САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

На правах рукописи

Дворкина Татьяна Евгеньевна

РАЗРАБОТКА АЛГОРИТМОВ ДЛЯ АНАЛИЗА ГРАФОВ ГЕНОМНОЙ СБОРКИ И ГЕНОМНЫХ СБОРОК

Научная специальность 1.5.8
Математическая биология, биоинформатика

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель:
кандидат физико-математических наук
Певзнер Павел Аркадьевич

Санкт-Петербург — 2023

Оглавление

Стр.

Введение 5

Глава 1. Выравнивание биологических последовательностей на графы сборки 12

1.1 Вступление 12

1.1.1 Методы секвенирования 12

1.1.2 Геномная сборка и граф геномной сборки 13

1.1.3 Выравнивание двух последовательностей 14

1.1.4 Выравнивание биологической последовательности на граф сборки 15

1.2 Выравнивания длинных прочтений на граф сборки 18

1.2.1 Описание основного алгоритма 18

1.2.2 Выравнивание последовательности на граф сборки с помощью графа выравнивания 21

1.3 Выравнивание аминокислотных последовательностей на графы сборки 22

1.3.1 Общая схема алгоритма выравнивания аминокислотных последовательностей на граф сборки 23

1.3.2 Схема подсчета веса выравнивания в случае выравнивания аминокислотных последовательностей . . . 24

1.3.3 Граф выравнивания для выравнивания аминокислотных последовательностей 25

1.4 Результаты 26

1.4.1 Сравнение методов выравнивания длинных прочтений на граф сборки 26

1.4.2 Тестирования метода выравнивания аминокислотных последовательностей 28

1.5 Заключение 31

Глава 2. Поиск генов инсектицидных белков в графе сборки . . 33

2.1 Вступление 33

Стр.

2.1.1 Инсектицидные белки *Bacillus thuringiensis* 33
2.1.2 Поиск генов инсектицидных белков 34
2.2 Общая схема работы конвейера ORFograph 35
2.3 Выравнивание ГИБ и доменных СММ к графу сборки 37
2.4 Поиск старт- и стоп-кодонов 38
2.5 Генерация полных кодирующих последовательностей (CDS) . . . 38
2.5.1 Анализ позиций СММ в последовательностях генов 39
2.5.2 Фильтрация предполагаемых белков, конфликтующих с путями контигов 39
2.6 Кластеризация последовательностей и выбор представителей . . 40
2.7 Результаты 41
2.7.1 Общая схема тестирования конвейера ORFograph 41
2.7.2 Симулированные наборы данных 42
2.7.3 Изолированные образцы бактерий из базы данных NCBI, содержащие ГИБ 46
2.7.4 Метагеномные данные метро Нью-Йорка 48
2.8 Заключение 49

Глава 3. Автоматический анализ сборок центромерных последовательностей 50
3.1 Вступление 50
3.1.1 Центромеры человека и их структура 50
3.1.2 Постулат эволюции центромер 52
3.1.3 Автоматическая аннотация центромер человека 53
3.2 Задача разложения строки 57
3.2.1 Граф разложения строки 57
3.2.2 Алгоритм StringDecomposer 59
3.2.3 Преобразование строк из нуклеотидного алфавита в алфавит блоков 60
3.3 Результаты алгоритма StringDecomposer 62
3.3.1 Описание данных 62
3.3.2 Тестирование StringDecomposer на прочтениях с высоким количеством ошибок 63
3.3.3 Определение первых гибридных мономеров 66

Стр.

3.4 Аннотация центромер 69
3.4.1 Схема работы инструмента CentromereArchitect 70
3.4.2 Схема работы инструмента HORmon 72
3.5 Аннотация последовательностей центромер генома человека . . . 73
3.6 Заключение 79

Заключение 81

Список сокращений и акронимов 83

Словарь терминов 84

Список литературы 90

Список рисунков 106

Список таблиц 108

Введение

Актуальность работы. Проблема получения полной геномной последовательности является первостепенной для биологических исследований и различных прикладных областей, от медицины до сельского хозяйства. Процесс определения последовательности нуклеотидов генома называется *секвенированием*. Последовательности, получаемые в результате процедуры секвенирования, называются *прочтения*. Длина прочтений обычно намного короче длины исследуемых геномов. Процесс объединения прочтений в более длинные геномные последовательности, *контиги*, называется *сборкой генома*.

Существует два основных типа сборки генома — сборка на основе референсной последовательности и сборка *de novo*. Референсная сборка использует информацию из уже собранных последовательностей генома референсного организма того же вида. Сборка *de novo* решает более сложную задачу получения контигов из набора прочтений без какой-либо дополнительной информации об их порядке в изначальном геноме.

Первые сборщики геномов *de novo* использовали подход Overlap-Layout-Consensus (OLC) для получения контигов [1; 2]. С появлением технологий секвенирования второго поколения (также известных как технологии секвенирования следующего поколения или NGS) в 2005 году количество прочтений увеличилось в разы, а их длина уменьшилась примерно с 1000 нуклеотидов до 100—350 нуклеотидов, поэтому подход OLC был заменен алгоритмами, основанными на графах последовательностей [3]. В процессе сборки генома набор прочтений и их потенциальные связи обычно представляют в виде графовой структуры, называемой *граф сборки*.

Прочтения, получаемые с помощью технологий секвенирования второго поколения (или *короткие прочтения*), стали основой для автоматической сборки и анализа геномов, однако их короткая длина не позволяет осуществлять качественную непрерывную сборку сложных участков генома с большим количеством повторов. Только с приходом технологий секвенирования третьего поколения (TGS), впервые описанных в 2008—2009 г.г., удалось продвинуться в сборке сложных участков последовательности генома, содержащих большое количество повторов. Однако по сравнению с короткими прочтениями NGS, первые прочтения TGS (или *длинные прочтения*) имели очень низкую точность

(коэффициент ошибок достигал 10—15%), и это опять же не позволяло выполнять сборку сложных областей, используя только данные TGS. Для повышения качества сборки стали распространяться автоматические методы, основанные на интеграции данных из различных источников [4; 5].

Графы сборки содержат больше информации о геномной последовательности, чем окончательные контиги, поскольку они хранят дополнительные связи между прочтениями. А в сочетании с другими источниками информации о геномной последовательности, например, другими типами прочтений или последовательностями белков, их можно использовать в различных приложениях, от коррекции ошибок в прочтениях [5] до улучшения сложных сборок [4;6] и реконструкции гаплотипов *de novo* [7]. Для решения данных задач обычно требуется найти потенциальное местоположение последовательности длинного прочтения или последовательности белка в графе сборки или, более формально, найти наилучшее выравнивание последовательности нуклеотидов или аминокислот на граф сборки. Первые две главы этой диссертации описывают решение задачи *выравнивания* биологических последовательностей на графы сборки из коротких прочтений и применение предложенных методов к задаче извлечения белков непосредственно из графов сборки.

Одной из самых сложных частей генома человека для сборки являются *центромеры*, особая часть хромосомы, где две хроматиды (две половины хромосомы) прикреплены друг к другу. В течение более чем двух десятилетий после публикации первого референсного генома человека [8] последовательности центромер оставались самыми длинными несобранными областями генома человека из-за сложной повторяющейся структуры последовательности [9]. Появление длинных прочтений с низким уровнем ошибок [10] и сверхдлинных прочтений [11], часто имеющих длину в сотни тысяч нуклеотидов, позволило получить первые полные сборки центромер человека [9;12]. Для анализа впервые полученных сборок центромер ученым понадобились инструменты для обработки их последовательности и выявления повторяющихся элементов [13]. Эта проблема имеет решающее значение для метаанализа сборок центромер и изучения эволюции центромер. Разработке методов анализа центромер посвящена третья глава диссертации.

Основные задачи данной работы:

– разработка и реализация алгоритмов выравнивания нуклеотидных и аминокислотных последовательностей на граф сборки;

– разработка и реализация конвейера для извлечения белковых последовательностей из графов сборки;
– разработка и реализация алгоритмов анализа структуры центромер человека.

Практическая ценность и научная новизна. Результатом данной работы стали следующие инструменты, выложенные в открытый доступ:

– SPAligner: инструмент для выравнивания биологических последовательностей на графы сборки де Брюйна, построенные из данных NGS (является частью инфраструктуры сборщика SPAdes [14]) (`cab.spbu.ru/software/spaligner/`);
– ORFograph: конвейер для извлечения фрагментированных в сборке последовательностей белков из графов сборки де Брюйна, построенных на основе данных NGS (`github.com/ablab/orf-search`);
– StringDecomposer: инструмент для точного разделения центромерных последовательностей на набор заданных повторяющихся блоков (сателлитных мономеров) (`cab.spbu.ru/software/stringdecomposer/`);
– HORmon: инструмент для аннотирования центромер человека (`cab.spbu.ru/software/HORmon/`);
– CentromereArchitect: первый инструмент для аннотирования центромер человека, включающий в себя модуль для первичного извлечения мономеров (на данный момент является частью HORmon)(`cab.spbu.ru/software/stringdecomposer/`).

В работе [15] над инструментом SPAligner предложены новые подходы для решения задачи выравнивания последовательностей на граф сборки. На текущий момент SPAligner [15] — единственный инструмент, позволяющий выравнивать как нуклеотидные, так и аминокислотные последовательности на графы сборки. Качество выравниваний прочтений на граф с помощью SPAligner сравнимо с альтернативными подходами.

ORFograph [16] — единственный конвейер, который позволяет извлекать последовательности белков произвольного семейства, пользуясь непосредственно графами сборки. В ходе работы над инструментом ORFograph были предложены новые методы фильтрации последовательностей потенциальных белков на основе информации из контигов и соединений графа. Также было показано, что ORFograph можно использовать на уже проанализированных данных и

извлекать новые потенциальные гены, не обнаруженные в оригинальных исследованиях.

Инструменты StringDecomposer [17], CentromereArchitect [18] и HORmon [19] сыграли ключевую роль в получении и анализе первых сборок последовательностей центромер человека, созданных усилиями Консорциума Telomere-to-Telomere (T2T) [20]. Работа над данными инструментами позволила понять принципы образования центромерных последовательностей, например, такие как образование так называемых гибридных мономеров, а также описать эти принципы в виде набора правил. Конвейер HORmon представляет собой первый и пока единственный существующий инструмент, позволяющий автоматически анализировать центромерные сборки человека и большие наборы центромерных прочтений, руководствуясь текущими представлениями об эволюции центромерных последовательностей.

Методы. Алгоритм выравнивания последовательностей на графы сборки в инструменте SPAligner [15] основан на алгоритме выравнивания прочтений в сборщике hybridSPAdes, описанном в [4]. Он использует классическую идею «seed-and-extend», используемую также в методах выравнивания двух и более последовательностей таких как BWA-MEM [21] или Minimap2 [22]. SPAligner использует BWA-MEM [21] для поиска коротких точных выравниваний или зерен («seeds») между последовательностью запроса и ребрами графа сборки, далее запускает алгоритм динамического программирования, чтобы расширить («extend») эти короткие выравнивания до полного выравнивания запроса на графе сборки. В [15] была описана новая модификация подхода динамического программирования для расширения коротких выравниваний, что снизило временную сложность; также изначальный алгоритм динамического программирования, предложенный в [4], был расширен для случая выравнивания аминокислотных последовательностей на граф.

Для извлечения последовательностей потенциальных белков, или открытых рамок считывания (ОРС), конвейер ORFograph [16] объединяет инструмент SPAligner [15] для поиска выравниваний последовательностей на графы сборки, и инструмент PathRacer [23] для выравнивания доменов генов, заданных скрытыми марковскими моделями (СММ), на графы сборки. Сначала ORFograph извлекает множество ОРС, соответствующих входным шаблонам, из графов сборки с помощью инструментов SPAligner и PathRacer, затем выполняет фильтрацию полученных последовательностей для выявления потенциальных генов,

фрагментированных по нескольким контигам. В [16] описаны новые алгоритмы поиска ОРС в графах сборки и новые алгоритмы выполнения фильтрации ОРС на основе структуры графа сборки и контигов.

StringDecomposer [17] принимает центромерную последовательность и набор блоков (называемых мономерами), на которые ее можно разбить, в качестве входных данных и использует алгоритм динамического программирования [24], чтобы найти наилучшее разделение последовательности на блоки. Это был первый инструмент, разработанный в рамках Консорциума T2T, для автоматизации анализа последовательности центромер, так как многие центромерные мономеры уже были извлечены из коротких прочтений и искусственно построенных эталонных моделей центромер [13; 25].

HORmon представляет собой конвейер для полного анализа последовательности центромер [19]. Он использует инструмент CentromereArchitect [18] для извлечения начального набора мономеров и StringDecomposer для разделения центромеры на мономеры. После этого HORmon извлекает повторы более высокого порядка (ПВП), используя концепцию графа де Брюйна, и разделяет центромеру на мономеры и ПВП.

Инструмент SPAligner был реализован на языке C++, в то время как все остальные инструменты были преимущественно реализованы на языке Python.

Апробация работы. Результаты данной работы были представлены в устных и стендовых докладах на нескольких международных конференциях:

1. HORmon: automated annotation of human centromeres, Постер, BiATA 2021, виртуальная конференция;
2. CentromereArchitect: inference and analysis of the architecture of centromeres, Доклад, ISMB 2021, виртуальная конференция;
3. The string decomposition problem, Доклад, ISMB 2020, виртуальная конференция;
4. SPAligner: A tool for alignment of biological sequences to assembly graph, Доклад, BiATA 2019, Санкт-Петербург, Россия;
5. GAligner: A tool for alignment of long reads to assembly graph, Постер, BiATA 2018, Санкт-Петербург, Россия.

Помимо постерных и устных докладов на международных конференциях данные работы презентовались на научных семинарах различных лабораторий.

Публикации. Основные результаты, описанные в данной диссертации, были опубликованы в виде пяти статей [15–19] в журналах, индексируемых в

базах данных Web of Science и Scopus. Все пять статей опубликованы в журналах, которые в настоящее время оцениваются как Q1.

Основные положения, выносимые на защиту.

– На основе модуля выравнивания длинных прочтений на граф сборки сборщика hybridSPAdes [4] был разработан инструмент SPAligner. В инструменте SPAligner не только были предложены и реализованы новые алгоритмы для выравнивания длинных прочтений, но также функциональность была расширена на выравнивание аминокислотных последовательностей. Наше тестирование показало, что режим выравнивания длинных прочтений SPAligner дает такие же или лучшие результаты по сравнению с текущими решениями;
– ORFograph — единственный метод, который выполняет извлечение последовательностей потенциальных генов непосредственно из графов сборки. Его можно использовать для идентификации новых белков и выделения длинных последовательностей белков, которые не могут быть выявлены из финальных контигов из-за их высокой фрагментированности;
– Инструменты StringDecomposer, CentromereArchitect и HORmon представляют собой первые инструменты для автоматического анализа последовательностей центромер. В StringDecomposer реализован единственный алгоритм, который *всегда* находит оптимальное разбиение центромерной последовательности на заданные блоки. Инструменты CentromereArchitect и HORmon были успешно использованы для анализа структуры первых полных сборок центромер человека и помогли сформировать более четкое понимание принципов эволюции центромерных последовательностей.

