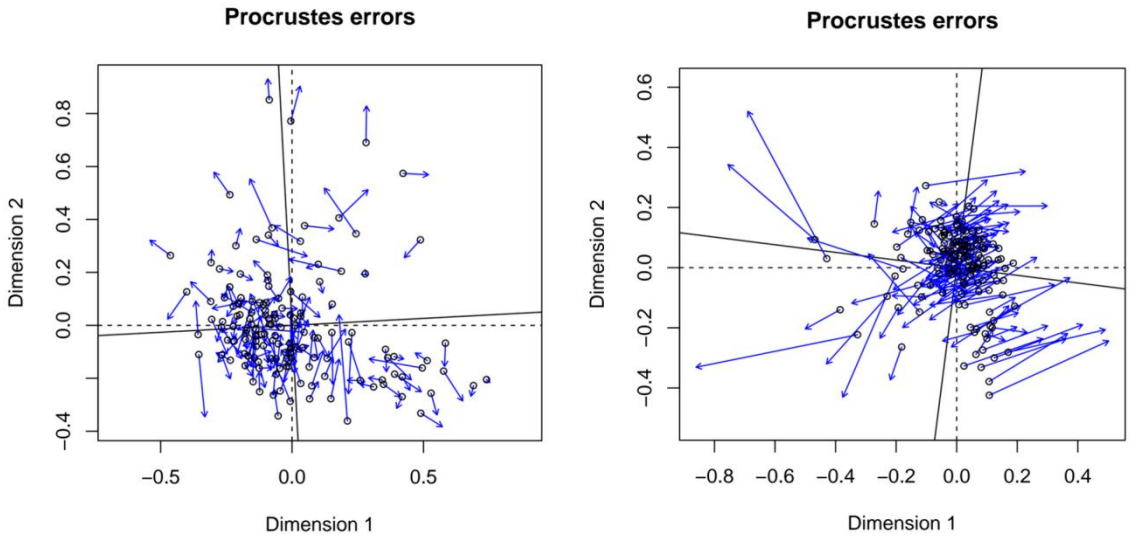
Были подсчитаны матрицы различия между образцами по таксономическому и функциональному разложению с использованием индекса Брея-Кертиса. После этого было произведено сравнение полученных матриц с матрицей расстояний, вычисленной предлагаемым методом *MetaFast*. Результаты сравнения представлены в таблице 12.

Таблица 12 – Сравнение матриц расстояния, полученных *MetaFast* и вычисленных по таксономическому и функциональному составу

	Матрица по таксономическому составу (TAX org)	Матрица по функциональному составу (TAX genus)
Корреляция Спирмена r	0,91	0,78
$p-value$	0,001	0,001

После этого по каждой из полученных матриц расстояния было отдельно выполнено многомерное шкалирование (*multi-dimensional scaling*, *MDS*), позволяющее перевести всю информацию об объектах сравнения на двумерную плоскость (с частичным сохранением расстояния между ними). Полученные представления были наложены друг на друга для выявления отличий. Этот метод анализа называется «прокрустер анализ» (*procrustes analysis*). Результаты такого анализа приведены на рисунке 33. Для каждого метагенома на рисунке изображена стрелка, обозначающая как сильно изменилось его расположение на плоскости для двух разных результатов шкалирования.

Для оценки эффективности предложенного метода была выполнена попытка запустить анализ метагеномов методом *crAss* вместе с их совместной сборкой. Ни один из сборщиков *Newbler*, *Velvet* и *SPAdes* не смог произвести совместную сборку данных такого объема при ограничении оперативной памяти в 256 Гб. Поэтому сравнение *MetaFast* с *crAss* оказалось невозможным на данном наборе данных.



А) Б)

Рисунок 33 – Прокрустер анализ для многомерного шкалирования матриц расстояния, полученной MetaFast и вычисленной по: А) таксономическому составу; Б) функциональному составу.

Таким образом, результаты экспериментов на этом наборе данных показывают, что *MetaFast* способен обрабатывать сотни метагеномов при едином анализе. Этого не удается достичь при использовании существующих методов с совместной сборкой.

3.3.6. Эксперименты с метагеномами виром озер: сравнение возможностей анализа для новых микробиот

Набор данных № 4 с метагеномами виром озер (вирусных сообществ озер) был использован для тестирования *MetaFast* и других методов сравнительного анализа метагеномов как более трудный набор данных, чем бактериальные метагеномы. Его сложность состоит в высоком разнообразии присутствующих организмов и большой доле неизвестных последовательностей в них. Для виромных образцов доля чтений с неизвестным источником типично имеет большой процент (до 60–90%) из-за того, что эти сообщества еще плохо изучены и только небольшое

число референсных последовательностей присутствует в базах данных [87]. Кроме того, частота мутаций у вирусов на несколько порядков выше, чем у других микроорганизмов, что ведет к нераспознаваемой гомологии [78]. Из-за этого традиционные методы сравнения метагеномов, основанные на выравнивании чтений на каталог известных геномов, обычно плохо подходят для таких наборов данных.

На данном наборе данных были запущены традиционные методы профилирования таксономического состава (*Kraken*, *CLARK*, *FOCUS*, *MetaPhlAn2*), метод *crAss* вместе с совместной сборкой, выполненной сборщиком *Newbler*, и предлагаемый подход *MetaFast*.

В то время как *MetaFast* использовал всю совокупность данных для анализа (96 ± 4 % чтений), *CLARK* и *Kraken* смогли идентифицировать только 1% чтений – несмотря на то, что эти программы используют базу геномов *NCBI/RefSeq*, содержащую большое число полных геномов бактериальных, простейших и вирусных организмов. *MetaPhlAn2* оказался способен определить конкретные вирусы и бактериальный состав исследуемых метагеномов, однако больша́я доля состава не была определена на уровне конкретных видов. При этом особенности используемых алгоритмов в *MetaPhlAn2* не позволяют напрямую узнать число неопознанных чтений. *FOCUS* не смог обработать данный набор метагеномов из-за того, что его база не содержит вирусных последовательностей.

В таблице 13 приведена информация о запусках методов, которые смогли отработать на данном наборе данных. На рисунке 34 представлены значения корреляции между матрицами расстояния, полученными этими методами.

Таблица 13 – Детальное сравнение времени работы, используемой памяти и параметров, используемых алгоритмами. *FOCUS*, *CLARK* и *Kraken* не включены в сравнение по причине неудачных запусков на данном наборе данных

Метод	Время работы, минут	Исполь-зуемая память, Гб	Число записей в референсной базе данных	Размер *k*-мера	Процент исполь-зованных чтений
MetaFast	45	11.4	–	31	96 ± 4
MetaPhlAn2	130	3.3	~ 17000	255	–
crAss (*Newbler*)	240 (8100)	47.0 (146.0)	–	–	–

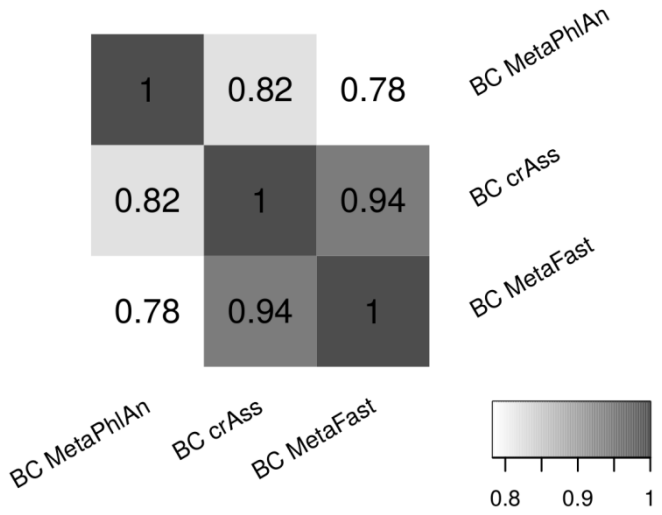
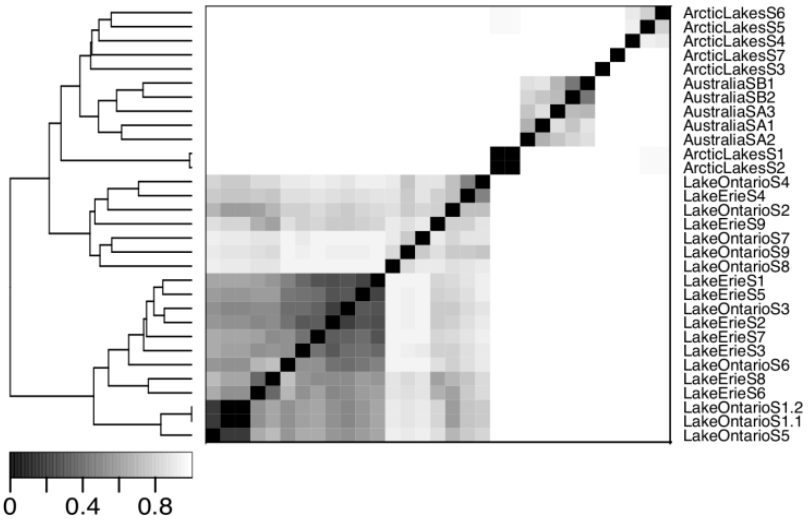


Рисунок 34 – Корреляция Спирмена между матрицами расстояния, полученными разными методами

Анализ данного набора *MetaFast*'ом занял 45 минут с использованием 11.4 Гб ОЗУ. Выполнение многомерного шкалирования по полученной матрице расстояний показало превосходное разделение исследуемых образцов по месту происхождения образца (рисунок 35 Б). Образцы из озер Эри и Онтарио оказались смешаны в результатах

кластерного анализа, что находится в соответствии с тем фактом, что эти два озера являются соседними пресными озерами в одинаковом климате, поэтому сообщества из них должны быть более схожи между собой, чем с сообществами из других озер.

А)



Б)

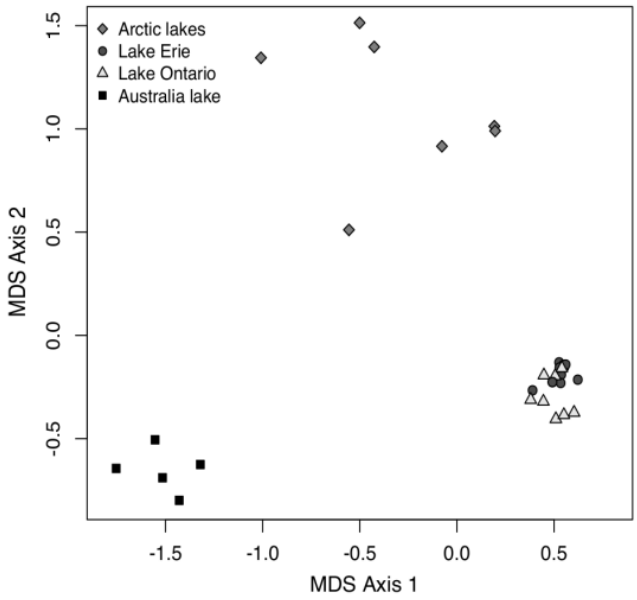
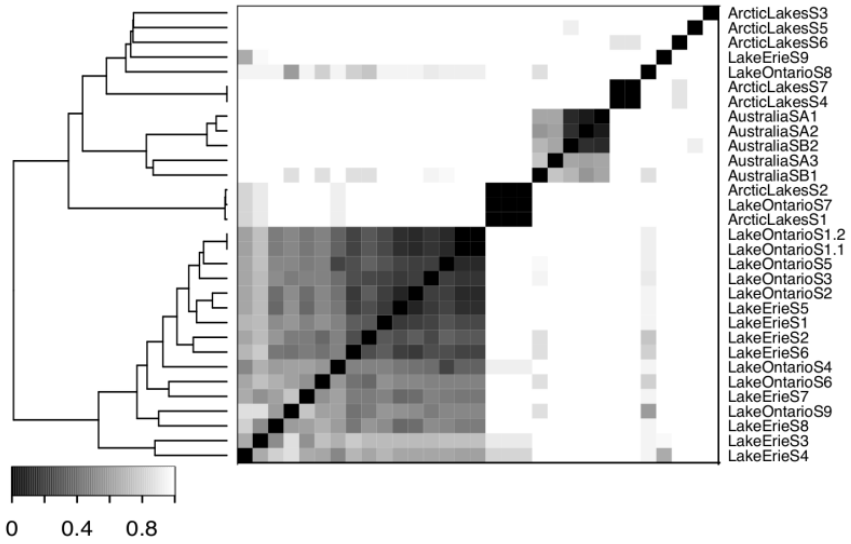


Рисунок 35 – Результаты выполнения А) кластерного анализа; Б) многомерного шкалирования по матрице расстояния, полученной предложенным методом *MetaFast*

MetaPhlAn2 отработал за 2 часа 10 минут и использовал 3.3 Гб ОЗУ. Результаты кластерного анализа и многомерного шкалирования по полученной матрице расстояния показаны на рисунке 36. Корреляция

Спирмена полученной матрицы с матрицей, вычисленной *MetaFast*, составила $r = 0.78$. Это свидетельствует о том, что, даже несмотря на то, что *MetaPhlAn2* не смог определить часть данных, в целом результаты работы алгоритмов схожи. Сравнение результатов кластерного анализа между этими двумя методами говорит о том, что даже при общей схожести матриц расстояния, *MetaPhlAn2* более склонен к смешиванию образцов из разных мест обитания, которые должны быть довольно различны – Арктические, Австралийские и несколько озер в Канаде и США.

А)



Б)

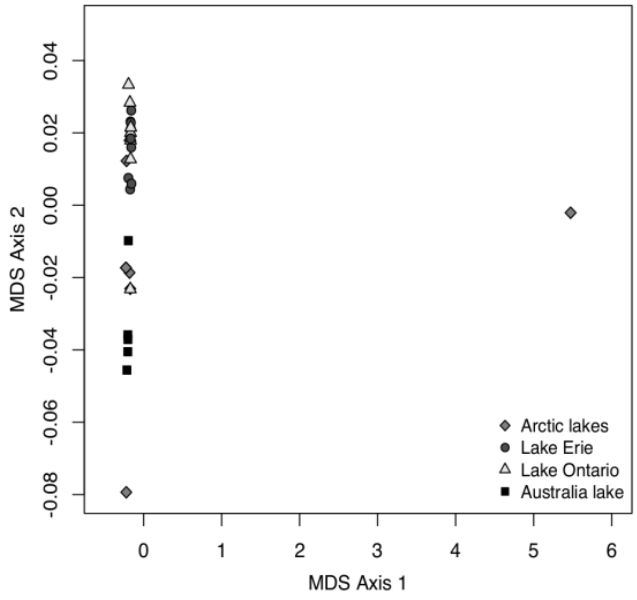
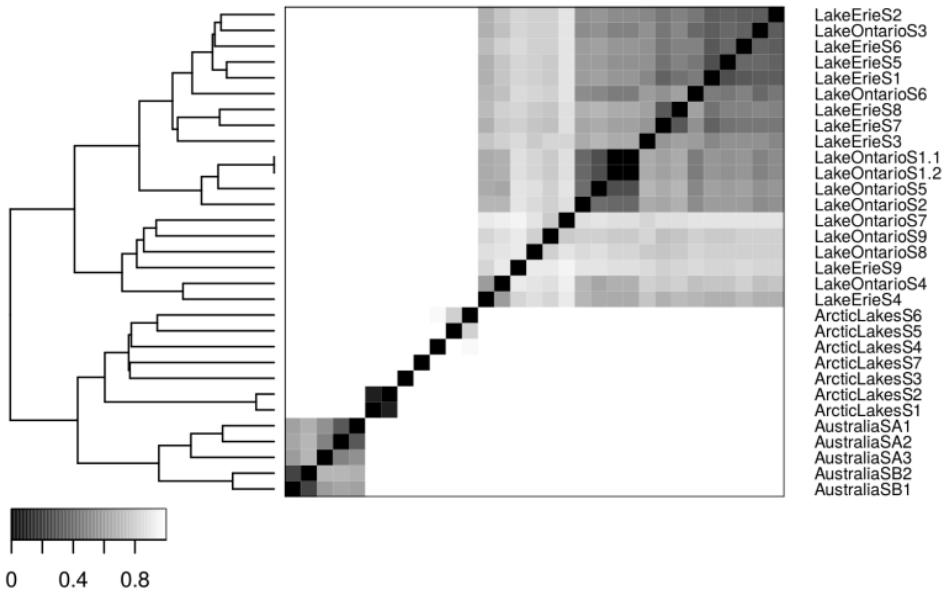