Личный вклад. В проекте SPAligner автор внес основной вклад в разработку и реализацию алгоритма. Автор получил первичные результаты и провел тестирование возможных вариантов использования SPAligner. Несмотря на то, что данный проект был основан на алгоритмах и кодовой базе модуля выравнивания длинных прочтений на граф сборщика hybridSPAdes [4], методы и реализация были существенно переработаны и расширены, что улучшило качество и скорость предыдущей версии модуля и позволило использовать данную функциональность не только в рамках задачи сборки.

В проекте ORFograph автор разработал и реализовал все алгоритмы, провел все тесты и эксперименты.

Автор является единственным участником проекта StringDecomposer и ключевым участником проектов CentromereArchitect и HORmon. В проектах CentromereArchitect и HORmon автор разработал и внедрил процедуры декомпозиции центромерной последовательности на ПВП.

Во всех проектах автор участвовал в проведении экспериментов и получении результатов, и активно участвовал в подготовке публикации.

Объем и структура диссертации. Данная работа включает в себя введение, 3 главы и заключение. Общий объем работы — 108 страницы, включая 22 рисунков и 8 таблиц. Работа ссылается на 137 научных публикаций.

В **первой главе** формулируется задача выравнивания длинных прочтений и аминокислотных последовательностей на графы сборки из коротких прочтений, и описываются алгоритмы, схема работы и результаты инструмента SPAligner, представленного в [15].

Во **второй главе** обсуждается идея извлечения последовательностей потенциальных белков из сложных участков графов сборки. В этой главе описывается конвейер ORFograph, который извлекает уже известные последовательности и последовательности потенциальных белков, демонстрируется применения данного конвейера к извлечению из графов сборки последовательностей белков важного класса токсинов инсектицидов. Эта глава основана на работе [16].

Третья глава описывает основные идеи и методы, реализованные в инструментах StringDecomposer, CentromereArchitect и HORmon, и представляет результаты этих инструментов на последней на данный момент сборке генома человека [9]. Материал этой главы основан на работах [17–19].

В **заключении** обсуждается влияние алгоритмов и инструментов, предложенных в предыдущих главах, на научное сообщество и описывает возможные направления их развития.

Глава 1. Выравнивание биологических последовательностей на графы сборки

1.1 Вступление

1.1.1 Методы секвенирования

Первый метод секвенирования был предложен британским биохимиком Фредериком Сэнгером и его коллегами в 1977 году [26] и был назван методом секвенирования по Сэнгеру (или методом обрыва цепи). С помощью данного метода можно секвенировать фрагменты ДНК длиной до 900 нуклеотидов. Секвенирование по Сэнгеру использовалось в проекте «Геном человека» [8], который был нацелен на создание первой точной и наиболее полной сборки генома человека. Для получения полной сборки в данном проекте осуществлялся поиск пар прочтений, у которых достаточно длинный суффикс одного прочтения был практически идентичен префиксу второго (или, иными словами, у которых было достаточно длинное *перекрытие*). Далее пары прочтений склеивались в более длинную последовательность.

Основным недостатком этого метода является его высокая стоимость по сравнению с другими технологиями (2 400 000 долларов США для секвенирования 1 миллиарда нуклеотидов) [27]. В настоящее время геномы секвенируются с использованием менее дорогих и более быстрых методов. Тем не менее секвенирование по Сэнгеру иногда используется для проверки результатов сборки генома, полученных другими технологиями, а также для секвенирования отдельных фрагментов ДНК, например, таких как фрагменты, используемые в клонировании ДНК [28].

На смену секвенированию по Сэнгеру пришли технологии секвенирования нового поколения (NGS). Одним из самых популярных подходов среди методов секвенирования нового поколения является технология, предложенная Illumina/Solexa в 2005 году. Помимо технологии от Illumina существует много других методов NGS, и, хотя каждый из них использует свою уникальную тех-

нологию, большинство из них имеют общий набор свойств, который отличает их от секвенирования по Сэнгеру [27]:

- **Небольшая длина полученного фрагмента ДНК:** на платформе Illumina длина одного прочтения варьируется от 50 до 350 нуклеотидов;
- **Относительная дешевизна:** секвенирование генома с помощью технологии NGS стоит порядка 5—150 долларов США за 1 миллиард нуклеотидов;
- **Высокая скорость:** реакции секвенирования происходят одновременно, поэтому результаты получаются намного быстрее.

В 2008—2009 г.г. начали появляться технологии секвенирования третьего поколения (TGS) [29]. Их основное отличие от методов NGS заключается в том, что благодаря совершенно новыми технологиям для чтения геномной последовательности они производят гораздо более длинные прочтения. На данный момент длина одного прочтения может составлять до нескольких миллионов нуклеотидов. Две наиболее распространенные технологии были предложены компаниями Pacific Biosciences (PacBio) [30] и Oxford Nanopore Technologies (ONT) [31]. До недавнего времени основным недостатком прочтений, получаемых с помощью TGS, был высокий процент ошибок (около 10—15% ошибок).

1.1.2 Геномная сборка и граф геномной сборки

Сборка последовательности ДНК является сложной комбинаторной задачей, которая была впервые сформулирована в начале 1980-х в связи с первыми проектами секвенирования ДНК. Результатом первых автоматических попыток сборки генома стала парадигма Overlap-Layout-Consensus (OLC) [32], где сборка проводилась в три этапа:

- **Overlap:** осуществляется поиск перекрытий между прочтениями с точностью, соответствующей ожидаемой частоте ошибок технологии секвенирования;
- **Layout:** используя полученные перекрытия, прочтения упорядочиваются и создается мозаичная сборка генома;
- **Consensus:** создается консенсусная последовательность прочтений, охватывающих одну область секвенирования.

С появлением технологий NGS количество прочтений и их длины сильно изменились, и подход OLC постепенно был заменен алгоритмами на основе графов последовательностей. Двумя основными направлениями стали подход, основанный на графах строк (англ. String graph), предложенный в [33], и подход, основанный на графах де Брюйна (англ. de Bruijn graph), предложенный в [34]. В графах строк каждая вершина представляет собой прочтение, а ребро представляет перекрытие двух прочтений. Графы строк стали основой для сборщиков Celera и Newbler [3; 32].

Графы де Брюйна стали использоваться в большинстве популярных сборщиков для коротких прочтений, таких как Velvet [35], ABySS [36], SOAPdenovo [37], SPAdes [38] и т. д. Для получения графа де Брюйна из прочтений извлекаются все k-меры (подстроки длины k), которые образуют ребра графа де Брюйна, далее два ребра соединяются вершиной, если суффикс длины $k-1$ последовательности первого ребра является префиксом последовательности второго. *Сжатый* граф де Брюйна можно построить из обычного графа де Брюйна, заменив все неветвящиеся пути единичными ребрами. Таким образом, каждому ребру и каждой вершине в графе сборки соответствует последовательность нуклеотидов (метки). Длина последовательности в вершине фиксирована внутри одного графа и равна $k-1$, в то время как длина последовательности на ребре может варьироваться. Контиги сборки генома представляют собой набор путей в графе сборки. Инструмент SPAligner [15], представленный в этой главе, выравнивает биологические последовательности на сжатые графы де Брюйна.

1.1.3 Выравнивание двух последовательностей

Рассмотрим две нуклеотидные (или аминокислотные) последовательности Q и T. *Выравниванием* двух последовательностей называется размещение одной последовательности под другой с целью нахождения похожих участков. *Глобальным выравниванием* Q и T называется выравнивание данных последовательностей по всей длине, таким образом что каждому символу одной последовательности соответствует либо символ другой последовательности, либо символ пробела. *Полуглобальным выравниванием* последовательности Q на по-

следовательность T называется глобальное выравнивание Q на произвольную подстроку T.

Рассмотрим *схему штрафов*, которая задает штраф за каждую операцию по изменению последовательности Q, например, за замену, удаление или вставку символа. Тогда *оптимальным выравниванием* Q на T будет считаться выравнивание, которое соответствует *наименьшему* штрафу за преобразование Q в T. Также в схему штрафов часто входит вознаграждение за совпадение символов в выравнивании. Простейшая схема штрафов, называемая *редакционным расстоянием*, присваивает штраф 1 за замену, удаление или вставку символа. Операции и значения штрафов в схеме штрафов могут меняться в зависимости от поставленной задачи.

Степень похожести или *идентичность* (англ. identity) двух последовательностей обычно вычисляют как разницу между длиной запроса и редакционным расстоянием, деленную на длину запроса.

Выравнивание двух последовательностей является важной задачей в анализе геномных данных. Базовые алгоритмы выравнивания, основанные на процедуре динамического программирования [39], требуют много вычислительных ресурсов и памяти для обработки большого количества длинных биологических последовательностей. Практические решения, которые выполняют выравнивание коротких и длинных прочтений на контиги сборки и друг на друга (BWA-MEM [21], Minimap2 [22], Bowtie2 [40]), используют эвристический подход, называемый *seed-and-extend*, чтобы ускорить выравнивание. Ключевая идея подхода seed-and-extend состоит в том, чтобы найти такие k-меры, называемые *зерна* (seeds), которые встречаются и в последовательности запроса и в целевой последовательности, чтобы оценить на какую часть целевой последовательности выравнивается какая часть запроса, а затем расширить (*extend*) выравнивания k-меров, чтобы получить полное выравнивание.

1.1.4 Выравнивание биологической последовательности на граф сборки

Одним из основных этапов сборки геном большинства сборщиков из коротких прочтений [14;41;42] является построение графа де Брюйна из получен-

ных прочтений, таким образом, из процесса обработки можно легко получить не только контиги, но и сам граф сборки. Графы сборки содержат избыточное количество соединений между прочтениями, но некоторые из этих соединений отражают истинные соединения, потерянные на этапе построения финальных контигов. Для использования информации о таких соединениях некоторые задачи могут решаться с помощью графов сборки, а не с помощью финальных контигов. Задача выравнивания последовательности на граф является основной задачей, которая должна быть решена для этой цели [43–47]. Решение задачи выравнивания длинных склонных к ошибкам прочтений (например, производимых PacBio и ONT) на графы сборки из коротких прочтений особенно важна в гибридной сборке [4; 6], коррекции ошибок прочтений [5] и разрешении гаплотипов [7]. Выравнивание аминокислотных последовательностей к графам сборки полезно для анализа сложных метагеномных наборов данных.

Современные решения задачи выравнивания последовательности к графу сборки также используют идею seed-and-extend, но с некоторыми модификациями. В то время как этап поиска коротких зерен практически аналогичен стадии поиска зерен в линейном случае, на этапе расширения зерен возникают новые трудности, связанные с нелинейной структурой графа сборки.

Такие гибридные сборщики как hybridSPAdes [4] и Unicycler [6] выполняют выравнивание длинных прочтений на графы сборки из коротких прочтений в качестве одного из этапов процесса сборки, тем самым дополняя недостающие соединения и упрощая граф сборки в том числе в местах, соответствующих сложным повторным регионам.

Кроме того, существуют инструменты, специально предназначенные для решения задачи выравнивания последовательностей на граф. Подмодуль для выравнивания конвейера vg [43] был разработан для выравнивания прочтений скорее к графам геномных вариаций, чем к графам сборки, однако теоретически мог быть применен к любым графам последовательностей. На первом этапе vg ищет супермаксимальные равные совпадения (SMPC), используя библиотеку GCSA2 [48], затем SMPC фильтруются и упорядочиваются в цепочки. Далее vg «раскручивает» циклы в графе, преобразуя его в ориентированный ациклический граф, и применяет метод динамического программирования, ускоренный с помощью технологии распараллеливания OKMД, для построения выравниваний между парами соседних SMPC в цепочке.

GraphAligner [49] стал широко используемым инструментом для выравнивания последовательностей длинных прочтений на графы сборки. GraphAligner $v1.0.4$ (тестируемый в оригинальной статье [15]) использовал библиотеку MUMmer4 [50] для определения потенциальных зерен выравнивания, в то время как самая последняя версия GraphAligner $v1.0.16$ использует алгоритм поиска зерен, основанный на минимайзерах, что делает последнюю версию более быстрой, но менее точной. Далее каждое зерно расширяется отдельно с помощью алгоритма Мейерса, основанного на битовых векторах, усовершенствованного для выравнивания последовательностей на графы [51].

В августе 2022 года было объявлено о пререлизе нового инструмента выравнивания последовательностей на граф в рамках инструментария Minigraph [52]. Как указано на странице инструмента на Github, данный инструмент использует многие идеи и часть реализации из Minimap2 [22]. Тем не менее, на данный момент инструмент для выравнивания из инструментария Minigraph работает только с графами последовательностей, где перекрытия между двумя последовательностями (или длины $k-1$-меров в графах де Брюйна) равны 0, поэтому его невозможно использовать на графах сборки без дополнительных преобразований.

Насколько нам известно, ни один инструмент не поддерживает выравнивание аминокислотных последовательностей. Такие инструменты как PathRacer [23] и MegaGTA [53] поддерживают выравнивание скрытых моделей Маркова (СММ), часто описывающие последовательности семейства генов или их доменов, на графы сборки, что теоретически можно рассматривать как обобщение случая выравнивания последовательности к графу. Однако, такие инструменты являются более трудоемкими из-за решения более общей задачи (выравнивание ориентированного ациклического графа на граф сборки).

Результатом, представленным в этой главе, является решение задач выравнивания нуклеотидных и аминокислотных последовательностей на графы сборки из коротких прочтений: сначала был улучшен подмодуль для выравнивания длинных прочтений сборщика hybridSPAdes, затем были предложены новые методы и расширения, и реализован инструмент SPAligner. Таким образом, SPAligner является частью архитектуры сборщика hybridSPAdes и использует алгоритмы, реализованные в данном сборщике.

В исходном исследовании [15] результаты SPAligner сравнивались с результатами инструментов vg [43] и GraphAligner [49]. Как было показано в ра-

ботах [15; 49], vg заточен на выравнивание коротких прочтений на графы вариаций, а на длинных прочтениях и сложных графах сборки продемонстрировал долгое время выполнения и малое количество выровненных прочтений, поэтому было решено опустить сравнение с vg в данной работе.

1.2 Выравнивания длинных прочтений на граф сборки

1.2.1 Описание основного алгоритма

Рассмотрим нуклеотидную последовательность S и граф сборки G. Мы опустим некоторые технические подробности, не влияющие на описание алгоритма, и будем рассматривать граф сборки G как ориентированный граф с нуклеотидными последовательностями (метками) на его ребрах и с нулевыми перекрытиями между соседними ребрами (т. е. последовательности вершин имеют нулевую длину). Путем P в графе G будет называться последовательность ребер $e_1, e_2, \ldots, e_n$, такая что каждое ребро является входящим для начальной вершины следующего ребра. Тогда если $Label(e_1), Label(e_2), \ldots, Label(e_n)$ это метки ребер пути, то меткой $Label(P)$ пути P будет являться строка, полученная конкатенацией строк $Label(e_1), Label(e_2), \ldots, Label(e_n)$. Путь оптимального полуглобального выравнивания последовательности S (запроса) на граф сборки G может быть представлен как путь P в G с наименьшим штрафом полуглобального выравнивания последовательности S на $Label(P)$ по сравнению со всеми остальными путями в G [15; 45; 49].

SPAligner применяет эвристический подход к задаче поиска полуглобального выравнивания последовательности на граф [15]. Сначала строится индекс всех меток ребер графа, используя библиотеку BWA-MEM [21]. Для каждой последовательности S процесс поиска наилучшего выравнивания начинается с поиска *якорей*, то есть областей высокого сходства между S и метками ребер G. В данной работе было решено использовать якоря, т. е. локальные выравнивания, найденные BWA-MEM, вместо базовых идеальных выравниваний в качестве зерен, чтобы извлечь выгоду из способности BWA-MEM фильтровать и расширять начальные значения до надежных локальных выравниваний

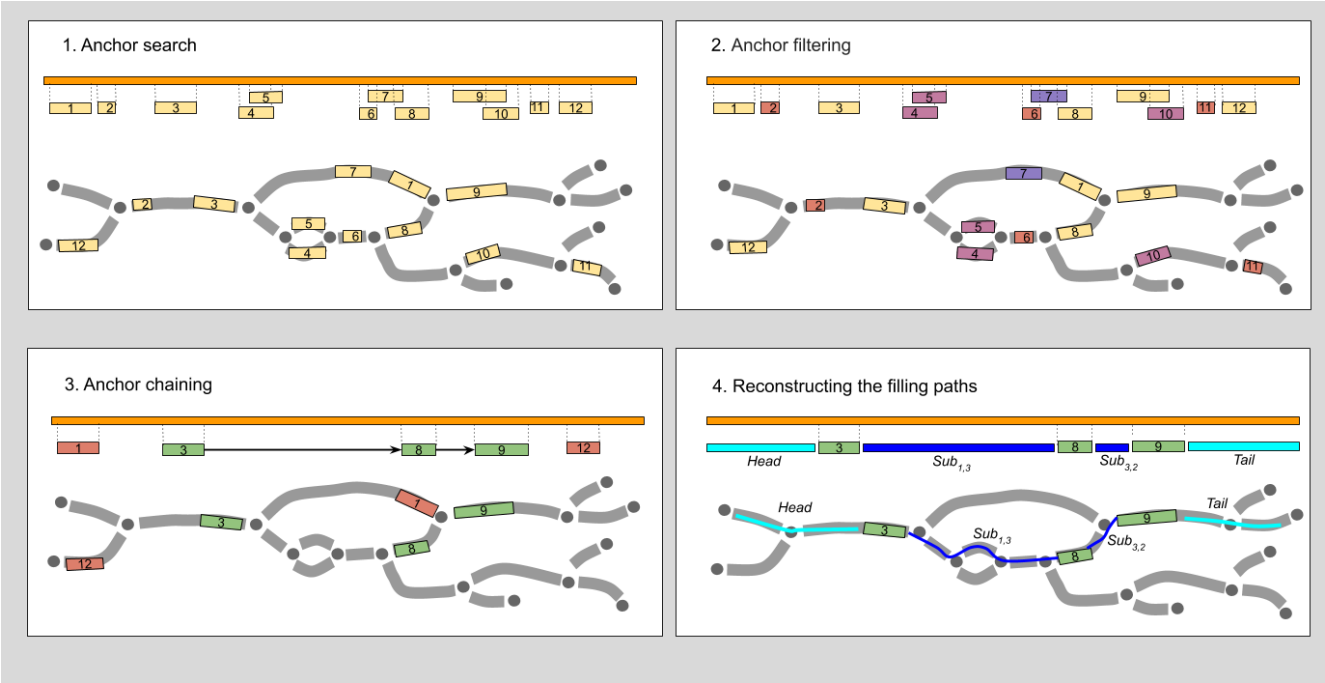

Рисунок 1.1 — Алгоритм инструмента SPAligner для выравнивания нуклеотидной последовательности на граф сборки. Процесс выравнивания последовательности запроса S (представленной оранжевой линией) к графу сборки G (представленного серыми линиями, направленными слева направо) состоит из четырех шагов. **Вверху слева: Поиск якорей.** Выравнивание якорей между S и метками ребер в G определяется с помощью инструмента BWA-MEM. **Вверху справа: Фильтрация якорей.** Идентифицированные якоря фильтруются на основе определенных критериев: фильтруются якоря с длиной меньше K (якоря 2, 6 и 11), также якоря фильтруются, если они расположены в середине длинного ребра (якоря 7) или если они в основном покрыты другими якорями (якоря 4, 5 и 10). **Внизу слева: Построение цепочки якорей.** С помощью процедуры динамического программирования идентифицируется самая тяжелая цепочка совместимых якорей. **Внизу справа: Восстановление путей между якорями и на концах последовательности.** Полное выравнивание последовательности S восстанавливается путем выравнивания подстрок S между последовательными якорями цепочки, а также подстрок префикса и суффикса.

в случае линейных последовательностей. После этого SPAligner фильтрует и упорядочивает полученные якоря и пытается найти наилучшее выравнивание S, которое проходит через финальные якоря. Процесс выравнивания S на граф сборки G состоит из четырех основных шагов [15], показанных на рисунке 1.1 и описанных более подробно в следующих абзацах.

Поиск якорей. Надежные локальные выравнивания между последовательностью S и ребрами G, сгенерированные с помощью BWA-MEM, рассматриваются как начальные якоря. Каждый якорь a описывается его позицией

выравнивания на S, идентификатором ребра и позицией выравнивания на ребре.

Фильтрация якорей. Далее SPAligner фильтрует якоря по размеру (убирая слишком короткие), их положению и их взаимному расположению в последовательности S.

Построение цепочки якорей. В графе сборки G пара якорей считается *совместимой*, если разница между их минимальным расстоянием в G и их расстоянием на последовательности S отличается не больше, чем в $min\_stretching = 1.5$ раза. Якорям присваиваются веса, равные их длинам в последовательности запроса S. SPAligner ищет самую тяжелую цепочку совместимых якорей с помощью динамического программирования и рассматривает полученную цепочку как скелет конечного выравнивания [4].

Восстановление путей между якорями и на концах последовательности. На этом этапе SPAligner пытается соединить пары соседних якорей в скелете с помощью поиска пути P_{Sub} с такой меткой $Label(P_{Sub})$, которая имеет минимальный штраф выравнивания с соответствующей областью в последовательности запроса. Во-первых, SPAligner пытается найти путь с минимальным штрафом выравнивания способом, реализованным в исходном сборщике hybridSPAdes [4], т. е. SPAligner генерирует до X путей между двумя соседними якорями и находит путь с наименьшим штрафом среди полученных путей. Если между двумя якорями больше X путей, то SPAligner пытается найти наилучшее выравнивание, используя подход динамического программирования. Для этого мы переформулировали задачу выравнивания как задачу нахождения пути минимального веса в соответствующем графе выравнивания [51; 54], следуя классическому подходу, использованному в предыдущих исследованиях [4; 51; 54; 55] (более подробно граф выравнивания описан в следующем разделе). В некоторых ситуациях SPAligner не может соединить соседние якоря, что приводит к тому, что путь выравнивания распадается на несколько непересекающихся подпутей. Кроме того, SPAligner пытается расширить выравнивание за пределы первого и последнего якоря (т. е. префикса и суффикса), чтобы получить полное выравнивание последовательности запроса.

1.2.2 Выравнивание последовательности на граф сборки с помощью графа выравнивания

Рассмотрим подпоследовательность запроса Sub, граф сборки G и линейную схему подсчета штрафа за вставки/удаления (т. е. штраф за вставку/удаление нуклеотида равен $\sigma > 0$, а штраф за замену равен $\mu > 0$). Позиция pos_G в G описывается парой $< e, pos_e >$, где e — это ребро, а pos_e — позиция на ребре на e. Определим ребра графа выравнивания $SG(G,Sub)$ с вершинами, соответствующими всем парам позиций в G и позиций в Sub, следующим образом:

1. $< pos_G, pos_{Sub} > \rightarrow < pos'_G, pos_{Sub} >$ с весом σ;
2. $< pos_G, pos_{Sub} > \rightarrow < pos_G, pos_{Sub} + 1 >$ с весом σ;
3. $< pos_G, pos_{Sub} > \rightarrow < pos'_G, pos_{Sub} + 1 >$, с весом 0, если $Sub[pos_{Sub} + 1]$ совпадает с $G[pos'_G]$, и μ иначе;
4. $< posEnd_G, pos_{Sub} > \rightarrow < posStart_G, pos_{Sub} >$ с весом 0, если $posEnd_G$ и $posStart_G$ расположены на одной вершине G (представляют собой соединение ребер),

где переменная pos_G проходит через каждую позицию в графе G, тогда как pos'_G проходит через все позиции графа, в которые можно попасть из pos_G, а pos_{Sub} проходит через все допустимые позиции в последовательности Sub.

Для поиска минимального пути выравнивания P_{Sub} между двумя якорями a_i и a_{i+1} в скелетной цепочке с координатами конечной позиции якоря a_i pos_{Gi} и координатами начальной позиции якоря a_{i+1} pos_{Gi+1}, нужно найти минимальный по весу путь, соединяющий вершины $< pos_{Gi}, 0 >$ и $< pos_{Gi+1}, |Sub| >$ в $SG(G, Sub)$ по лемме 5 в [55]. Поскольку ребра $SG(G, Sub)$ имеют неотрицательные веса, этот путь можно найти, запустив алгоритм Дейкстры [4; 55] в $SG(G, Sub)$. Кроме того, предлагаемый метод подходит и для аффинной системы подсчета штрафа за вставки/удаления (т. е. штраф за открытие удаления $\sigma > 0$, штраф за расширение удаления $\gamma > 0$ и штраф за замену $\mu > 0$) [56].

Сложность запуска алгоритма Дейкстры в графе выравнивания $SG(G,S)$, который ищет оптимальное выравнивание между запросом S и графом сборки G, составляет $O(|G| \cdot |S| \cdot \log(|G| \cdot |S|))$, где $|G|$ — суммарная длина всех ребер в G, а $|S|$ — длина последовательности S. В то же время в работах [51; 57]

описаны алгоритмы со сложностью $O(|G| \cdot |S|)$ для классической схемы оценки редакционного расстояния, где μ и σ равны 1.

Примечательно, что такая же сложность может быть достигнута простой модификацией алгоритма Дейкстры, которая может быть реализована гораздо проще, чем алгоритмы описанные в [51; 57]. В классической схеме подсчета редакционного расстояния ребра $SG(G,S)$ имеют веса либо 0, либо 1. Нетрудно показать, что в этом случае очередь с приоритетом алгоритма Дейкстры содержит только элементы, имеющие не более двух различных значений, которые могут отличаться только на 1, в любой момент выполнения алгоритма. Таким образом, очередь с приоритетом может быть заменена такой структурой данных как дек (англ. double-ended queue), где все вершины с соответствующим расстоянием через ребра весом 0 (ребра с весом 1), добавляются к началу (к концу) очереди. Реализация очереди с приоритетом с помощью дека обеспечивает выполнение всех необходимых операций за время $O(1)$, улучшая общее время выполнения до $O(|G| \cdot |S|)$.

Недавно в [45] авторы предложили аналогичное расширение с временной сложностью $O(|G| \cdot |S|)$ для аффинной системы подсчета штрафа за вставки/удаления.

В этой работе было решено воспользоваться существующими высоко оптимизированными решениями для выравнивания последовательностей друг с другом [58]. В текущей реализации SPAligner пытается выровнять суффикс Sub сразу на всю последовательность текущего ребра графа G с помощью библиотеки Edlib [58], опуская позиции внутри ребра, при поиске кратчайшего пути с использованием алгоритма Дейкстры.

1.3 Выравнивание аминокислотных последовательностей на графы сборки

Рассмотрим аминокислотную последовательность S_p и граф сборки G с нуклеотидными последовательностями на ребрах и нулевой длиной перекрытий между соседними ребрами. Путь P в G с меткой $Label(P)$ считается *валидным*, если $Label(P)$ может быть представлена в виде аминокислотной последовательности $Translated(Label(P))$, т. е. она может быть транслирована в пра-

вильную аминокислотную последовательность без стоп-кодонов внутри. Путем оптимального полуглобального выравнивания последовательности S_p на граф сборки G является валидный путь P в G, такой что штраф полуглобального выравнивания S_p на $Translated(Label(P))$ является минимальным по сравнению со всеми валидными путями в G.

1.3.1 Общая схема алгоритма выравнивания аминокислотных последовательностей на граф сборки

Процедура выравнивания аминокислотных последовательностей SPAligner похожа на процедуру выравнивания нуклеотидных последовательностей, но обладает отличиями, указанными ниже (см. рис. 1.2).

Поиск якорей. Сначала SPAligner ищет выравнивания якорей между нуклеотидным представлением S_p и предварительно проиндексированными метками транслированных ребер для шести возможных сдвигов. Для этого каждая аминокислота в S_p заменяется лексикографически минимальным кодоном, как и каждый триплет в метке ребра.

Фильтрация якорей. Процедура фильтрации, выполняемая на этом шаге, аналогична фильтрации якорей нуклеотидной последовательности. В процедуре выравнивания аминокислотных последовательностей нет этапа формирования цепи, поскольку аминокислотные последовательности обычно короче и могут быть восстановлены из одного якоря.

Расширение выравнивания. SPAligner пытается расширить каждое выравнивание якоря на графе сборки, чтобы получить путь, который имеет наилучшее выравнивание с последовательностью S_p (см. «Граф выравнивания для выравнивания аминокислотной последовательности»).

Постобработка выравниваний. Для аминокислотной последовательности S_p SPAligner может найти более одного оптимального пути через граф G, так как расширяет каждый якорь отдельно. Окончательный результат содержит все уникальные пути, длина которых превышает определенную часть длины S_p (по умолчанию: 80%). Также производится пересчет веса каждого выравнивания с помощью библиотеки Parasail [59] для получения веса выравнивания в соответствии с аминокислотными матрицами замен (BLOSUM или PAM).