Рисунок 36 – Результаты выполнения А) кластерного анализа;
Б) многомерного шкалирования по матрице расстояния *MetaPhlAn2*

Метод сравнительного анализа, основанный на совместной сборке – *crAss*, также использовался при сравнении. Непосредственная сборка метагеномных образцов с помощью *Newbler*'а заняла достаточно много времени – пять дней и 15 часов, используя при этом 146 Гб памяти. Последующий анализ средством *crAss* потребовал четыре часа и 47 Гб памяти. Полученная матрица расстояния оказалась сильно коррелирована с матрицей, вычисленной *MetaFast*: корреляция Спирмена $r = 0.94$. Результаты кластерного анализа и многомерного шкалирования по полученной матрице представлены на рисунке 37. Сравнивая результаты многомерного шкалирования для двух методов, основанных на сборке – *crAss* и *MetaFast*, видно, что они кластеризуются одинаково. Представленные данные говорят об эквивалентности полученных результатов.

Результаты экспериментов на данном наборе данных показывают, что *MetaFast* способен хорошо работать и на ранее неисследованных средах, в отличие от традиционных методов, которые сильно зависят от наличия референсных последовательностей. Метод *crAss*, основанный на анализе совместной сборки, также способен обрабатывать такие исходные данные, однако требуемое на это время на два порядка превышает время работы предложенного метода.

А)



Б)

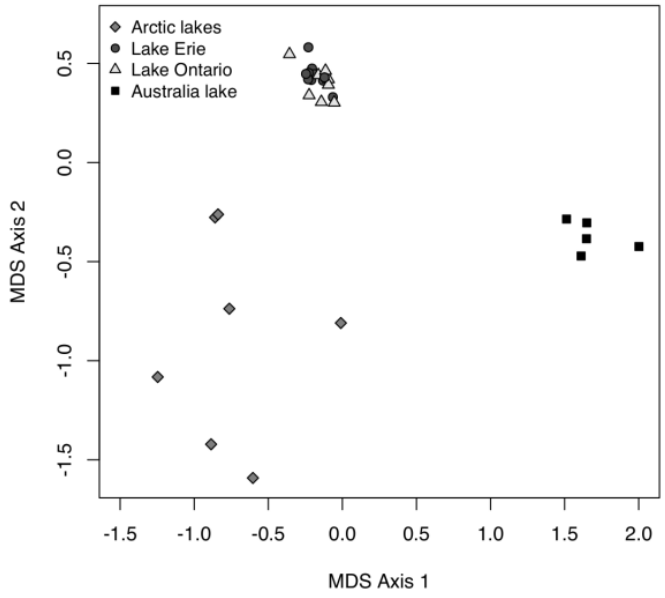


Рисунок 37 – Результаты выполнения А) кластерного анализа; Б) многомерного шкалирования по матрице расстояния, полученной методом *crAss*

Выводы по главе 3

1. Разработан автоматизированный метод сравнительного анализа метагеномов, основанный на анализе компонент связности в графе де Брейна.
2. Было проведено экспериментальное исследование предложенного метода *MetaFast* и существующих решений на четырех различных наборах исходных данных.
3. Экспериментальное исследование продемонстрировало преимущества *MetaFast* над существующими решениями по требуемым для работы вычислительным ресурсам и возможности работы на данных из ранее неисследованных сообществ.
4. Результаты и методы, описанные в данной главе, были опубликованы в работах автора [101, 105, 109, 112, 113].

ГЛАВА 4. ВНЕДРЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ

Настоящая глава посвящена описанию внедрений результатов диссертационной работы в образовательные и научно-исследовательские процессы.

Результаты диссертационной работы внедрены в учебный процесс в Санкт-Петербургском политехническом университете Петра Великого и в Университете ИТМО. Результаты работы также внедрены в Казанском (Приволжском) Федеральном Университете при выполнении научно-исследовательских работ по анализу геномов бактерий.

4.1. Внедрение результатов работы в учебный процесс в Санкт-Петербургском политехническом университете Петра Великого

В 2016 году в рамках магистерской программы «Прикладная математика и информатика. Биоинформатика» в Санкт-Петербургском политехническом университете Петра Великого были проведены лекционные занятия на тему «*de novo* сборка генома», а также магистрантами были выполнены лабораторные работы.

Целью лабораторных работ было изучение работы сборщиков на разных исходных данных. В работе требовалось собрать геномы двух бактерий – *Buchnera aphidicola* и *Mycobacterium abscessus*. Исходные данные состояли из симулированных чтений (для бактерии *B. aphidicola*) и реальных данных (для бактерии *M. abscessus)*. Дополнительная информация об используемых наборах данных приведена в таблице 14.

Таблица 14 – Информация об используемых наборах данных

№	Название организма	Платформа секвенирования	Размер генома, млн нукл.	Дополнительная информация о наборе данных
1.	*Buchnera aphidicola* str. Tuc7	Симулированные данные (модель *Illumina GA*)	0,64	Непарные чтения, длина чтений – 200 нукл. Покрытие исходного генома – 15,6х.
2.	*Mycobacterium abscessus* ATCC 19977	*Illumina HiSeq*	5,1	Парные чтения, размер фрагмента – 335 нукл., длина чтения – 100 нукл. Покрытие исходного генома – 11х.

Каждая лабораторная работа выполнялась одним или двумя магистрантами. В рамках работы им предлагалось выполнить следующие шаги: произвести анализ качества исходных данных, выполнить шаги по улучшению качества исходных данных (при необходимости), произвести сборку данных, произвести анализ качества полученной сборки. Магистрантам рекомендовалось производить сборку бактерий разными сборщиками и с разными параметрами выбранных сборщиков, а также произвести сравнение полученных сборок и выбрать лучшую из них. По результатам выполнения лабораторной работы магистрант оформлял отчет. Пример отчета приведен в Приложении 2.

В 2016 году описанная лабораторная работа была выполнена восемью магистрантами.

4.2. Внедрение результатов работы в учебный процесс в Университете ИТМО

В 2016 году в рамках курса лекций по биоинформатике на кафедре «Компьютерные технологии» Университета ИТМО были проведены лекционные занятия на тему «Сборка генома *de novo* и сравнительный анализ метагеномов», а также студентами были выполнены лабораторные работы по анализу метагеномов.

Целью выполнения лабораторной работы было изучение возможностей анализа метагеномов существующими методами. Студентам предлагалось использовать разные подходы, в том числе традиционные методы (выравнивание чтений на каталог известных геномов), методы, основанные абстрактном разложении данных (анализ *k*-мерного спектра чтений), методы, основанные на совместной сборке чтений метагеномов, и разработанный подход *MetaFast*. По результатам выполнения лабораторной работы студенты оформляли отчет.

Для лабораторных работ были использованы два метагеномных набора данных, информация о которых представлена в таблице 15.

Таблица 15 – Информация об используемых метагеномных наборах данных

№	Описание набора	Число мета-геномов	Число чтений на метагеном, тысяч	Технология секвенирования и длина чтения, нукл.
1.	Искусственный набор метагеномов из четырех изученных бактерий	3	1.2 ± 0.7	Симулированные данные (90)
2.	Искусственный набор метагеномов микробиоты кишечника человека	30	1000 ± 0	Симулированные данные (100)

В 2016 году данная лабораторная работа была выполнена двенадцатью студентами Университета ИТМО.