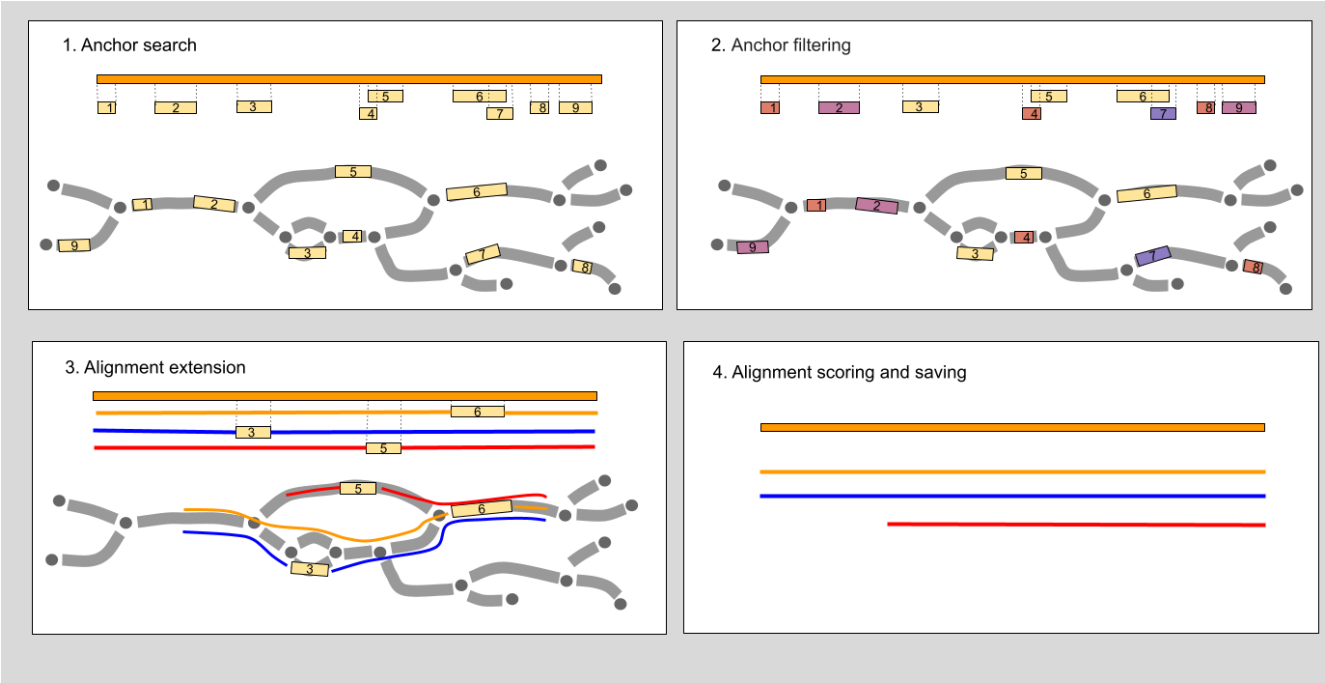

Рисунок 1.2 — Алгоритм SPAligner для выравнивания аминокислотной последовательности на граф сборки. Процесс выравнивания последовательности запроса S_p (представленной оранжевой линией) к графу сборки G (представленным серыми линиями, направленными слева направо) состоит из четырех этапов. **Вверху слева: Поиск якорей.** Поиск якорей с использованием инструмента BWA-MEM между S_p и предварительно проиндексированными метками ребер из G, преобразованными в соответствии с шестью возможными сдвигам. **Вверху справа: Фильтрация якорей.** Идентифицированные якоря фильтруются на основе определенных критериев: якоря фильтруются, если их длина меньше, чем K (для якорей 1, 4, 8), если они расположены посередине длинного ребра (якоря 2, 9) или перекрыты другими якорями (якорь 7). **Внизу слева: Расширение якорей.** Каждый якорь расширяется для получения наилучшего выравнивания. **Внизу справа: Фильтрация и сохранение выравнивания.** Полученные выравнивания фильтруются по их длине (красное выравнивание отбрасывается), и их вес пересчитывается с помощью библиотеки Parasail [59].

1.3.2 Схема подсчета веса выравнивания в случае выравнивания аминокислотных последовательностей

Для выравнивания длинных прочтений SPAligner использует обычную схему штрафов выравниваний нуклеотидных последовательностей с фиксированными штрафами за замену, в то время как для выравнивания аминокислотных последовательностей SPAligner расширяет якоря, используя специализированные матрицы аминокислотных замен, такие как матрицы BLOSUM или

PAM [60]. Числа в таких матрицах отражают относительную частоту аминокислот и частоту замен между различными аминокислотами.

В стандартной матрице замен M большее число соответствует большей частоте появления исхода. Для того чтобы использовать значения из матрицы M в подходе SPAligner, где требуется минимизировать штраф выравнивания, рассматривается матрица M', равная $M' = -M$ (по умолчанию $M = BLOSUM90$). Для выравнивания аминокислотной последовательности мы используем только линейные штрафы за вставку/удаление с коэффициентом σ (в наших экспериментах используется $\sigma = 5$).

1.3.3 Граф выравнивания для выравнивания аминокислотных последовательностей

Для выравнивания подпоследовательности запроса Sub_p на граф сборки G, используя матрицу замен аминокислот и штраф за вставку/удаление σ, мы создаем граф выравнивания SG_p. Каждая вершина в SG_p представлена тройкой $< pos_G, pos_{Sub_p}, fs >$, где pos_G — позиция в графе G, pos_{Sub_p} — позиция в Sub_p, а fs — текущая подстрока кодона длиной от 0 до 2 (пустая строка представлена символом ε). Вершины в SG_p строятся для всех возможных троек, как описано выше, а затем соединяются ребрами по следующим правилам:

1. $< pos_G, pos_{Sub_p}, fs > \rightarrow < pos'_G, pos_{Sub_p}, fs + G[pos'_G] >$ с весом 0, если $|fs| < 2$;
2. $< pos_G, pos_{Sub_p}, fs > \rightarrow < pos'_G, pos_{Sub_p}, \varepsilon >$ с весом σ, если $|fs| = 2$;
3. $< pos_G, pos_{Sub_p}, fs > \rightarrow < pos'_G, pos_{Sub_p}+1, \varepsilon >$ с весом $M'[Translate(fs+G[pos'_G]), Sub_p[pos_{Sub_p}+1]]$, и $|fs| = 2$;
4. $< pos_G, pos_{Sub_p}, \varepsilon > \rightarrow < pos_G, pos_{Sub_p}+1, \varepsilon >$ с весом σ;
5. $< posEnd_G, pos_{Sub_p}, fs > \rightarrow < posStart_G, pos_{Sub_p}, fs >$ с весом 0, если $posEnd_G$ и $posStart_G$ относятся к одной и той же вершине G (то есть представляют собой соединение двух ребер),

где pos_G — позиция в G, pos'_G — позиция в G, продолжающая pos_G, а pos_{Sub_p} проходит все возможные позиции в Sub_p.

Путь оптимального выравнивания для фиксированной подпоследовательности Sub_p, начинающейся с выравнивания якоря, может быть получен из пути

минимального веса в графе выравнивания SG_p. Однако алгоритм Дейкстры не может быть применен непосредственно к SG_p из-за наличия отрицательных весов в матрице M' и, следовательно, отрицательных ребер в SG_p. Для решения данной проблемы мы добавляем константу C (большую максимального значения в M) к весам ребер в правилах 3 и 4 и применяем к полученному графу SG'_p, который имеет только неотрицательные веса, алгоритм Дейкстры.

Несложно показать, что путь минимального веса между любыми двумя вершинами $s =< x, 0, \varepsilon >$ и $e =< y, |Sub_p| + 1, \varepsilon >$ в SG'_p также является путем минимального веса между s и e в SG_p. Если взять произвольный путь P в SG_p между s и e с весом w_P, его вес после преобразования (в SG'_p) равен $w_p + C \cdot |Sub_p|$, так как P содержит ровно $|Sub_p|$ ребер с измененными весами. Таким образом, веса всех путей между s и e будут увеличены на одну и ту же константу $|Sub_p| \cdot C$, и мы можем получить путь полуглобального выравнивания Sub_p в SG_p, запустив алгоритм Дейкстры на модифицированном графе.

1.4 Результаты

1.4.1 Сравнение методов выравнивания длинных прочтений на граф сборки

Мы сравнили SPAligner с последней версией широко используемого инструмента GraphAligner v1.0.16 [49]. Как и в исходном исследовании [15], мы запустили GraphAligner и SPAligner на PacBio и ONT прочтениях трех разных организмов: *E. coli* штамм K12, *S. cerevisiae* штамм S288C и *C. elegans* штамм Bristol N2 (Таблица 1). Графы сборки были построены с помощью сборщика SPAdes v3.12 [14].

SPAligner тестировался в двух режимах: базовый режим (SPAligner), полностью описанный выше; быстрый режим (SPAligner w/o AG), в котором SPAligner пытается построить выравнивания между якорями, используя только подход, реализованный в hybridSPAdes [4], без запуска алгоритма Дейкстры на графе выравнивания (параметры $run\_dijkstra = false$; $restore\_ends = false$). GraphAligner также тестировался в двух режимах: базовый (GA) и более

	E. coli K12	*S. cerevisiae* S288C	*C. elegans* Bristol N2
Референс	U00096.2	GCA_000146045.2	GCA_000002985.3
Illumina	ERA000206	ERP016443	-
PacBio	PacBio DevNet	ERR1655118	PacBio DevNet
ONT	Данные R9 из Loman Lab	ERP016443	PRJEB22098

Таблица 1 — Данные для тестирования SPAligner и GraphAligner. Из-за отсутствия в свободном доступе актуальных данных ONT для штамма Bristol N2 были использованы прочтения, секвенированные из близкородственного штамма *C. elegans* VC2010 [61] дикого типа (см. оригинальное исследование для обсуждения их родства).

точный режим (GA-MUM), где использовались наибольшие уникальные совпадения (MUM) вместо минимайзеров в качестве зерен и ограничения на параметры динамического программирования были ослаблены («`-seeds-mum-count -1 -C 500000 -max-cluster-extend 5 -b 35`»).

Инструменты сравнивались по следующим показателям:

- *Количество выровненных прочтений* (MR): прочтение считается выровненным, если 80% его длины покрываются одним путем выравнивания;
- *Качество выравнивания (%)* (AI): средняя похожесть между найденным выравниванием и запросом по всем выровненным прочтениям;
- *Время выполнения (ч:мин:сек)* (T): время работы инструмента в 16 потоков;
- *Память (Гб)* (M): пиковое использование ОЗУ при 16 потоках.

Результаты сравнения представлены в таблицах 2 и 3 для PacBio и ONT прочтений соответственно.

Все четыре конфигурации инструментов дали одинаковые результаты по средней похожести и количеству выровненных прочтений (см. Таблицы 2, 3). SPAligner имеет тенденцию выравнивать больше прочтений, чем другие конфигурации, без большой потери в качестве выравнивания. Анализ таких прочтений, выровненных на более чем 80% своей длины только с помощью SPAligner, показал, что он создает более длинные пути за счет восстановления выравнивания в тех частях, где прочтения и ребра графа имеют меньшее сходство. Так SPAligner требует гораздо больше времени, чем другие конфигурации. В то же время, SPAligner без режима AG имеет близкое время выполнения к GA и GA-MUM. Таким образом, подход SPAligner может конкурировать с GraphAligner,

	Aligner	MR, %	AI,%	T, ч:мин:сек	M, Гб
E. coli	GA $v1.0.16$	81.94	87	0:00:08	0.3
	GA-MUM $v1.0.16$	83.5	87	0:00:31	0.2
	SPAligner w/o AG	82.19	87	0:02:47	0.3
	SPAligner	88.27	86	0:03:23	0.3
S. cerevisiae	GA $v1.0.16$	50.37	87	0:00:13	0.4
	GA-MUM $v1.0.16$	51.36	87	0:00:44	0.3
	SPAligner w/o AG	49	86	0:02:15	0.4
	SPAligner	56.51	85	0:48:49	0.6
C. elegans	GA $v1.0.16$	79.45	87	0:00:49	1.5
	GA-MUM $v1.0.16$	82.12	87	0:02:41	1.2
	SPAligner w/o AG	74.01	87	0:06:27	0.6
	SPAligner	84.43	86	0:43:57	1.0

Таблица 2 — Результаты SPAligner и GraphAligner при выравнивании прочтений PacBio на графы сборки из коротких прочтений, построенных с помощью SPAdes с размером k-меров равным 77. Каждый набор данных содержал 10 тыс. прочтений длиной более 2 тыс. нуклеотидов. Все прогоны выполнялись с использованием 16 потоков. Для преодоления высокой фрагментации графа сборки, возникающей из-за отсутствия качественных доступных данных секвенирования *C. elegans* Bristol N2 Illumina, для получения графа сборки *C. elegans* использовали симулированные прочтения Illumina. Качество выравниваний измерялось с помощью четырех показателей: количество выровненных прочтений (MR), средняя похожесть выравниваний (AI), время выполнения (T) и пиковое использование ОЗУ (M).

современным решением для выравнивания нуклеотидных последовательностей на граф.

1.4.2 Тестирования метода выравнивания аминокислотных последовательностей

Метагеномные сборки, как правило, сильно фрагментированы из-за внутривидовых и межвидовых повторов, вызванных генами, переносимыми горизонтально, консервативными областями или близкородственными штаммами [62]. Фрагментация контигов может привести к фрагментации сегментов, кодирую-

	Aligner	MR, %	AI,%	T, ч:мин:сек	M, Гб
E. coli	GA $v1.0.16$	61.19	87	0:00:10	0.5
	GA-MUM $v1.0.16$	63.25	87	0:00:36	0.4
	SPAligner w/o AG	65.3	86	0:02:20	0.4
	SPAligner	67.65	86	0:02:33	0.4
S. cerevisiae	GA $v1.0.16$	57.4	83	0:00:16	0.4
	GA-MUM $v1.0.16$	58.37	83	0:00:37	0.3
	SPAligner w/o AG	55.43	83	0:00:55	0.3
	SPAligner	61.55	82	0:20:47	0.5
C. elegans	GA $v1.0.16$	60.09	86	0:00:57	1.4
	GA-MUM $v1.0.16$	64.06	86	0:02:53	1.2
	SPAligner w/o AG	60.01	87	0:05:56	0.8
	SPAligner	65.82	86	0:39:09	1.1

Таблица 3 — Результаты SPAligner и GraphAligner при выравнивании прочтений ONT на графы сборки из коротких прочтений, построенных с помощью SPAdes с размером k-меров равным 77. Каждый набор данных содержал 10 тыс. прочтений длиной более 2 тыс. нуклеотидов, за исключением набора данных *E. coli* ONT, который содержал приблизительно 7 тыс. прочтений соответствующей длины. Все прогоны выполнялись с использованием 16 потоков. Для преодоления высокой фрагментации графа сборки, возникающей из-за отсутсвия качественных доступных данных секвенирования *C. elegans* Bristol N2 Illumina, для получения графа сборки *C. elegans* использовали симулированные прочтения Illumina. Качество выравниваний измерялось с помощью четырех показателей: количество выровненных прочтений (MR), средняя похожесть выравниваний (AI), время выполнения (T) и пиковое использование ОЗУ (M).

щих гены в сборке генома, что может существенно повлиять на последующий функциональный анализ микробного сообщества [63; 64]. SPAligner отличается от других программ выравнивания последовательностей к графам тем, что позволяет искать выравнивания аминокислотных последовательностей к графам сборки и идентифицировать фрагментированные гены.

В этом разделе мы демонстрируем применение SPAligner для поиска генов устойчивости к антибиотикам в наборе метагеномных данных сточных вод. В эксперименте рассматривается только лучшее выравнивание белков среди всех произведенных выравниваний для конкретного запроса.

Бета-лактамазы представляют собой ферменты, продуцируемые различными бактериями, обеспечивающие устойчивость к широко используемым бета-лактамным антибиотикам. Идентификация генов бета-лактамаз в образцах

окружающей среды крайне важна для отслеживания распространения антибиотикорезистентности. В работе [65] авторы провели поиск белков устойчивости к противомикробным препаратам (AMR), включая бета-лактамазы, в наборах метагеномных данных сточных вод и городских поверхностных вод. Набор данных сточных вод больниц имеет наибольшее количество генов бета-лактамаз среди всех наборов данных, представленных в исследовании в соответствии с исходной статьей [65]. Набор данных (SRA SRR5997548), состоящий из 3,3 млн прочтений Illumina 2×250 нуклеотидов (средний размер вставки 350 нуклеотидов), был собран с помощью metaSPAdes $v3.12$ с настройками по умолчанию.

Для того чтобы показать, что SPAligner способен находить больше белковых последовательностей, чем при обработке контигов, мы проанализировали гены AMR, обнаруженные в собранных контигах, с помощью современного инструмента AMRFinder [66], а также гены AMR, обнаруженные SPAligner в соответствующем графе метагеномной сборки. И AMRFinder, и SPAligner выровняли 4810 белковых последовательностей из базы данных эталонных генов устойчивости к противомикробным препаратам [66]: частичные выравнивания, которые охватывают менее 75% длины последовательности запроса, были отфильтрованы, и были рассмотрены только выравнивания с похожестью более чем 90% . Окончательные последовательности, найденные с помощью AMRFinder и SPAligner, были сгруппированы с использованием односвязной кластеризации (single-linkage clustering) с порогом подобия 90%. Из полученных 89 кластеров один содержит только последовательности, найденные AMRFinder, а четыре кластера содержат только последовательности, найденные SPAligner. Последовательности из кластера, содержащего только выравниваниями инструмента AMRFinder, представляют пограничный случай и охватывают 90,7% длины запроса, в то время как SPAligner обнаружил немного более короткое выравнивание, занимающее 89,7% длины запроса, и оно было отфильтровано. Был проведен поиск представителей четырех кластеров, найденных SPAligner, в контигах с помощью tBLASTx [67]. Все они были лишь частично ($< 75\%$ длины запроса) покрыты контигами или имели низкую похожесть ($< 70\%$).

В то же время SPAligner реконструировал последовательности из семи (bla_{KPC}, bla_{CTX-M}, bla_{SHV}, bla_{TEM}, bla_{IMP}, bla_{VIM} и bla_{OXA}) из девяти семейств белков бета-лактамаз из первоначального исследования. Два оставшихся семейства имели низкое покрытие прочтениями ($\sim$ 20X) по сравнению

с другими семействами ($\sim$ 150X) [65]. На рисунке 1.3 показано выравнивание бета-лактамазы IMP-4, разбросанной по нескольким контигам.

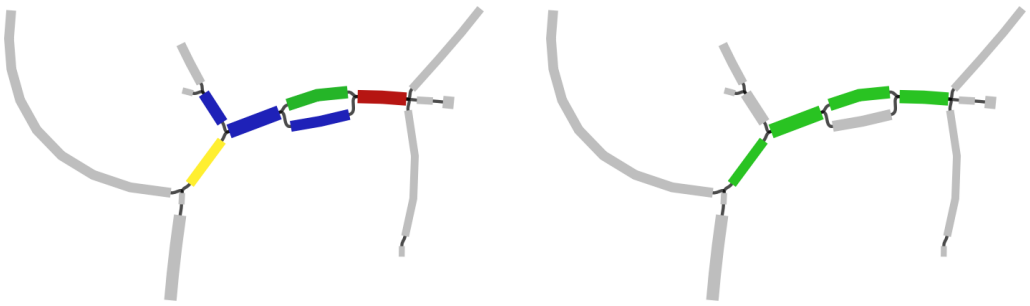

Рисунок 1.3 — Подграф метагеномной сборки с окрашенными контигами (Слева) и реконструированная фрагментированная последовательность белка бета-лактамазы (Справа). На левой панели рисунка представлен подграф сборки, визуализированный программой Bandage [68]. Различные цвета (синий, красный, зеленый и коричневый) представляют четыре разных контига сборки metaSPAdes. На правой панели представлен путь выравнивания (зеленые ребра) для гена IMP-4 в данном подграфе, который имеет длину 246 аминокислот и имеет идентичность выравнивания 98%.

1.5 Заключение

В этой главе мы продемонстрировали, что SPAligner является эффективным инструментом для нахождения выравниваний как длинных, подверженных ошибкам прочтений, так и аминокислотных последовательностей на графы сборки из коротких прочтений. Благодаря работе над SPAligner появились алгоритмы и подходы, которые помогут ускорить разработку алгоритмов выравнивания длинных прочтений на графы. В то же время стоит отметить, что данная реализация сильно зависит от кодовой базы SPAdes и ее ограничений (например, SPAligner можно применять только к графам де Брюйна).

Инструмент SPAligner конкурентоспособен по сравнению с другими программами для выравнивания длинных прочтений на графы сборки. Кроме того, SPAligner — единственный инструмент, который можно использовать для успешного восстановления полных белковых последовательностей непосредственно из графов метагеномной сборки. В то время как извлечение всех возможных аминокислотных последовательностей из графов сборки представляет

собой только начальный этап извлечения кодирующей последовательности белка, конвейер ORFograph [16], представленный в следующей главе, использует информацию из контигов и структуру графа для получения наиболее надежных кандидатов в белковые последовательности.

Глава 2. Поиск генов инсектицидных белков в графе сборки

2.1 Вступление

2.1.1 Инсектицидные белки *Bacillus thuringiensis*

Биопестициды обычно определяют как виды пестицидов, полученных из живых организмов или продуктов их жизнедеятельности [69]. Основными преимуществами биопестицидов перед синтетическими пестицидами являются:

- высокая специфичность (т. е. воздействует только на целевые и близкородственные организмы);
- эффективны в малых дозах (т. е. не оказывают негативного воздействия на окружающую среду, так как быстро растворяются);
- менее токсичны.

Растительные пестициды [69] — это биопестициды, которые широко используются в сельском хозяйстве. Обычно они представляют собой пестицидные белки из генов, встроенных в ДНК растений. Бактерии *Bacillus thuringiensis* (Bt) стали ценным источником пестицидных белков. Они был выделены из различных экосистем, таких как почва, вода, мертвые насекомые и ткани человека. Во время споруляции и стационарной фазы роста Bt продуцирует кристаллические (Cry) и цитотоксические (Cyt) инсектицидные белки, тогда как вегетативные инсектицидные белки (Vip) продуцируются во время вегетативной фазы роста. Было показано, что эти белки эффективны против различных насекомых, принадлежащих к отрядам чешуекрылых, жесткокрылых, полужесткокрылых и двукрылых [70], и широко используются в качестве ключевого элемента в стратегиях борьбы с насекомыми-вредителями [71]. Недавние исследования также показали, что некоторые из них активны против раковых клеток человека [72].

Активное использование Bt-инсектицидных белков привело к развитию устойчивости у насекомых по отношению к пестицидам, производимых на их основе [73]. За последние тридцать лет количество известных генов, кодирующих кристаллические инсектицидные белки Cry, увеличилось с 14 до более чем

700 [74]. Постоянная потребность в новых генах инсектицидных белков (ГИБ) требует новых методов идентификации ГИБ.

2.1.2 Поиск генов инсектицидных белков

Первоначально обнаружение генов Cry, Cyt и Vip проводилось с использованием методов, основанных на полимеразной цепной реакции (ПЦР) [75;76]. Системы E-PCR [77], PCR-RFLP [76] и PCR-SSCP [78] позволяли идентифицировать потенциально новые ГИБ. Их успех был основан на качественном отборе праймеров, а их попытки идентифицировать новые ГИБ в основном имели экспериментальный характер и ограничивались идентификацией трехдоменных Cry-генов [79].

Развитие технологий секвенирования открыло новые возможности разработки более дешевых и быстрых способов для обнаружения ГИБ. Инструменты определения генов GeneMark [80], Prodigal [81], Glimmer [82] и их метагеномные версии metaGeneMark [83], metaProdigal [84], metaGlimmer [85] осуществляют поиск генов в прочтениях и контигах. Для нахождения потенциальных ГИБ полученные гены-кандидаты можно сравнить с базами данных уже известных ГИБ. Кроме того, появились такие инструменты, как BtToxin_scanner и BtToxin_Digger [86], которые выполняют поиск именно Bt-токсинов в прочтениях и контигах. Они получают геномные последовательности и доменные СММ в качестве входных данных и предсказывают последовательности генов Cry, Cyt или Vip, используя методы машинного обучения. Однако все эти методы могут восстанавливать только гены, расположенные внутри одного контига.

В этой главе представлен конвейер ORFograph для извлечения открытых рамок считывания (ОРС) непосредственно из графов сборки. С помощью ORFograph ГИБ-кандидаты могут быть идентифицированы, даже если их части расположены на нескольких контигах, что позволяет выявить больше потенциальных ГИБ для последующих этапов обработки полного цикла поиска ГИБ и дает значительное преимущество по сравнению с предыдущими подходами.

ORFograph — это инструмент, который принимает в качестве входных данных граф сборки де Брюйна и набор доменных СММ или белковых последовательностей, ищет похожие ОРС в графе сборки и выполняет первоначальную

фильтрацию кандидатов. Таким образом, он может идентифицировать не только ГИБ, но и любые белки, подобные тем, которые предоставлены в качестве входных данных. В этой работе мы концентрируемся на поиске токсинов Cry, Cyt и Vip, поскольку их идентификация имеет практическое значение и поскольку они имеют более длинные последовательности по сравнению с последовательностями других белков (рис. 2.1). Таким образом их последовательности обычно более фрагментированы и часто расположены на нескольких контигах.

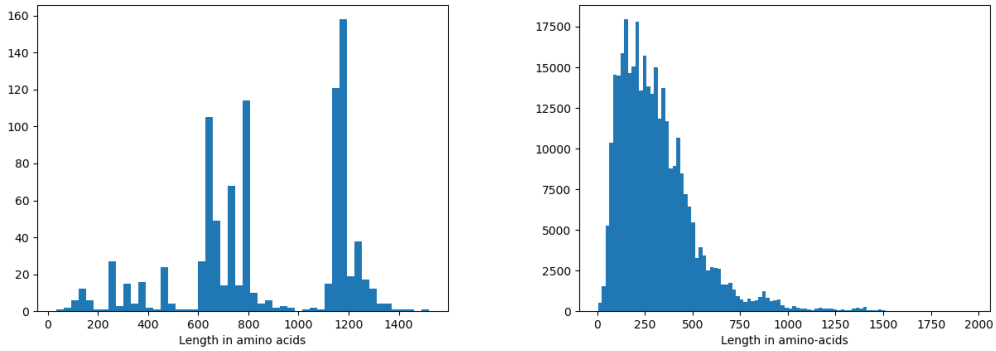

Рисунок 2.1 — Гистограммы длин 941 известного инсектицидного белка Cry и Vip (Слева), всех бактериальных генов из базы данных Uniprot [87] (Справа). Инсектицидные белки Cry и Vip характеризуются значительной долей (84%) генов, имеющих длину более 600 аминокислот. И наоборот, лишь небольшая часть (7%) всех генов из базы данных Uniprot имеет длину, превышающую 600 аминокислот.

2.2 Общая схема работы конвейера ORFograph

ORFograph принимает на вход набор доменных CMM, набор белковых последовательностей и граф сборки, построенный одним из современных сборщиков генома [38;41;42], которые строят графы де Брюйна из набора коротких прочтений. Он выводит ОРС, найденные в графе сборки, которые либо содержат домены, описанные входными CMM, либо похожи на входные последовательности белков. ORFograph объединяет эти последовательности в кластеры и находит наиболее надежного представителя для каждого кластера. Конвейер ORFograph можно разделить на четыре шага, описанных ниже (рис. 2.2).

Выравнивание известных инсектицидных белков/CMM на граф сборки. ORFograph использует инструмент PathRacer [23] для выравнивания

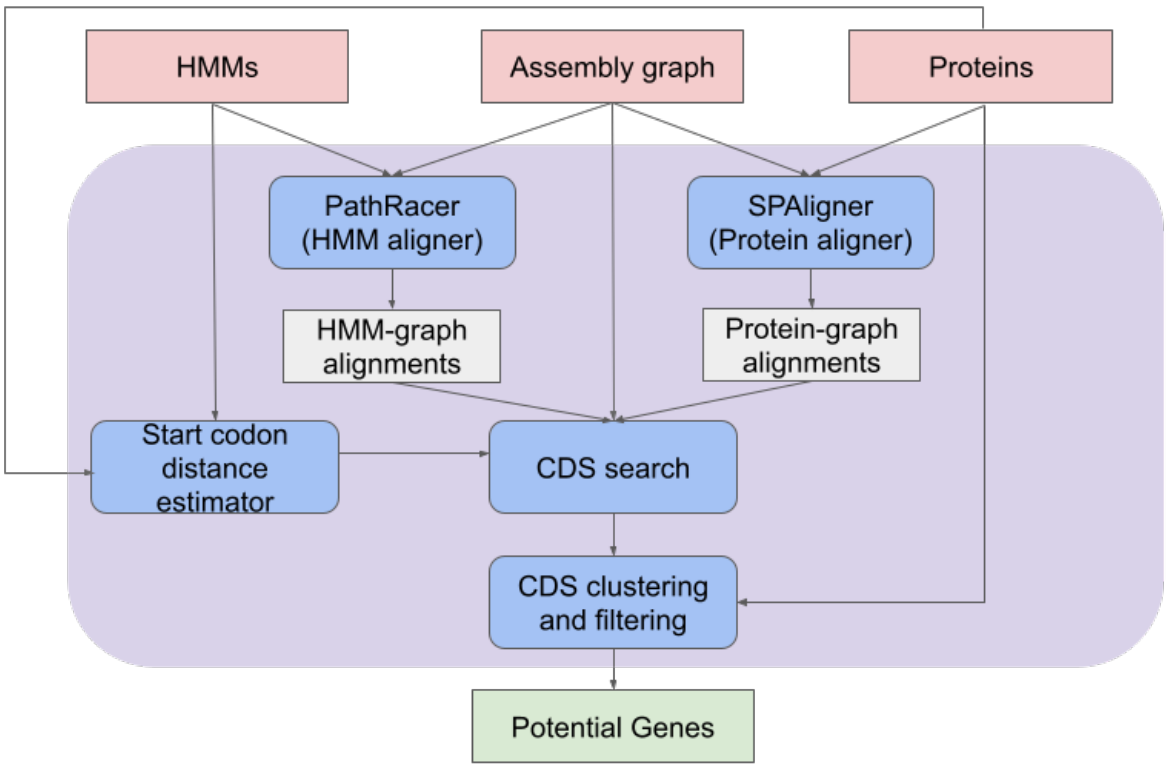

Рисунок 2.2 — Общая схема конвейера ORFograph. ORFograph принимает на вход граф сборки де Брюйна, CMM и белковые последовательности. На первом этапе он строит выравнивание CMM и белковых последовательностей на граф сборки, используя PathRacer и SPAligner, соответственно. После этого ORFograph выполняет первоначальный поиск и фильтрацию ОРС, используя якорные пути, сгенерированные на предыдущем шаге. Наконец, ORFograph выполняет кластеризацию полученных последовательностей ОРС.

CMM на граф сборки и сохраняет все выравнивания с e-value менее 10^{-9} и длиной более 90% от длины CMM. Входные белковые последовательности выравниваются на граф сборки с помощью инструмента SPAligner [15]. Эти выравнивания в дальнейшем рассматриваются как частичные ОРС или *якорные пути*, которые должны быть расширены до полных ОРС. Якорные пути могут быть представлены как частью ребра, так и путем в графе сборки, который проходит через одно или несколько ребер.

Поиск старт- и стоп-кодонов. На данном шаге ORFograh пытается расширить каждый якорный пути до полной ОРС. ORFograh рассматривает конец якорного пути в графе сборки и пытается найти все возможные его продолжения до ближайшего стоп-кодона. После этого ORFograph выполняет аналогичную процедуру, чтобы найти потенциальные стартовые кодоны, рассматривая все «обратные» пути от своего начала до ближайших стоп-кодонов в обратно-

комплиментарном направлении в графе и определяя предполагаемые стартовые кодоны между ними.