4.3. Внедрение результатов работы в Казанском (Приволжском) Федеральном Университете

Результаты работы использовались при выполнении научно-исследовательских работ по анализу геномов бактерий в учебно-научной лаборатории масс-спектрометрии Института фундаментальной медицины и биологии Казанского (Приволжского) федерального университета (КФУ). Программное обеспечение *ITMO Genome Assembler* было использовано для сборки геномов шести бактерий: *Serratia grimesii* Strain A2, *Bacillus ginsengihumi* Strain M2.11, *Pantoea sp.* Strain 3.5.1, *Bacillus pumilus* Strain 3-19, *Bacillus pumilus* Strain 7P и *Serratia marcescens* Strain SM6. После сборки каждый из геномов был аннотирован и проанализирован с целью выявления особенностей исследуемых образцов. Большинство бактерий были изолированы из почвы Республики Татарстан, взятых в разных местах и в разное время. Секвенирование производилось на разных платформах и с разным покрытием. Некоторые из бактерий были одновременно секвенированы на двух разных платформах – *Ion Torrent PGM* и *454 GS Junior* (*Roche*) – для получения большего покрытия исходного генома и, как следствие, итоговой сборки лучшего качества.

Дополнительная информация об исследованных бактериях приведена в таблице 16.

Таблица 16 – Информация об исследованных бактериях

№	Название организма	Размер генома, млн нукл.	Платформа секвенирования	Дополнительная информация о наборе
1.	*Serratia grimesii* Strain A2	5,1	454 GS Junior (Roche)	Непарные чтения, длина чтений – 500 нукл. Покрытие исходного генома – 12х.
2.	*Bacillus ginsengihumi* Strain M2.11	3,7	454 GS Junior (Roche)	Непарные чтения, длина чтений – 500 нукл. Покрытие исходного генома – 25х.
3.	*Pantoea sp.* Strain 3.5.1	4,9	Ion Torrent PGM	Непарные чтения, длина чтений – 200 нукл. Покрытие – 23х.
			454 GS Junior (Roche)	Непарные чтения, длина чтений – 600 нукл. Покрытие – 9х.
4.	*Bacillus pumilus* Strain 3-19	3,6	Ion Torrent PGM	Парные чтения, длина чтений – 70 нукл. Покрытие – 9,6х.
			454 GS Junior (Roche)	Непарные чтения, длина чтений – 500 нукл. Покрытие – 26х.
5.	*Bacillus pumilus* Strain 7P	3,6	Ion Torrent PGM	Парные чтения, длина чтений – 60 нукл. Покрытие – 6,8х.
			454 GS Junior (Roche)	Непарные чтения, длина чтений – 450 нукл. Покрытие – 19х.
6.	*Serratia marcescens* Strain SM6	6,1	Illumina MiSeq	Парные чтения, длина чтений – 300 нукл. Покрытие исходного генома – 290х.

После получения итоговой сборки производился дополнительный анализ, который включал в себя выполнение следующих шагов:

- анализ полученных контигов (характеристики связности, протяженности, коэффициент GC-состава);
- выполнение скэффолдинга с использованием известных референсных геномов близкородственных бактерий;
- сравнение полученных контигов/скэффолдов с известными референсными геномами близкородственных бактерий (выравнивание, подсчет покрытия, попарное сравнение, отображение результатов сравнения в виде кольцевой и линейных диаграмм);
- аннотирование контигов/скэффолдов (с использованием систем *Rapid Annotation using Subsystems Technology* (*RAST*), *NCBI Prokaryotic Genome Automatic Annotation Pipeline* (*PGAAP*));
- изучение аннотации на предмет наличия генов, отличающих исследуемый штамм от родственных бактерий;
- изучение конкретных генов, присутствующих в геноме, сопоставление их со схожими генами в других образцах с целью выявления отличий (*SNV*, структурных вариаций).

Выводы по главе 4

1. Результаты диссертационной работы внедрены в учебный процесс в Санкт-Петербургском политехническом университете Петра Великого и в Университете ИТМО при проведении лекционных занятий и лабораторных работ.
2. Результаты диссертационной работы использовались в Казанском (Приволжском) Федеральном Университете при выполнении научно-исследовательских работ по анализу геномов бактерий.

ЗАКЛЮЧЕНИЕ

В диссертационной работе получены следующие результаты:

1. Предложен автоматизированный метод сборки генома *de novo* на основе совместного применения графа де Брейна и графа перекрытий, оптимизированный по объему используемой памяти и применимый при обучении геномной биоинформатике.
2. Предложен автоматизированный метод сравнительного анализа метагеномов на основе анализа компонент связности в графе де Брейна, оптимизированный по вычислительным ресурсам и применимый в учебном процессе.
3. Предложенные методы были реализованы на языке программирования *Java*. Разработанные программы свободно доступны в сети Интернет.
4. Проведено экспериментальные сравнения разработанных программ с существующими на разных наборах данных секвенирования. Сравнение производилось как по метрикам качества получаемых результатов, так и по необходимым вычислительным ресурсам. Экспериментальные исследования подтверждают применимость разработанных подходов.
5. Результаты работы были внедрены в учебный процесс в Санкт-Петербургском политехническом университете Петра Великого и в Университете ИТМО, а также использовались в Казанском (Приволжском) Федеральном Университете при выполнении научно-исследовательских работ по анализу геномов бактерий.

СПИСОК ИСТОЧНИКОВ

Печатные издания на русском языке

1. Государственный контракт «Разработка методов сборки генома, сборки транскриптома и динамического анализа протеома». Итоговый отчет по II этапу «Разработка и реализация метода сборки транскриптома на основе восстановления фрагментов. Проведение экспериментальных исследований разработанного ЭО ПК. Обобщение и оценка результатов исследований». – 2013. – 1149 стр.
2. *Казаков С. В.* Разработка алгоритма упрощения графа перекрытий при сборке геномных последовательностей. – 2013. – Магистерская диссертация. НИУ ИТМО. – 47 стр.

Печатные издания на английском языке

3. *Watson J., Crick F.* Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid // Nature. 1953. Vol. 171, no. 4356, Pp. 737, 738.
4. *Crick F.* On protein synthesis / In Symposia of the Society for Experimental Biology. 1958. Vol. 12, Pp. 138.
5. *Sanger F., Nicklen S., Coulson A.* Dna sequencing with chain-terminating inhibitors. // Proc. Natl. Acad. Sci. USA. 1977. Vol. 74, no. 12. Pp. 5463–5467.
6. *Chinault A., Carbon J.* Overlap hybridization screening: isolation and characterization of overlapping dna fragments surrounding the leu2 gene on yeast chromosome iii // Gene. 1979. Vol. 5, no. 2. Pp. 111–126.
7. *Staden R.* A strategy of dna sequencing employing computer programs // Nucleic Acids Res. 1979. Vol. 6, no. 7. Pp. 2601–2610.
8. *Lander E., Waterman M.* Genomic mapping by fingerprinting random clones: a mathematical analysis // Genomics. 1988. 2(3), pp. 231–239.

9. *Böckenhauer H.-J., Bongartz D.* Algorithmic Aspects of Bioinformatics. Springer-Verlag Berlin Heidelberg. 2007. – 396 pp.
10. *El-Metwally S., Ouda O., Helmy M.* Next generation sequencing technologies and challenges in sequence assembly. Vol. 7. Springer Science & Business. 2014. – 118 pp.
11. *Quail M., et al*. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers // BMC genomics. 2012. Vol 13. No. 1. Pp. 341.
12. *Handelsman J., Rondon M., Brady S., Clardy J., Goodman R.* Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products // Chemistry & biology. 1998. 5(10), Pp. R245–R249.
13. *Cock P., Fields C., Goto N., Heuer M., Rice P*. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants // Nucleic Acids Research, 2010, Vol. 38, No. 6, Pp. 1767–177.
14. *Gallant J., Maier D., Storer J*. On finding minimal length superstrings. // Journal of Computer and System Sciences. 1980. Vol. 20. No. 1. Pp. 50–58.
15. *Garey M., Johnson D.* Computers and Intractability. Freeman, New York. 1979.
16. *Lesk A.* (Edited). Computational Molecular Biology, Sources and Methods for Sequence Analysis. Oxford University Press. 1988.
17. *Li. M.* Towards a DNA sequencing theory // Proc. of 31st IEEE Symp. on Foundations of Computer Science. 1990. Pp. 125–134.
18. *Peltola H., Soderlund H., Tarhio J., Ukkonen E.* Algorithms for some string matching problems arising in molecular genetics // Information Processing. 1983. Vol. 83. Pp. 53–64.
19. *Storer J.* Data compression: methods and theory. Computer Science Press. 1988.