Генерация полных кодирующих последовательностей (CDS). Для каждого пары найденных старт- и стоп-кодонов для определенного якорного пути ORFograph выполняет полный перебор, чтобы сгенерировать все префиксные пути, ведущие от выбранного старт-кодона к началу якорного пути, и все суффиксные пути от конца якорного пути к выбранному стоп-кодону. Для ускорения алгоритма установлено ограничение: генерируется не более 1000 путей. Пути, полученные склейкой каждой пары пути префикса и пути суффикса с якорным путем, представляют собой пути потенциальных генов. В дальнейшем они фильтруются на основе информации о путях контигов. Затем итоговые пути транслируются в аминокислотные последовательности, и одинаковые аминокислотные последовательности удаляются.

Кластеризация потенциальных белков и выбор представителей. Поскольку количество потенциальных белковых последовательностей может быть очень большим, ORFograph группирует полученные последовательности и выбирает набор представителей в каждом кластере на основе их взаимного расположения в графе сборки.

В следующих четырех разделах данные четыре шага описаны более подробно.

2.3 Выравнивание ГИБ и доменных CMM к графу сборки

ORFograph строит якорные пути, выполняя выравнивание доменных CMM [23] или аминокислотных последовательностей белков [15] на граф сборки для обнаружения частичных выравниваний ОРС. При выравнивании CMM ORFograph часто находит большое количество коротких путей, поскольку CMM обычно описывают короткий домен внутри гена. ORFograph исключает такие короткие пути выравнивания из дальнейшего рассмотрения, если они представляют собой подпути более длинных (и, следовательно, более надежных) путей выравнивания с таким же сдвигом рамки считывания.

2.4 Поиск старт- и стоп-кодонов

Для каждого якорного пути ORFograph запускает обход в ширину [88] на графе сборки от начальной (конечной) позиций якорного пути, чтобы идентифицировать все потенциальные старт-(стоп-)кодоны. Во время работы алгоритма каждая новая посещенная вершина в графе обхода в ширину описывается не только своей позицией в графе сборки, но и сдвигом рамки считывания (длиной 1, 2 или 3), на котором она была достигнута, таким образом храня последний пройденный кодон. Алгоритм обхода в ширину не продолжает свой поиск от вершин, для которых последний пройденный кодон является стоп-кодоном.

2.5 Генерация полных кодирующих последовательностей (CDS)

Как только для каждого частичного выравнивания OPC определены наборы старт- и стоп-кодонов, ORFograph генерирует соответствующие CDS, представляющие собой потенциальные белки. Путь каждой CDS делится на три части: префикс, середину и суффикс, которые соответствуют пути от выбранного старт-кодона до крайнего левого положения якорного пути, якорному пути и пути от крайнего правого положения до выбранного стоп-кодона. ORFograph производит полный перебор путей для генерации всех потенциальных префиксов и суффиксов по отдельности и объединяет их с центральной частью, соответствующей последовательности якорного пути, чтобы сформировать набор полных путей, представляющих предполагаемые белки. Для ускорения генерации CDS ORFograph с помощью алгоритма поиска в глубину [88] предварительно вычисляет набор ребер, которые могут быть пройдены от найденных стартовых кодонов до начала якорного пути (и от конца якорного пути до потенциальных стоп-кодонов). ORFograph перебирает только эти ребра для создания путей префиксов и суффиксов. Далее ORFograph объединяет последовательности каждого префикса, якорного пути и каждого суффикса для создания предполагаемого белка. Поскольку количество потенциальных полноразмерных генов может быть большим, ORFograph фильтрует и ранжирует сконструированные последовательности на основе критериев, описанных ниже.

2.5.1 Анализ позиций СММ в последовательностях генов

На данном шаге ORFograph по входным последовательностям и СММ пытается оценить наиболее вероятное положение СММ относительно начала или конца последовательности гена. Для получения статистики по положению входных СММ в потенциальных генах был использован инструмент HMMer [89]. С помощью данного инструмента для каждой СММ было посчитано распределение расстояния от начала (конца) последовательности гена до позиции СММ. ORFograph использует это распределение для оценки наиболее вероятного расстояния от старт-(стоп-) кодона до положения СММ в последовательностях данного семейства белков и определяет наиболее вероятные префиксы (суффиксы).

2.5.2 Фильтрация предполагаемых белков, конфликтующих с путями контигов

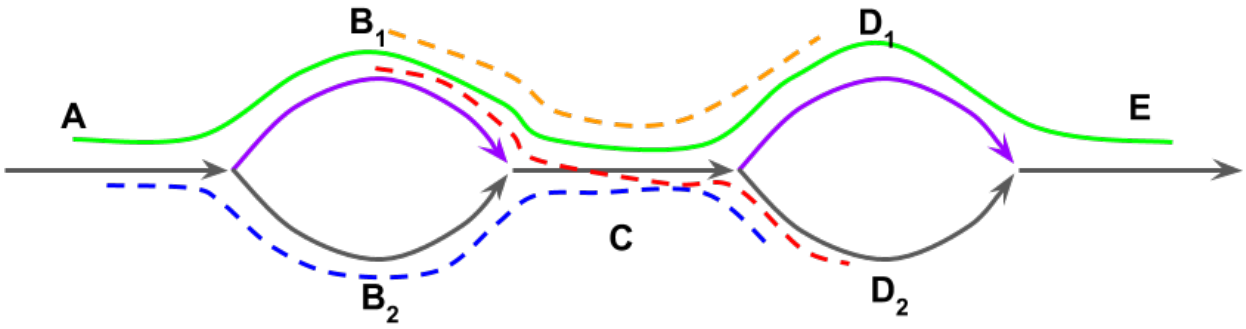

Рисунок 2.3 — Противоречащие и непротиворечащие пути в графе сборки. Есть два уникальных ребра B_1 и D_1. Все остальные ребра классифицируются как неуникальные ребра. Ребра B_2 и D_2 имеют длину меньше порога minLength, а ребра A, C и E имеют недопустимые для уникальных ребер входящие и/или выходящие степени вершин. Зеленый путь рассматривается как путь контига, а красный, синий и желтый пути — как пути потенциальных генов. Зеленый путь содержит оба уникальных ребра, а красный путь содержит только одно из них, поэтому они противоречат друг другу. Зеленый путь не пересекается с желтым путем по уникальным ребрам и содержит те же уникальные ребра, что и синий путь, поэтому в этих случаях противоречия нет.

ORFograph сравнивает путь каждого предполагаемого белка с набором путей контигов и отфильтровывает последовательности, чью пути конфликтуют с путями контигов. ORFograph идентифицирует так называемые *уникальные* ребра, последовательности которых встречаются в геноме один раз, используя простой набор эвристик. В частности, ребро считается уникальным, если его длина превышает пороговое значение *minLength* (по умолчанию: 300 нуклеотидов), исходящая степень его начальной вершины превышает 1, и входящая степень его конечной вершины превышает 1.

ORFograph рассматривает путь каждого предполагаемого белка и находит все непрерывные пути, которые пересекают этот путь хотя бы по одному уникальному ребру. Если при этом существует уникальное ребро, которое содержится только в пути предполагаемого белка, но не в пути контига, или, наоборот, только в пути контига, но не в пути рассматриваемого предполагаемого белка, то ORFograph отфильтровывает этот предполагаемый белок, поскольку он противоречит пути контига (рис. 2.3).

2.6 Кластеризация последовательностей и выбор представителей

Для сокращения количества кандидатов, ORFograph выполняет односвязную кластеризацию всех последовательностей, получившихся в финале предыдущей стадии. ORFograph группирует две последовательности вместе, если одна последовательность является подстрокой другой или если их процент идентичности превышает заданный порог (по умолчанию: 90%). Этот шаг помогает упростить анализ, группируя очень похожие последовательности в кластеры.

ORFograph выводит набор представителей для каждого кластера, следуя набору правил. Во-первых, он идентифицирует все ребра в графе сборки, которые принадлежат хотя бы одному пути последовательности кластера и имеют длину, превышающую 1000 нуклеотидов. Затем для каждого пути в кластере ORFograph рассматривает триплет информации, состоящий из вероятности расстояния до стартового кодона, среднего покрытия пути прочтениями (т. е. среднего количества прочтений, которые поддерживают каждый нуклеотид в последовательности пути) и длины пути. Он сравнивает триплеты разных путей и считает путь P более надежным, чем другой путь P', если его триплет лек-

сикографически больше. ORFograph сортирует белковые последовательности в лексикографически неубывающем порядке их триплетов и добавляет потенциальный белок в список представителей, если он содержит длинное ребро, которое не было представлено ни в одной из ранее рассмотренных последовательностей из списка.

2.7 Результаты

2.7.1 Общая схема тестирования конвейера ORFograph

Мы извлекли 941 последовательность ГИБ (811 Cry и Cyt токсинов, а также 130 Vip токсинов) из базы данных NCBI по их инвентарным номерам, указанным в номенклатурном списке Bt [90].

Кроме того, мы загрузили 66 СММ, представляющих домены, обычно встречающиеся в ГИБ, из базы данных Pfam [91], с количеством скрытых состояний от 25 до 350.

Для оценки производительности ORFograph, мы провели тестирование ORFograph на наборах симулированных метагеномных данных, на наборах данных из реальных изолированных бактерий Bt и на одном наборе метагеномных данных, содержащих образцы Bt. Все наборы данных находятся в открытом доступе по ссылке https://figshare.com/s/f20604a5333bbe4514c9. Мы использовали SPAdes [38] или metaSPAdes [92] (для наборов метагеномных данных) для построения графов сборки, а затем запустили конвейер ORFograph, запустив SPAligner на 941 последовательности ГИБ и PathRacer на 66 загруженных СММ.

Известно, что гены Cry и Cyt обычно располагаются на плазмидах бактерий Bt. Мы попытались использовать plasmidSPAdes [93] для извлечения полных плазмидных последовательностей, и, как следствие, полных последовательностей генов. Было обнаружено, что у этого метода есть ограничения (обсуждаемые в [16]), которые не позволяют нам использовать его в нашем тестировании.

CryProcessor [94] — это инструмент, разработанный для обнаружения Cry-белков, который использует PathRacer [23] для извлечения последовательно-

стей Cry-генов из графов сборки. Однако он имеет ограниченную функциональность, поскольку может идентифицировать только трехдоменные токсины Cry и не подходит для общего обнаружения генов из графов сборки. Не представляется возможным сравнить CryProcessor с ORFograph, и из-за того, что CryProcessor рассматривает только последовательности, идентифицированные PathRacer, без учета путей ОРС, содержащих эти последовательности.

2.7.2 Симулированные наборы данных

Для тестирования ORFograph на симулированных данных, мы использовали набор данных TOY, смоделированный на основе 30 геномов [95]. Набор данных содержит 74 млн. парных прочтений длиной 100 нуклеотидов (средний размер вставки 180 нуклеотидов). Ошибки в прочтениях моделировались на основе прочтений Illumina HiSeq.

Мы взяли две полные сборки геномов Bt из базы данных NCBI: BtA (NCBI ID=GCF_002025105) и BtB (NCBI ID=GCF_001598095) и сопоставили их последовательности с 941 последовательностью ГИБ. Мы исключили плазмиды с токсиноподобными последовательностями, чтобы контролировать состав Cry-генов в плазмидах и варьировать сложность наборов данных. Мы исключили плазмиду NZ_CP020005.1 из BtA и плазмиды NZ_CP014852.1, NZ_CP014853.1 из BtB, содержащие частичное или полное выравнивание токсинов из базы данных ГИБ, сформировав симулированные наборы данных BtA_{pure} и BtB_{pure}.

Мы варьировали сложность задачи идентификации ГИБ, добавляя в BtA_{pure} и BtB_{pure} различное количество CDS Cry-генов (0, 1, 2, 6) с различным сходством между CDS, которые варьировались от идентичности в 45% до идентичности в 92% (таблицы 4 и 5).

Мы сформировали пять наборов данных, смешивая набор данных TOY с прочтениями Illumina, симулированными программой Art-Simulator [96] с параметрами «`hs20 -l 100 -f 20 -m 180 -s 10`» из BtA_{pure} и BtB_{pure}, обогащенными кодирующими последовательностями ГИБ (Таблица 4).

Мы запустили ORFograph на графах сборки и контигах, представленных после запуска сборщика SPAdes v3.14.1. СММ доменов в данном тестировании

Название набора данных	Описание
BtA_{pure}	BtA без плазмид NZ_CP020005.1
BtB_{pure}	BtB без плазмид NZ_CP014852.1, NZ_CP014853.1
BtA: Cry1Aa1	BtA_{pure} с добавлением Cry1Aa1 CDS в NZ_CP020003.1 на позицию 27668
BtA: Cry1Aa1 + Cry1Ab1	*BtA*: Cry1Aa1 с добавлением Cry1Ab1 CDS в NZ_CP020004.1 на позицию 24976
BtA: Cry1Aa1 + Cry1Ac1	*BtA*: Cry1Aa1 с добавлением Cry1Ac1 CDS в NZ_CP020004.1 на позицию 12488
BtA: Cry1Aa1 + Cry1Ab1 + Cry1Ac1	*BtA*: Cry1Aa1 + Cry1Ab1 с добавлением Cry1Ac1 CDS в NZ_CP020004.1 на позицию 12488
BtB: Cry1Ca1 + Cry2Aa1 + Cry2Ac1	BtB_{pure} с добавлением Cry1Ca1 CDS в NZ_CP020003.1 на позицию 14716, Cry2Aa1 CDS в NZ_CP014848.1 на позицию 8614, and Cry2Ac1 CDS в NZ_CP014849.1 на позицию 19166

Таблица 4 — Информация о семи симулированных наборах данных.

были исключены из запуска ORFograph, чтобы снизить количество ложноположительных срабатываний, так как многие общедоступные домены встречаются в широком спектре токсинов. Таблица 6 иллюстрирует результаты ORFograph.

В наборе данных NoCry ORFograph не определил ни одного ГИБ. В наборе данных Cry1Aa1 с одним геном Cry данных ген был расположен на одном контиге, поэтому ORFograph легко идентифицировал этот ген.

При рассмотрении наборов данных с двумя Cry-генами, SPAdes не смог собрать области, где были расположены эти гены, в то время как ORFograph обнаружил все белки Cry. В наборе данных Cry1Aa1+Cry1Ab1 с двумя похожими кодирующими последовательностями ORFograph обнаружил оба гена, но также две химерные последовательности. В случае набора данных с двумя отдален-

Названия Cry-генов (ID в NCBI)	Названия Cry-генов					
	Cry1Aa1	Cry1Ab1	Cry1Ac1	Cry1Ca1	Cry2Aa1	Cry2Ab1
Cry1Aa1 (aaa22353)	100	92	88	77	45	45
Cry1Ab1 (aaa22330)		100	88	76	45	45
Cry1Ac1 (aaa22331)			100	75	45	45
Cry1Ca1 (caa30396)				100	45	46
Cry2Aa1 (aaa22335)					100	89
Cry2Ab1 (aaa22342)						100

Таблица 5 — Информация о процентной идентичности между шестью выбранными Cry-белками. Каждая ячейка отображает процент идентичности между парами генов Cry1Aa1, Cry1Ab1, Cry1Ac1, Cry1Ca1, Cry2Aa1, и Cry2Ab1.