20. *Blum A., Jiang T., Li M., Tromp J., Yannakakis M.* Linear approximation of shortest superstrings // J. ACM. 1994. Vol. 41. No. 4. Pp. 630–647.
21. *Teng S.-H., Yao F. F.* Approximating shortest superstrings // SIAM J. Comput. 1997. Vol 26. No. 2. Pp.410–417.
22. *Czumaj A. Gasieniec L., Piotrow M., Rytter W.* Sequential and parallel approximation of shortest superstrings // J. Algorithms. 1997. Vol. 23. No. 1. Pp. 74–100.
23. *Kosaraju S., Park J., Stein C.* Long tours and short superstrings // Proc. 35th Annual IEEE Symposium on Foundations of Comp. Sci. 1994. Pp. 166–177.
24. *Armen C., Stein C.* Improved length bounds for the shortest superstring problem // Algorithms and Data Structures. 1995. Pp. 494–505.
25. *Armen C., Stein C.* A 2 2/3-approximation algorithm for the shortest superstring problem // In CPM, 1996. Pp. 87–101.
26. *Breslauer D., Jiang T., Jiang Z.* Rotations of periodic strings and short superstrings // J. Algorithms. 1997. Vol. 24. No. 2. Pp. 340–353.
27. *Sweedyk Z.* A 2½-approximation algorithm for shortest superstring // SIAM J. Comput. 1997. Vol. 29. No. 3. Pp. 954–986.
28. *Kaplan H., Lewenstein M., Shafrir N., Sviridenko M.* Approximation algorithms for asymmetric tsp by decomposing directed regular multigraphs // J. ACM. 2005. Vol 52. No. 4. Pp. 602–626.
29. *Paluch K., Elbassioni K., Zuylen A.* Simpler approximation of the maximum asymmetric traveling salesman problem // STACS. 2012. Vol. 14. Pp. 501–506.
30. *Tarhio J., Ukkonen E.* A greedy approximation algorithm for constructing shortest common superstrings // Theor. Comput. Sci. 1988. Vol. 57. No. 1. Pp. 131–145.
31. *Turner J.* Approximation algorithms for the shortest common superstring problem // Inf. Comput. 1989. Vol. 83. No. 1. Pp. 1–20.

32. *Kaplan H., Shafrir N.* The greedy algorithm for shortest superstrings // Inf. Process. Lett. 2005. Vol. 93. No. 1. Pp. 13–17.
33. *Myers E.* The fragment assembly string graph // Bioinformatics. 2005. Vol. 21, issue supl 2. Pp. ii79–ii85.
34. *Medvedev P., Georgiou K., Myers G., Brudno M.* Computability of models for sequence assembly // Lecture Notes Comput. Sci. 2007. Vol. 4645. Pp. 289–301.
35. *Edmonds J., Johnson E.* Matching, Euler tours and the Chinese postman // Mathematical Programming, 1973. Vol. 5, no. 1. Pp. 88–124.
36. *Pevzner P., Tang H., Waterman M.* An Eulerian path approach to DNA fragment assembly // Proc. Natl. Acad. Sci. USA. Vol. 98. 2001. Pp. 9748–9753.
37. *Medvedev P., Brudno M.* Maximum Likelihood Genome Assembly // Journal of Computational Biology, 2009. Vol. 16, no. 8. Pp. 1101–1116.
38. *Varma A., Ranade A., Aluru S.* An improved maximum likelihood formulation for accurate genome assembly // Proc. of ICCABS'11. 2011. Pp. 165–170.
39. *Kapun E., Tsarev F.* De Bruijn Superwalk with Multiplicities Problem is NP-hard // BMC bioinformatics, 2013. 14(5), S7.
40. *Medvedev P., Pham S., Chaisson M., Tesler G., Pevzner P.* Paired de Bruijn Graphs: A Novel Approach for Incorporating Mate Pair Information into Genome Assemblers // Journal Of Computational Biology, 2011. Vol. 18. No. 11. Pp. 1625–1634.
41. *Pham S. K., Antipov D., Sirotkin A., Tesler G., Pevzner P., Alekseyev M.* Pathset Graphs: A Novel Approach for Comprehensive Utilization of Paired Reads in Genome Assembly // Journal Of Computational Biology, 2012. Vol. 19. Pp. 1–13.
42. *Zerbino D., Birney E.* Velvet: algorithms for de novo short read assembly using de Bruijn graphs // Genome Research. 2008. Vol. 18, no. 5. Pp. 821–829.

43. *Simpson J., Wong K., Jackman S., Schein J., Jones S., Birol I.* ABySS: a parallel assembler for short read sequence data // Genome Res. 2009. Vol. 19, no. 6. Pp. 1117–1123.
44. *Miller J., Koren S., Sutton G.* Assembly algorithms for next-generation sequencing data // Genomics, 2010. 95(6), Pp.315–327.
45. *International Human Genome Sequencing Consortium.* Initial sequencing and analysis of the human genome // Nature, 2001. 409, Pp. 860–921.
46. *Dobrynin P., Liu S., Tamazian G., Xiong Z., Yurchenko A., Krasheninnikova K., Kliver S., Schmidt-Küntzel A., Koepfli K., Johnson W., Kuderna L.* Genomic legacy of the African cheetah, Acinonyx jubatus // Genome biology, 2015. 16(1), Pp.1–20.
47. *Schuster S. C.* Next-generation sequencing transforms today's biology // Nature, 2007. 200(8), Pp.16–18.
48. *Kleftogiannis D., Kalnis P., Bajic V.* Comparing Memory-Efficient Genome Assemblers on Stand-Alone and Cloud Infrastructures // PLoS ONE, 2013. 8(9): e75505.
49. *Chikhi R., Rizk G.* Space-efficient and exact de Bruijn graph representation based on a Bloom filter // Algorithms for Molecular Biology, 2013. 8:22.
50. *Chikhi R., Limasset A., Jackman S., Simpson J., Medvedev P.* On the representation of de Bruijn graphs / In Research in Computational Molecular Biology, 2014, Pp. 35–55.
51. *Okanohara D., Sadakane K.* Practical entropy-compressed rank/select dictionary / In Proceedings of the Meeting on Algorithm Engineering & Expermiments, 2007. Pp. 60–70. Society for Industrial and Applied Mathematics.
52. *Salikhov K., Sacomoto G., Kucherov G.* Using cascading Bloom filters to improve the memory usage for de Brujin graphs // Algorithms for Molecular Biology, 2014. 9(1), Pp.1–10.

53. *Butler J., MacCallum I., Kleber M., Shlyakhter I., Belmonte M., Lander E., Nusbaum C., Jaffe D.* ALLPATHS: de novo assembly of whole-genome shotgun microreads // Genome research, 2008. 18(5), Pp. 810–820.
54. *Zimin A., Marçais G., Puiu D., Roberts M., Salzberg S., Yorke J.* The MaSuRCA genome assembler // Bioinformatics, 2013. 29(21), Pp.2669–2677.
55. *Chevreux B., Pfisterer T., Drescher B., Driesel A., Müller W., Wetter T., Suhai S.* Using the miraEST assembler for reliable and automated mRNA transcript assembly and SNP detection in sequenced ESTs // Genome research, 2004. 14(6), Pp.1147–1159.
56. *Bankevich A., Nurk S., Antipov D., Gurevich A., Dvorkin M., Kulikov A., Lesin V., Nikolenko S., Pham S., Prjibelski A., Pyshkin A.* SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing // Journal of Computational Biology, 2012. 19(5), Pp.455–477.
57. *Luo R., Liu B., Xie Y., Li Z., Huang W., Yuan J., He G., Chen Y., Pan Q., Liu Y., Tang J.* SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler // GigaScience, 2012. 1:18.
58. *Ye C., Ma Z., Cannon C., Pop M., Douglas W.* Exploiting sparseness in de novo genome assembly // BMC bioinformatics, 2012. 13(6), S1.
59. *Gurevich A., Saveliev V., Vyahhi N., Tesler G.* QUAST: quality assessment tool for genome assemblies // Bioinformatics, 2013. 29(8), Pp. 1072–1075.
60. *Nielsen H., Almeida M., Juncker A., Rasmussen S., Li J., Sunagawa S., Plichta D., Gautier L., Pedersen A., Le Chatelier E., Pelletier E.* Identification and assembly of genomes and genetic elements in complex metagenomic samples without using reference genomes // Nature biotechnology, 2014. 32(8), Pp.822–828.

61. *Shamsaddini A., Pan Y., Johnson W., Krampis K., Shcheglovitova M., Simonyan V., Zanne A., Mazumder R.* Census-based rapid and accurate metagenome taxonomic profiling // BMC genomics, 2014.15(1), Pp.918.
62. *Vinga S., Almeida J.* Alignment-free sequence comparison – a review // Bioinformatics, 2003. 19(4), Pp.513–523.
63. *Wu Y.W., Ye Y.* A novel abundance-based algorithm for binning metagenomic sequences using l-tuples // Journal of Computational Biology, 2011. 18(3), Pp.523–534.
64. *Chatterji S., Yamazaki I., Bai Z., Eisen J.* CompostBin: A DNA composition-based algorithm for binning environmental shotgun reads / In Annual International Conference on Research in Computational Molecular Biology, 2008. Pp. 17–28. Springer Berlin Heidelberg.
65. *Silva G., Cuevas D., Dutilh B., Edwards R.* FOCUS: an alignment-free model to identify organisms in metagenomes using non-negative least squares // PeerJ 2, 2014. e425.
66. *Wu Y.W., Simmons B., Singer S.* Maxbin 2.0: an automated binning algorithm to recover genomes from multiple metagenomic datasets // Bioinformatics, 2016. 32(4), Pp. 605–607.
67. *Dubinkina V., Ischenko D., Ulyantsev V., Tyakht A., Alexeev D.* Assessment of k-mer spectrum applicability for metagenomic dissimilarity analysis // BMC bioinformatics, 2016. 17(1), Pp. 1–11.
68. *Rasheed Z., Rangwala H.* Metagenomic taxonomic classification using extreme learning machines // Journal of bioinformatics and computational biology, 2012. 10(5), 1250015.
69. *Song K., Ren J., Reinert G., Deng M., Waterman M., Sun F.* New developments of alignment-free sequence comparison: measures, statistics and next-generation sequencing // Briefings in bioinformatics, 2014. 15(3), Pp. 343–353.

70. *Wang Y., Leung H., Yiu S., Chin F.* MetaCluster 5.0: a two-round binning approach for metagenomic data for low-abundance species in a noisy sample // Bioinformatics, 2012. 28(18), Pp. i356–i362.
71. *Wood D., Salzberg S.* Kraken: ultrafast metagenomic sequence classification using exact alignments // Genome biology, 2014. 15(3), R46.
72. *Ounit R., Wanamaker S., Close T., Lonardi S.* CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers // BMC genomics, 2015. 16(1), Pp.1–13.
73. *Truong D., Franzosa E., Tickle T., Scholz M., Weingart G., Pasolli E., Tett A., Huttenhower C., Segata N.* MetaPhlAn2 for enhanced metagenomic taxonomic profiling // Nature methods, 2015. 12(10), Pp. 902, 903.
74. *Peng Y., Leung H.C., Yiu S.M., Chin F.Y.* Meta-IDBA: a de Novo assembler for metagenomic data // Bioinformatics, 2011. 27(13), Pp. i94–i101.
75. *Namiki T., Hachiya T., Tanaka H., Sakakibara Y.* MetaVelvet: an extension of Velvet assembler to de novo metagenome assembly from short sequence reads // Nucleic acids research, 2012. 40(20), Pp. e155–e157.
76. *Boisvert S., Raymond F., Godzaridis É., Laviolette F., Corbeil J.* Ray Meta: scalable de novo metagenome assembly and profiling // Genome biology, 2012. 13(12), R122.
77. *Treangen T., Koren S., Sommer D., Liu B., Astrovskaya I., Ondov B., Darling A., Phillippy A., Pop M.* MetAMOS: a modular and open source metagenomic assembly and analysis pipeline // Genome biology, 2013. 14(1), R2.
78. *Dutilh B., Schmieder R., Nulton J., Felts B., Salamon P., Edwards R., Mokili J.* Reference-independent comparative metagenomics using cross-assembly: crAss // Bioinformatics, 2012. 28(24), Pp. 3225–3231.

79. *Myers E., Sutton G., Delcher A., Dew I., Fasulo D., Flanigan M., Kravitz S., Mobarry C., Reinert K., Remington K., Anson E.* A whole-genome assembly of Drosophila // Science, 2000. 287(5461), Pp. 2196–2204.
80. *Afshinnekoo E., Meydan C., Chowdhury S., Jaroudi D., Boyer C., Bernstein N., Maritz J.M., Reeves D., Gandara J., Chhangawala S., Ahsanuddin S.* Geospatial resolution of human and bacterial diversity with city-scale metagenomics // Cell systems, 2015. 1(1), Pp. 72–87.
81. *Qin J., Li Y., Cai Z., Li S., Zhu J., Zhang F., Liang S., Zhang W., Guan Y., Shen D., Peng Y.* A metagenome-wide association study of gut microbiota in type 2 diabetes // Nature, 2012. 490(7418), Pp. 55–60.
82. *Emerson J., Thomas B., Andrade K., Heidelberg K., Banfield J.* New approaches indicate constant viral diversity despite shifts in assemblage structure in an Australian hypersaline lake // Applied and environmental microbiology, 2013. 79(21), Pp. 6755–6764.
83. *Mohiuddin M., Schellhorn H.E.* Spatial and temporal dynamics of virus occurrence in two freshwater lakes captured through metagenomic analysis // Frontiers in microbiology, 2015. 6:960.
84. *de Cárcer D.A., López-Bueno A., Pearce D.A., Alcamí A.* Biodiversity and distribution of polar freshwater DNA viruses // Science advances, 2015. 1(5), e1400127.
85. *Tyakht A., Kostryukova E., Popenko A., Belenikin M., Pavlenko A., Larin A., Karpova I., Selezneva O., Semashko T., Ospanova E., Babenko V.* Human gut microbiota community structures in urban and rural populations in Russia // Nature communications, 2013. 4:900.
86. *Richter D., Ott F., Auch A., Schmid R., Huson D.* MetaSim – a sequencing simulator for genomics and metagenomics // PloS one, 2008. 3(10), e3373.
87. *Mokili J., Rohwer F., Dutilh B.* Metagenomics and future perspectives in virus discovery // Current opinion in virology, 2012. 2(1), Pp. 63–77.

Ресурсы сети Интернет

88. *Wetterstrand, K. A.* DNA sequencing costs: data from the NHGRI Genome Sequencing Program (GSP). National Human Genome Research Institute. – URL: http://www.genome.gov/sequencingcosts.
89. *DNA Sequencing Costs*: Data from the NHGRI Genome Sequencing Program (GSP). Overview. – URL: https://www.genome.gov/27541954/dna-sequencing-costs-data.
90. *Illumina, Inc.* – URL: http://www.illumina.com.
91. *FASTA format*. – URL: http://zhanglab.ccmb.med.umich.edu/FASTA/.
92. *FASTQ format*. – URL: https://en.wikipedia.org/wiki/FASTQ_format.
93. *de novo* Genome Assembly Assessment Project workshop *(dnGASP)*. – URL: http://big.crg.cat/rgasp_dngasp.
94. *The Assemblathon*. – URL: http://assemblathon.org.
95. CLC Genomics Workbench - QIAGEN Bioinformatics. – URL: https://www.qiagenbioinformatics.com/products/clc-genomics-workbench.
96. Genome Announcements Journal. – URL: http://genomea.asm.org.
97. *Научно-технический отчет* о выполнении первого этапа Государственного контракта «Разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям». 2011. – URL: http://is.ifmo.ru/genom/fragments/report-310276-1.pdf.
98. *Научно-технический отчет* о выполнении второго этапа Государственного контракта «Разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям». 2011. – URL:

http://is.ifmo.ru/genom/fragments/report-310276-2.pdf.

99. *Научно-технический отчет* о выполнении третьего этапа Государственного контракта «Разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям». 2012. – URL: http://is.ifmo.ru/genom/fragments/report-310276-3.pdf.
100. *Исенбаев В. В.* Разработка системы секвенирования ДНК с использованием paired-end данных. Бакалаврская работа. СПбГУ ИТМО. 2010. – URL: http://is.ifmo.ru/genom/_isenbaev_thesis.pdf.