но похожими Cry-генами, ORFograph разделил их, но также идентифицировал подстроку Cry1Aa1 как ОРС.

В случае, когда в метагеноме представлено шесть Cry-генов, ORFograph идентифицировал четыре кластера ORF: три из них представляют CDS Cry1-подобных белков, а один представляет Cry2-подобные белки.

Анализ результатов ORFograph на симулированных наборах данных продемонстрировал, что ORFograph успешно идентифицирует гены Bt, но может «склеить» очень похожие последовательности. К таким результатам могут привести следующие условия: (i) ограниченная информация из контигов, или (ii) несовершенная процедуры отбора репрезентативных последовательностей.

Название набора данных	Набор данных TOY +симулированные прочтения из:	Cry-токсины несобранные SPAdes	Результаты ORFograph
NoCry	BtA_{pure} и BtB_{pure}	-	Не найдено ни одного Cry-гена
Cry1Aa1	*BtA*: Cry1Aa1	-	Найдена одна CDS, похожая на Cry1A
Cry1Aa1 + Cry1Ab1	*BtA*: Cry1Aa1 + Cry1Ab1	Cry1Aa1, Cry1Ab1	Четыре последовательности, похожих на CDS Cry1Aa1 и Cry1Ab1
Cry1Aa1 + Cry1Ac1	BtA: Cry1Aa1 + Cry1Ac1	Cry1Aa1, Cry1Ac1	Три последовательности: одна идентична CDS Cry1Ac1, две другие имеют высокий процент идентичности с Cry1Aa1
AllCry	*BtA*: Cry1Aa1 + Cry1Ab1 + Cry1Ac1 and *BtB*: Cry1Ca1 + Cry2Aa1 + Cry2Ac1	Cry1Aa1, Cry1Ab1, Cry1Ac1, Cry1Ca1	Четыре кластера OPC: один кластер содержит последовательности похожие на Cry2-гены и три других кластера содержат последовательности похожие на гены из семейства Cry1.

Таблица 6 — Результаты тестирования конвейера ORFograph на пяти симулированных наборах данных разной сложности. В последнем наборе данных все последовательности первого кластера располагаются на одном контиге, во всех остальных кластерах последовательности генов располагаются на разных контигах. Второй кластер содержит ген Cry1Aa1 и ген похожий на Cry1Ab1, третий кластер содержит последовательности Cry1Aa1 и последовательность похожую на Cry1Ac1, и четвертый кластер содержит последовательность гена Cry1Ca.

2.7.3 Изолированные образцы бактерий из базы данных NCBI, содержащие ГИБ

Из базы данных NCBI были извлечены 2749 наборов данных. При помощи поиска ГИБ посредством инструмента Diamond [97] в извлеченных наборах данных были обнаружили наборы, которые потенциально могут содержать гены Cry, Cyt или Vip токсинов. Набор данных отбирался для дальнейшего анализа, если существует ГИБ, покрытый прочтениями рассматриваемого набора данных на не меньше чем 50% его длины с идентичностью выравнивания не меньше 80%.

Мы использовали ORFograph для анализа 342 графов сборки, полученных в результате запуска SPAdes. Среди этих наборов данных ORFograph идентифицировал 72, которые содержали выравнивания ГИБ, распределенные по нескольким контигам. В результате ORFograph нашел 419 потенциальных новых генов в выбранных наборах данных.

Центр ресурсов бактериальных пестицидных белков (Bacterial Pesticidal Protein Resource Center) обычно рассматривает только последовательности с аминокислотной идентичностью (AAI) не менее 95% к существующим ГИБ в качестве кандидатов ГИБ. Однако мы решили использовать менее строгий критерий и рассмотрели всех представителей с идентичностью не менее 90% к известным ГИБ, которые дополнительно не присутствовали в базе данных белков BLAST. В результате такой фильтрации получилось 232 кандидата.

Большая часть надежных ГИБ (77 из 232) была обнаружена в наборах данных SRR6238356, SRR6238345 и SRR4996216. Эти наборы данных были до этого проанализированы с помощью BtToxin_scanner в работе [101], в которой авторы исследовали перенос Cry-насыщенных плазмид между разными видами.

Наш анализ этих наборов данных подтверждает наличие ГИБ, обнаруженных в отдельных контигах, о которых сообщается в [101]. Однако BtToxin_scanner не смог идентифицировать 59 новых кандидатов в ГИБ, обнаруженных с помощью ORFograph (см. Таблицу S4 в [101]). Кроме того, два потенциальных ГИБ, идентифицированных BtToxin_scanner, были лишь частичными фрагментами ГИБ, обнаруженными ORFograph.

Набор данных SRR4996216 представляет собой прочтения Illumina из *B. thuringiensis* serovar aizawai штамм HD-133 и содержит 17 идентифи-

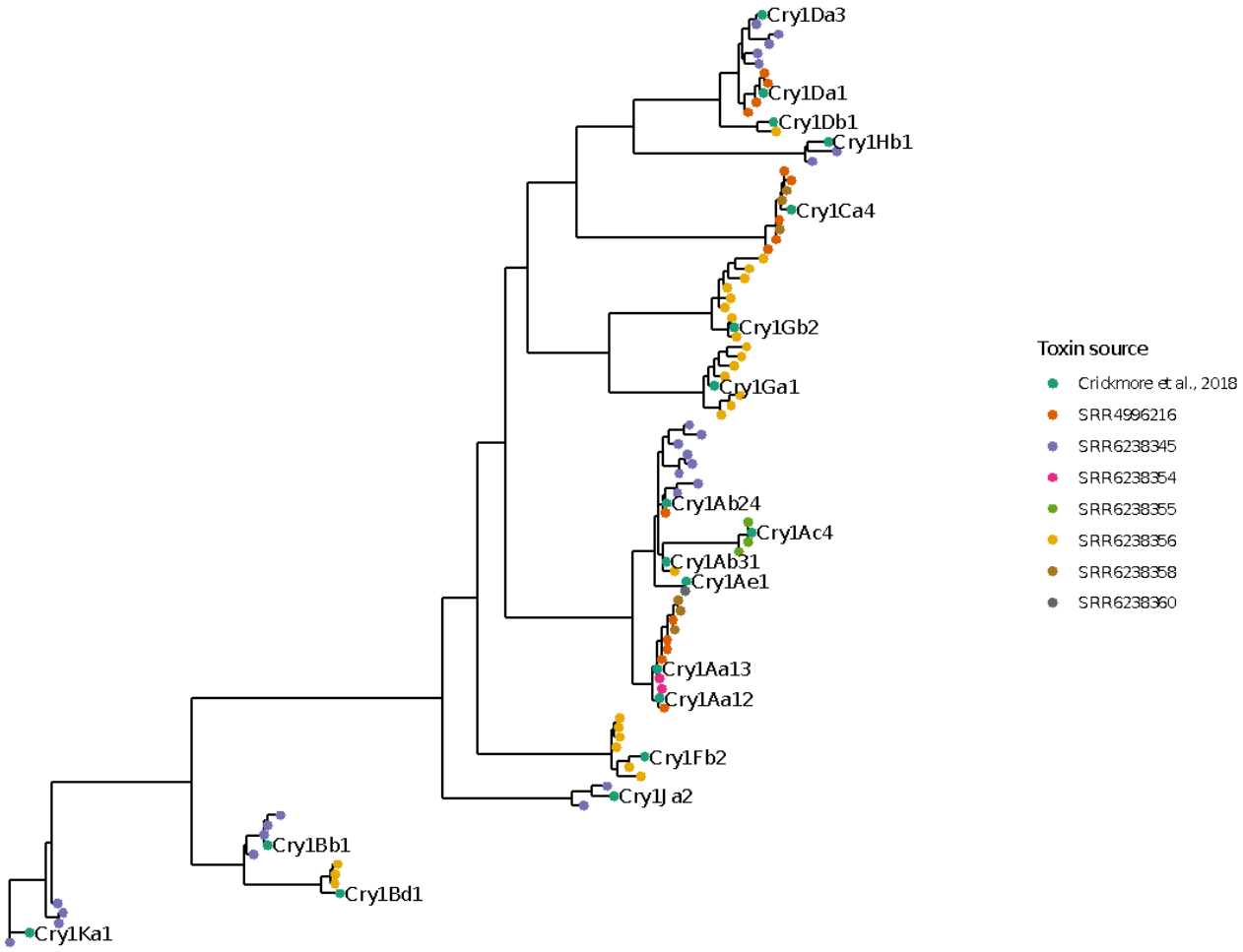

Рисунок 2.4 — Разнообразие Cry1-последовательностей, идентифицированных в данных изолированных бактерий. ORFograph использовался для идентификации предполагаемых новых ГИБ Cry1, которые затем были сопоставлены с последовательностями из номенклатурного списка Bt [98] с использованием Muscle v3.8.31 [99]. Филогения методом максимального правдоподобия для визуализации отношений между последовательностями Cry1 была построена с использованием FastTree v2.1.10 [100].

цированных ГИБ. В оригинальном исследовании [102] сборка проводилась с использованием конвейера A5-miseq [103], а аннотация генома выполнялась с использованием Prokka [104] и конвейера аннотации прокариотического генома (Prokaryotic Genome Annotation Pipeline) NCBI [105]. В первоначальном исследовании были идентифицированы Cry1Aa, Cry1Ab, Cry1Ca, Cry1Da, Cry1Ia, Cry2Ab и Cry9Ea, включая многие частичные или явно фрагментированные гены. Напротив, с помощью ORFograph мы идентифицировали 16 полных ГИБ, включая новые варианты Cry1Aa, Cry1b, Cry1Ca и Cry1Da.

Таким образом было показано, что ORFograph может значительно расширить набор ГИБ, обнаруженных в предыдущих исследованиях, однако по срав-

нению с анализом симулированных наборов данных этот анализ нельзя назвать тестированием, поскольку мы идентифицировали совершенно новые последовательности.

На рисунке 2.4 представлено сравнение предполагаемых новых ГИБ с существующими аннотациями в базе данных ГИБ, иллюстрируя значение такого подхода в целом и данного инструмента, ORFograph, в частности для новых обнаружения ГИБ.

2.7.4 Метагеномные данные метро Нью-Йорка

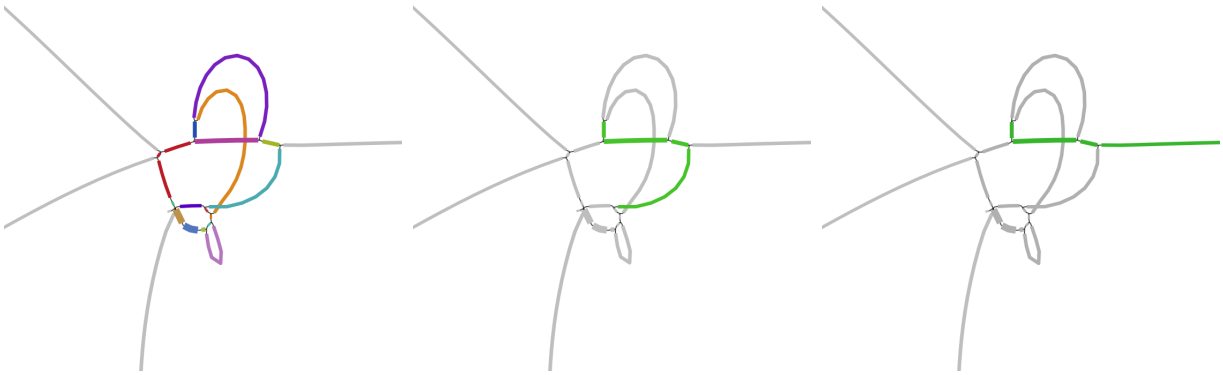

Рисунок 2.5 — Подграф графа метагеномной сборки с выравниваниями Cry4Aa и Cry4Ba генов. На левой панели представлены пути контигов в подграфе. Путь каждого контига отображается ребрами одного цвета. На центральной панели представлен путь гена Cry4Aa. На правой панели представлен путь гена Cry4Ba. Все подграфы визуализировались с помощью инструмента Bandage [68].

В исследовании [106] авторы исследовали микробное разнообразие метро Нью-Йорка, анализируя наборы прочтений из нескольких метагеномных образцов. Позже в другом исследовании [107] авторы воспользовались одним из полученных в ходе исследования [106] наборов данных (с номером SRR1748627) и собрали геном *B. thuringiensis* UBA3967. Мы запустили ORFograph на выбранном наборе данных с целью поиска Cry и Cyt-генов, которые потенциально могут возникнуть в *B. thuringiensis* UBA3967.

ORFograph обнаружил 48 кластеров потенциальных ГИБ в графе сборки, сгенерированном сборщиком metaSPAdes v3.12 [92]. Большинство представителей кластеров имеют идентичность 99% с тиоредоксинами, металлофосфуэс-

теразами и дисульфидными редуктазами, которые были получены из выравнивания домена CMM. Количество таких ложных срабатываний может быть уменьшено путем создания более специфичных CMM для Cry и Cyt-генов.

Тем не менее ORFograph также идентифицировал Cry60AA, Cry60BA, Cry11Aa, Cry10Aa, Cry4Aa и два гена Cry4Ba, а также три Cyt-токсина. При этом Cry4-гены были разбросаны по нескольким контигам (рис. 2.5), таким образом без ORFograph их идентификация была бы невозможна.

2.8 Заключение

В этой главе мы продемонстрировали, что анализ графов сборки помогает обнаружить последовательности генов, которые не могут быть идентифицированы традиционными методами, работающими исключительно с контигами. ORFograph использует инструменты для выравнивания CMM и последовательностей на граф сборки, далее обрабатывает их результат, создавая набор генов-кандидатов, которые согласуются со структурой графа сборки и путями контигов.

ORFograph является единственным конвейером, способным извлекать произвольные последовательности генов непосредственно из графа сборки на основе доменов CMM или последовательностей генов. Таким образом, ORFograph может ускорить идентификацию важных генов в наборах данных секвенирования.

Глава 3. Автоматический анализ сборок центромерных последовательностей

3.1 Вступление

3.1.1 Центромеры человека и их структура

Центромера определяется как участок хромосомы, где происходит расщепление ДНК в процессе клеточного деления с помощью специального белкового комплекса *кинетохора*. Центромера играет ключевую роль в сегрегации хромосом и наследовании во всех геномах позвоночных, поэтому изменение ее структуры может непосредственно влиять на генетические нарушения. Изучение структуры центромер крайне важно, так как это может раскрыть принципы эволюции центромер и помочь бороться с множеством болезней, в том числе с раком и с бесплодием [108].