Публикации автора

Статьи в журналах из перечня ВАК

101. *Казаков С. В., Шалыто А. А.* Анализ геномных и метагеномных данных в образовательных целях // Компьютерные инструменты в образовании. – 2016. – 3. – С. 5–15.
102. *Сергушичев А. А., Александров А. В., Казаков С. В., Царев Ф. Н., Шалыто А. А.* Совместное применение графа де Брейна, графа перекрытий и микросборки для *de novo* сборки генома // Известия Саратовского университета. Новая серия. Серия Математика. Механика. Информатика. – 2013. – Т. 13, вып. 2, ч. 2. – С. 51–57.
103. *Александров А. В., Казаков С. В., Мельников С. В., Сергушичев А. А., Царев Ф. Н.* Метод сборки контигов геномных последовательностей на основе совместного применения графов де Брюина и графов перекрытий // Научно-технический вестник информационных технологий, механики и оптики. – 2012. – 6(82). – С. 93–98.
104. *Александров А. В., Казаков С. В., Мельников С. В., Сергушичев А. А., Царев Ф. Н., Шалыто А. А.* Метод исправления ошибок в наборе

чтений нуклеотидной последовательности // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. – 2011. – 5(75). – С. 81–84.

Публикации в рецензируемых изданиях, индексируемых Web of Science или Scopus

105. *Ulyantsev V., Kazakov S., Dubinkina V., Tyakht A., Alexeev D.* MetaFast: fast reference-free graph-based comparison of shotgun metagenomic data // Bioinformatics. – 2016. – 32 (18). – Pp. 2760–2767.
106. *Kazakov S., Shalyto A.* Overlap graph simplification using edge reliability calculation / Proceedings of the 8th International Conference on Intelligent Systems and Agents 2014 (ISA 2014). – 2014. – Pp. 222–226.
107. *Bradnam K., Fass J., Kazakov S.,* et al. Assemblathon 2: evaluating *de novo* methods of genome assembly in three vertebrate species // GigaScience. – 2013. – 2(10). – Pp. 1–31.
108. *Alexandrov A., Kazakov S., Melnikov S., Sergushichev A., Shalyto A., Tsarev F.* Combining de Bruijn graph, overlap graph and microassembly for *de novo* genome assembly / Proceedings of the 12th annual conference in bioinformatics "Bioinformatics 2012". Stockholm, Sweden. – 2012. – Pp. 72.

Материалы конференций с участием автора

109. *Kazakov S., Ulyantsev V., Dubinkina V., Tyakht A., Alexeev D.* MetaFast: fast reference-free graph-based comparison of shotgun metagenomic data / Proceedings of the International Moscow Conference on Computational Molecular Biology 2015 (MCCMB'15). – Moscow, 2015.
110. *Казаков С. В., Шалыто А. А.* Сборка генома *de novo* на персональном компьютере / Материалы Всероссийской научной конференции по проблемам информатики СПИСОК-2016. – СПб.: ВВМ, 2016. – С. 220–222.

111. *Казаков С. В., Шалыто А. А.* Сборка генома *de novo* из данных высокопроизводительного секвенирования на персональном компьютере / Сборник трудов VII Международной научной конференции «Компьютерные науки и информационные технологии». – 2016. – С. 178–181.
112. *Казаков С. В., Ульянцев В. И., Дубинкина В. Б., Тяхт А. В., Алексеев Д. Г.* MetaFast: высокопроизводительный сравнительный анализ метагеномов на основе графа де Брейна / Сборник тезисов I международной школы-конференции студентов, аспирантов и молодых ученых «Биомедицина, материалы и технологии XXI века». – Казань, 2015. – С. 98.
113. *Ульянцев В. И., Казаков С. В., Дубинкина В. Б., Тяхт А. В., Алексеев Д. Г.* MetaFast – программное средство для высокопроизводительного сравнительного анализа метагеномов / Сборник трудов IV международной научно-практической конференции «Постгеномные методы анализа в биологии, лабораторной и клинической медицине». – Казань, 2014. – С. 103.
114. *Александров А. В., Казаков С. В., Мельников С. В., Сергушичев А. А., Царев Ф. Н., Шалыто А. А.* Метод сборки контигов геномных последовательностей на основе совместного применения графов де Брюина и графов перекрытий / Материалы Всероссийской научной конференции по проблемам информатики СПИСОК-2012. – 2012. – С. 415–418.
115. *Александров А. В., Казаков С. В., Мельников С. В., Сергушичев А. А., Царев Ф. Н.* Метод сборки контигов геномных последовательностей на основе совместного применения графов де Брюина и графов перекрытий / Сборник тезисов докладов I всероссийского конгресса молодых ученых. – СПб.: НИУ ИТМО, 2012. – 1. – С. 235–237.
116. *Александров А. В., Казаков С. В., Мельников С. В., Сергушичев А. А.* Метод *de novo* сборки контигов геномных последовательностей на

основе совместного применения графов де Брюина и графов перекрытий / Труды XIX Всероссийской научно-методической конференции «Телематика'2012». – 2012. – Т. 1. – С. 183–185.

117. *Александров А. В., Казаков С. В., Мельников С. В., Сергушичев А. А., Царев Ф. Н., Шалыто А. А.* Совместное применение графов де Брейна, графов перекрытий и микросборки для *de novo* сборки генома / Сборник тезисов III международной научно-практической конференции «Постгеномные методы анализа в биологии, лабораторной и клинической медицине». – Казань, 2012. – С. 45, 46.
118. *Александров А. В., Казаков С. В., Мельников С. В., Сергушичев А. А., Царев Ф. Н.* Метод сборки геномных последовательностей на основе совместного применения графов де Брюина и графов перекрытий / Тезисы II Международной научно-практической конференции «Постгеномные методы анализа в биологии, лабораторной и клинической медицине: геномика, протеомика, биоинформатика». – Новосибирск, 2011. – Т. 2. – С. 188.
119. *Александров А. В., Исенбаев В. В., Казаков С. В., Сергушичев А. А., Мельников С. В., Царев Ф. Н.* Метод сборки генома с помощью восстановления его фрагментов по парным чтениям / Сборник тезисов докладов конференции молодых ученых. – СПб.: НИУ ИТМО, 2011. – 1. – С. 220.

ПРИЛОЖЕНИЕ 1. СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2011614454

Программное средство для удаления ошибок из набора чтений нуклеотидной последовательности

Правообладатель(ли): ***Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» (RU)***

Автор(ы): ***Александров Антон Вячеславович, Исенбаев Владислав Вольдемарович, Казаков Сергей Владимирович, Мельников Сергей Вячеславович, Сергушичев Алексей Александрович, Царев Федор Николаевич (RU)***

Заявка № **2011612531**

Дата поступления **12 апреля 2011 г.**

Зарегистрировано в Реестре программ для ЭВМ

6 июня 2011 г.

Руководитель Федеральной службы по интеллектуальной собственности, патентам и товарным знакам

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2012616774

Программное средство для сборки квазиконтигов из парных чтений

Правообладатель(ли): ***федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Автор(ы): ***Александров Антон Вячеславович, Казаков Сергей Владимирович, Мельников Сергей Вячеславович, Сергушичев Алексей Александрович, Федотов Павел Валерьевич, Царев Федор Николаевич (RU)***

Заявка № **2012614488**

Дата поступления **4 июня 2012 г.**

Зарегистрировано в Реестре программ для ЭВМ

27 июля 2012 г.

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2013616471

Программное средство, реализующее алгоритм поиска перекрытий между квазиконтигами

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Александров Антон Вячеславович (RU), Казаков Сергей Владимирович (RU), Царев Федор Николаевич (RU), Сергушичев Алексей Александрович (RU), Федотов Павел Валерьевич (RU)***

Заявка № **2013614361**

Дата поступления **23 мая 2013 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***09 июля 2013 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2013619155

Программное средство, реализующее запуск этапов сборки генома через графический интерфейс пользователя

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Александров Антон Вячеславович (RU), Казаков Сергей Владимирович (RU), Царев Федор Николаевич (RU), Сергушичев Алексей Александрович (RU), Федотов Павел Валерьевич (RU)***

Заявка № **2013617222**

Дата поступления **08 августа 2013 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***26 сентября 2013 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2013660881

Программное средство, реализующее алгоритм упрощения графа перекрытий при сборке геномных последовательностей

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Александров Антон Вячеславович (RU), Казаков Сергей Владимирович (RU), Сергушичев Алексей Александрович (RU)***

Заявка № **2013618661**
Дата поступления **27 сентября 2013 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***21 ноября 2013 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

ПРИЛОЖЕНИЕ 2. ПРИМЕР ОТЧЕТА ПО ЛАБОРАТОНОЙ РАБОТЕ

Сборка геномов

Петухов Виктор, Плеханова Елена

7 мая 2016 г.

Для сборки использовались программы Spades и ITMO Genome Assembler.

1. Buchnera dataset

Запустим сборщики.

```
$: /home/victor/build/itmo-assembler.sh -w ./itmo/buchnera -i ./Datasets/buchnera_reads.fastq
$: spades.py -s ./Datasets/buchnera_reads.fastq -o ./spades/buchnera
$: abyss-pe name='buchnera' k=20 se="../../Datasets/buchnera_reads.fastq"

$: /opt/velvet-1.2.10/bin/velveth /home/vp76/BI/Results/bunchnera 20 -fastq /home/vp76/BI/Datasets/buchnera_reads.fastq
$: /opt/velvet-1.2.10/bin/velvetg ./Results/bunchnera
```

Результаты ITMO Genome Assembler:

- Total contigs: 8
- Contigs>=500bp: 1
- Total length: 643'970
- Maximal length: 641'549
- Mean length: 80'496
- Minimal length: 319
- N50: 641'549
- N90: 641'549

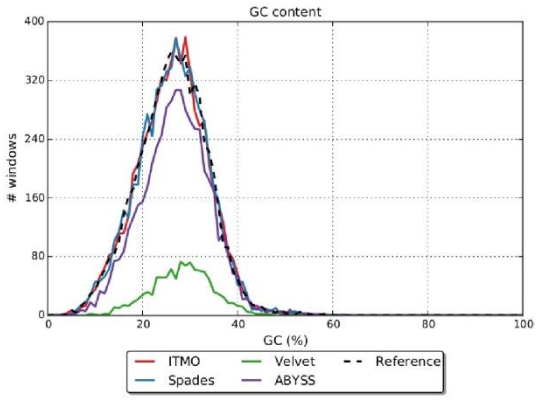
Quast.

```
$: ~/build/quast-4.0/quast.py --debug -R ./Datasets/buchnera_ref.fasta -o ./quast3/bunchera -l "ITMO, Spades" ./itmo/buchnera/contigs.fasta ./spades_bunchera/scaffolds.fasta
```

Таблица 1: Результаты QUAST

Assembly	ITMO	Spades	Velvet	ABYSS
# contigs (≥ 0 bp)	8	**1**	7842	1600
# contigs (≥ 1000 bp)	**1**	**1**	5	212
# contigs (≥ 5000 bp)	1	1	**0**	**0**
# contigs (≥ 10000 bp)	1	1	**0**	**0**
# contigs (≥ 25000 bp)	1	1	**0**	**0**
# contigs (≥ 50000 bp)	1	1	**0**	**0**
Total length (≥ 0 bp)	643970	641444	**833998**	665204
Total length (≥ 1000 bp)	**641549**	641444	5999	335179
Total length (≥ 5000 bp)	**641549**	641444	0	0
Total length (≥ 10000 bp)	**641549**	641444	0	0
Total length (≥ 25000 bp)	**641549**	641444	0	0
Total length (≥ 50000 bp)	**641549**	641444	0	0
# contigs	**1**	**1**	165	463
Largest contig	**641549**	641444	1271	4230
Total length	**641549**	641444	108419	513072
Reference length	641895	641895	641895	641895
GC (%)	26.32	26.32	28.63	26.92
Reference GC (%)	26.29	26.29	26.29	26.29
N50	**641549**	641444	636	1250
NG50	**641549**	641444	-	1044
N75	**641549**	641444	555	861
NG75	**641549**	641444	-	573
L50	**1**	**1**	69	142
LG50	1	1	-	199
L75	**1**	**1**	114	266
LG75	1	1	-	405
# N's per 100 kbp	0.00	0.00	0.00	0.00
NGA50	-	-	-	-

Velvet и Abyss не справились с задачей. Первые 2 сборщика получили фактически одинаковый результат. ITMO Assembler дополнительно нашёл 7 коротких контигов. По GC-контенту можно видеть, что последовательности всё же не идентичны:



Интересно, что Abyss выглядит здесь похожим на Spades и ITMO. Похоже, ему просто не хватило информации, чтобы собрать контиги в скаффолды.

2. MAbscessus dataset

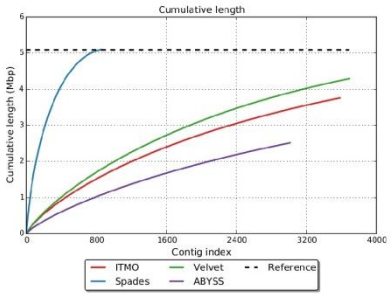
Запуск (теперь для парных прочтений):

```
$: /home/victor/build/itmo-assembler.sh -w ./itmo/m_abs -i ./Datasets/
   M_abscessus_reads_1.fastq ./Datasets/M_abscessus_reads_2.fastq
$: spades.py -1 Datasets/M_abscessus_reads_1.fastq -2 Datasets/M_abscessus_reads_2.
   fastq -o ./spades_m_abs
$: ~/build/quast-4.0/quast.py -R ./Datasets/M.abscessus_ref.fasta -o ./quast3/M_abs -l
   "ITMO, Spades" ./itmo/m_abs/contigs.fasta ./spades_m_abs/contigs.fasta

$: /opt/velvet-1.2.10/bin/velveth /home/vp76/BI/Results/M_abs 21 -separate -fastq /home
   /vp76/BI/Datasets/M_abscessus_reads_1.fastq /home/vp76/BI/Datasets/
   M_abscessus_reads_2.fastq
$: /opt/velvet-1.2.10/bin/velvetg ./Results/M_abs -exp_cov 11.0 -ins_length 135
```

Таблица 2: Результаты QUAST

Assembly	ITMO	Spades	Velvet	ABYSS
# contigs (≥ 0 bp)	6314	**940**	18428	10492
# contigs (≥ 1000 bp)	1431	777	1817	**685**
# contigs (≥ 5000 bp)	**0**	363	2	**0**
# contigs (≥ 10000 bp)	**0**	153	**0**	**0**
# contigs (≥ 25000 bp)	**0**	12	**0**	**0**
# contigs (≥ 50000 bp)	0	0	0	0
Total length (≥ 0 bp)	4681841	5124771	**5736860**	4491710
Total length (≥ 1000 bp)	2234667	**5043463**	2948760	907585
Total length (≥ 5000 bp)	0	**3928677**	10373	0
Total length (≥ 10000 bp)	0	**2454524**	0	0
Total length (≥ 25000 bp)	0	**373346**	0	0
Total length (≥ 50000 bp)	0	0	0	0
# contigs	3580	**857**	3687	3013
Largest contig	4955	**43627**	5330	3214
Total length	3757346	**5099459**	4290180	2514725
Reference length	5090491	5090491	5090491	5090491
GC (%)	63.90	64.11	64.01	63.80
Reference GC (%)	64.15	64.15	64.15	64.15
N50	1139	**9454**	1320	851
NG50	888	**9454**	1149	-
N75	787	**5345**	894	654
NG75	-	5353	673	-
L50	1097	**163**	1116	1066
LG50	1762	163	1441	-
L75	2088	**343**	2101	1911
LG75	-	342	2876	-
# N's per 100 kbp	**0.00**	**0.00**	44.99	21.99
NGA50	-	-	-	-



Здесь уже видно, что Spades даёт гораздо лучшее качество. Velvet, в отличие от прошлого раза, показал себя хорошо, и его результаты выглядят чуть лучше, чем у ITMO Assemblr'a. Abyss сильно позади.