Далее в этой работе мы рассматриваем только «живые», или «активные», центромеры, т. е. подпоследовательность каждой центромеры, связываемую с кинетохором [20]. Такие центромерные последовательности представляют собой сверхдлинные тандемные повторы (extralong tandem repeats), которые охватывают около 3% последовательности генома человека [109]. Они образованы последовательностями, называемыми *повторами высокого порядка* (ПВП), которые повторяются тысячи раз внутри одной центромеры (рис. 3.1). В свою очередь ПВП образованы более мелкими блоками, называемыми *альфа-сателлитами* или *мономерами*. Длина каждого мономера составляет около 171 нуклеотида. Сходство между двумя мономерами в пределах одного ПВП составляет около 50—90%. В пределах одной центромеры ПВП состоят из одного набора мономеров, идущих в определенном порядке, при этом от центромеры к центромере наборы мономеров отличаются. Например, подавляющее большинство ПВП в центромере хромосомы X (cenX) [12; 110] состоит из двенадцати мономеров в определенном порядке, а центромера хромосомы 8 представляет собой смесь двух ПВП, ПВП из 7 мономеров и ПВП из 11 мономеров (рис. 3.2).

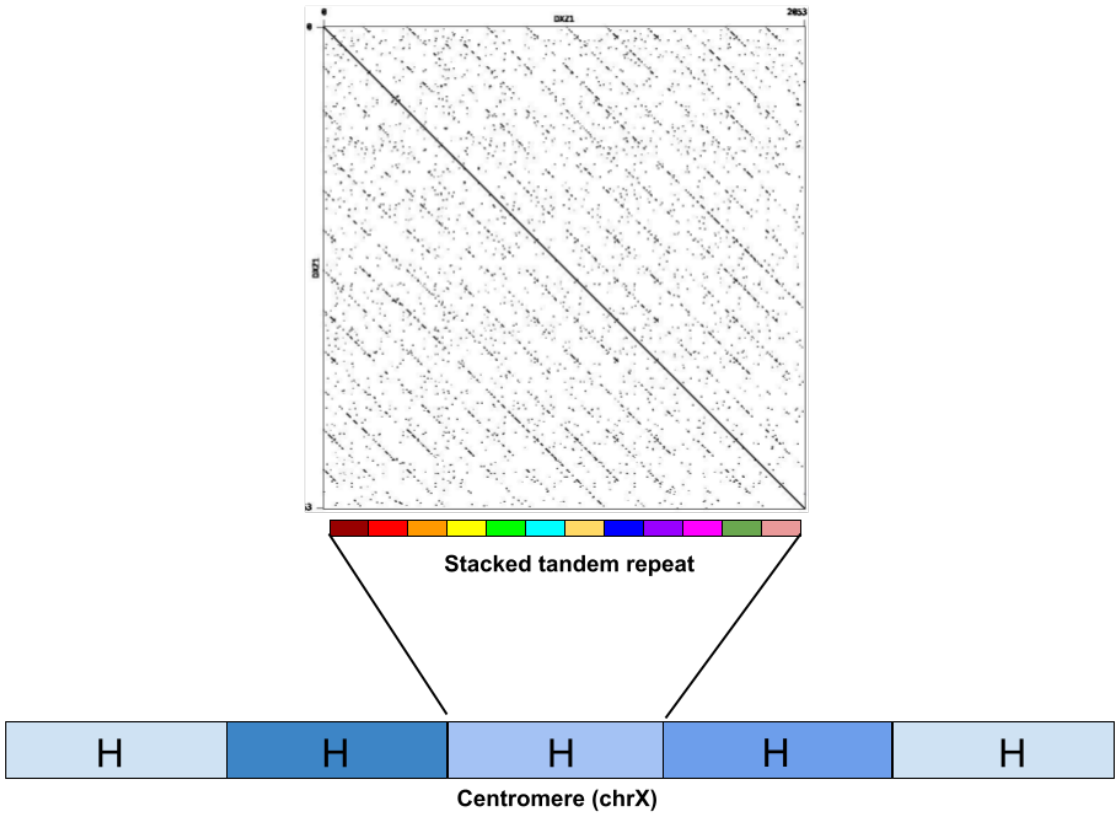

Рисунок 3.1 — Устройство центромеры хромосомы X. Сборка центромеры хромосомы X (cenX), представленная в работе [12], состоит из 18 100 мономеров, каждый длиной около 171 нуклеотидов (данная статистика практически не изменилась в итоговой сборке, представленной T2T [9]). Мономеры организованы в юниты примерно по 1 500 нуклеотидов, пять из которых изображены на рисунке оттенками синего, чтобы проиллюстрировать небольшие вариации. Каждый юнит представляет собой повтор высшего порядка (ПВП), или сложенный тандемный повтор (stacked tandem repeat), образованный последовательностью различных мономеров. Большинство юнитов в cenX представляют собой канонические ПВП, состоящие из двенадцати мономеров, показанных на рисунке двенадцатью разными цветами. В то время как канонические ПВП похожи на 95-100%, мономеры похожи только на 65-88%. Кроме того, cenX имеет небольшое количество частичных ПВП с различным количеством мономеров. На рисунке в верхней части текста изображен точечный график (англ. dot plot), показывающий самоподобие нуклеотидной последовательности канонического ПВП.

Также иногда центромерные последовательности содержат подстроки, не образованные мономерами (например, вставки транспозонов и LINE-элементы).

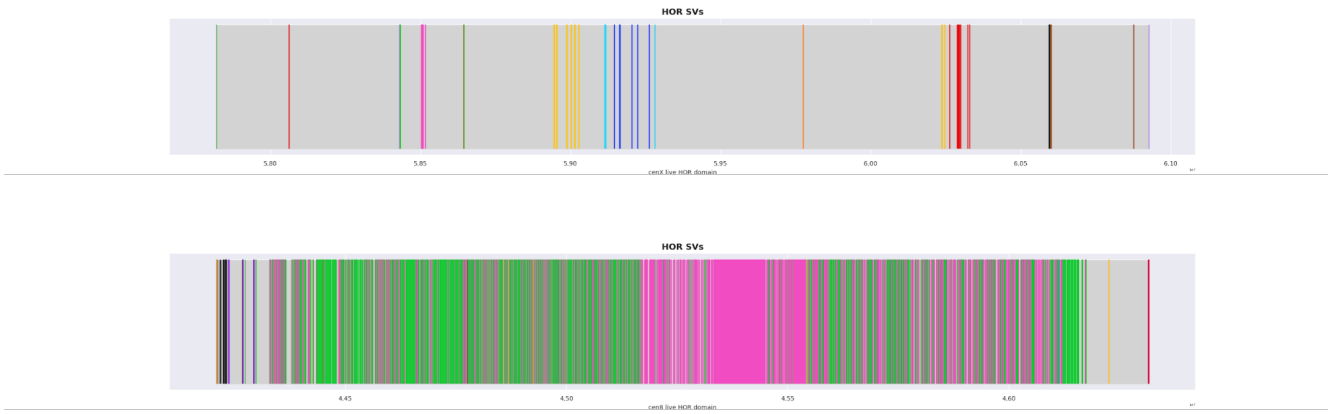

Рисунок 3.2 — Разложения на ПВП для центромеры хромосомы X и центромеры хромосомы 8. На панели сверху представлено разложение на ПВП для центромеры хромосомы X. ПВП, состоящие из 12 мономеров, представлены вертикальными линиями серого цвета. LINE-элемент окрашен в черный цвет. Другие цвета отображают ПВП иной структуры. На панели снизу представлено разложение на ПВП для центромеры хромосомы 8. Часто встречающийся ПВП из 7 мономеров окрашен в пурпурный цвет, а частый ПВП из 11 мономеров окрашен в зеленый цвет.

3.1.2 Постулат эволюции центромер

Будем называть мономер *частым*, если количество его выравниваний на центромерную последовательность превышает заранее заданный порог, и *редким* иначе. Мономер называется *гибридным*, если он является конкатенацией двух и более частых мономеров. Впервые наличие гибридных мономеров было показано в работах [12;13;17], также было выдвинуто предположение, что именно так образуются новые мономеры. *Каноническим* ПВП центромеры будет называться циклическая последовательность частых негибридных мономеров, которая эволюционировала с течением времени в текущую последовательность центромеры. К сожалению, на данный момент нет более конструктивного определения *канонического* ПВП.

Одной из самых сложных и одновременно важных задач, связанных с центромерами, является задача понимания того, как возникали и развивались центромерные последовательности [13; 111–113]. Несмотря на то, что на данных момент нет общепринятой теории эволюции центромер, есть набор положений, которые отражают текущий взгляд на данную теорию. В нашей работе мы

объединили положения, важные для разработки инструментов анализа последовательностей центромер, в постулат эволюции центромер (англ. Centromere Evolution Postulate).

Постулат эволюции центромер, впервые описанный в работе [19], включает в себя четыре пункта:

- Каждая существующая центромера человека произошла от одного канонического ПВП, образованного k *различными* мономерами.
- Каждый частый негибридный мономер в центромере соответствует ровно одному мономеру в каноническом ПВП.
- Каждый гибридный мономер произошел от конкатенации двух и более мономеров канонического ПВП.
- Последовательность центромеры состоит из юнитов канонического ПВП, юнитов *частичных* ПВП, являющихся подстроками канонического ПВП, и *вырожденных* ПВП, состоящих из одного гибридного мономера.

3.1.3 Автоматическая аннотация центромер человека

В 2001 проект «Геном человека» предоставил первую и наиболее полную на тот момент сборку генома человека [8]. Однако некоторые сложные части генома, в том числе центромеры, остались несобранными из-за отсутствия подходящих технологий секвенирования [9]. Несмотря на то, что до недавнего времени большинство попыток анализа центромерных последовательностей были полуручными, сосредоточенными на конкретных экспериментальных данных, такие несистематичные исследования значительно расширили наше понимание структуры и эволюции центромер, начиная с базового понимания того, что альфа-сателлитные мономеры представляют собой центромерные области и структурированы в более сложные повторы высшего порядка (ПВП) [114] до построения искусственно сконструированных центромерных последовательностей [13; 25] и понимания важности центромерной структуры для гомеостаза и стабильности генома [115].

Прогресс в технологиях секвенирования третьего поколения открыл возможности для разрешения сверхдлинных тандемных повторов, включая цен-

тромерные последовательности [9; 12]. В апреле 2022 консорциум Telomere-to-Telomere (T2T) представил новую референсную сборку генома человека, в которой были собраны все центромерные последовательности [9]. Эта работа стала важной вехой и вывела исследование центромер на новый уровень. Стало доступно больше качественных центромерных данных и научному сообществу стали нужны новые инструменты для их автоматического анализа.

В этой главе будет описан набор инструментов StringDecomposer [17], CentromereArchitect [18] и HORmon [19] для автоматического анализа последовательности центромер, которые были разработаны в рамках работы автора в консорциуме T2T. Кроме того, в этой главе обсуждаются результаты, полученные с использованием этих инструментов, которые отражают новое понимание структуры центромер.

Первостепенная задача, которую нужно было решить в рамках работы над анализом центромер, заключалась в том, чтобы эффективно и точно представить длинные, подверженные ошибкам прочтения и сборки центромер в виде цепочки уже известных мономеров. Далее данная задача будет называться задачей разложения строки (англ. String Decomposition Problem). Представление прочтений в виде строки мономеров крайне важно для работы некоторых сборщиков [12; 116] и для проведения сравнительного анализа в эволюционных исследованиях, если сборка центромер не получена или получена лишь частично [110]. CentroFlye [12], сборщик центромер из ультрадлинных, но подверженных ошибкам прочтений, использует прочтения, представленные в мономерном алфавите, для построения графа сборки, что позволяет ему игнорировать ошибки секвенирования на уровне нуклеотидов. Очень важно, чтобы такое представление было точным, чтобы избежать ошибок при построении графа сборки, и для этой цели был разработан инструмент StringDecomposer [17], в котором реализован алгоритм точного решения задачи разложения строки.

Недавно было показано, что обычные программы выравнивания последовательностей, такие как Minimap2 [22], дают плохие результаты в областях, содержащих большое количество повторов, и поэтому не могут использоваться для решения задачи разложения строки [117; 118].

Существуют более специализированные инструменты, такие как TandemRepeatsFinder [119] и Noise-Cancelling Repeat Finder [117], но и они лишь частично решают проблему. Инструмент TandemRepeatsFinder (TRF) [119] обрабатывает строку, содержащую тандемный повтор, без каких-либо предва-

рительных сведений о повторяющейся подстроке, т. е. мономере, или его длине. TRF выводит консенсус всех вхождений мономера и местоположение каждого такого вхождения во входной строке. При этом TRF не дает возможности идентифицировать различные мономеры, таким образом его результат требует дальнейшей обработки для решения задачи разложения строки. Другая сложность в обработке результата TRF заключается в том, что он сообщает о различных циклических сдвигах мономера для каждой обработанной центромерной последовательности. Инструмент Noise-Cancelling Repeat Finder (NCRF) [117] принимает в качестве входных данных строку, содержащую тандемный повтор, и один мономер, и разбивает строку на вхождения этого мономера. Таким образом, он не предназначен для работы с несколькими мономерами.

Инструменты Alpha-CENTAURI [120] и PERCON [121] были специально разработаны для работы с сателлитными мономерами. Задача разложения строки решается PERCON путем сравнения содержания октануклеотидов в предлагаемой последовательности с содержанием октануклеотидов во всех входных мономерах. Однако инструмент PERCON сложно протестировать, поскольку он совместим только с операционной системой Windows. Инструмент Alpha-CENTAURI [120] решает более общую задачу. Но при этом на первом этапе Alpha-CENTAURI выравнивает предварительно обученную CMM консенсусного мономера на все входящие строки, используя HMMer [89], таким образом получая начальные и конечные позиции для каждого выравнивания мономера на прочтение. Используя набор известных мономеров, каждому выравниванию мономера можно сопоставить мономер с наибольшей идентичностью. Полученное разложение можно сравнить с любым другим.

Естественным расширением инструмента StringDecomposer является инструмент, предоставляющий автоматическое решение для задачи полной аннотации центромер, т. е. инструмент, который принимает в качестве входных данных набор центромер или прочтений генома человека и консенсусный мономер, и выводит набор соответствующих мономеров (решение задачи извлечения мономеров), набор соответствующих ПВП (решение задачи извлечения ПВП) и разложение входных последовательностей на извлеченные мономеры и ПВП. Примечательно, что правильный ответ на все эти вопросы зависит от нашего определения исходной последовательности ПВП и от того, как менялась центромерная последовательность в процессе эволюции.

Большинство предыдущих подходов к решению задачи полной аннотации центромеры были полуручными и не предназначались для ответа на вопросы, связанные с эволюцией. Такие ранее разработанные подходы, как ColorHOR [122] и HORdetect [123], пытались извлечь ПВП из не полностью собранных коротких последовательностей генома, и из-за ограничений технологий секвенирования их основной задачей было собрать полные последовательности ПВП, а не понять, как разные ПВП в одной центромерной последовательности связаны друг с другом. Кроме того, не была указана точная целевая функция для вывода ПВП, из-за чего концепция ПВП сильно зависела от параметров, используемых для генерации набора мономеров [19].

Ранее представленный инструмент Alpha-CENTAURI [120] принимает на вход последовательности центромер человека и в качестве финального результата выдает набор мономеров и набор ПВП, на которые могут быть разложены заданные последовательности. Однако результат данного инструмента часто не соответствует полученным вручную ПВП [17]. Кроме того, кластеризация мономеров в Alpha-CENTAURI не идеальна, что снижает точность разложения строк на ПВП, поскольку инструмент создает ложные структуры ПВП.

В данной главе мы предлагаем два основных подхода для решения проблемы аннотации центромер. Инструмент CentromereArchitect (CA) был первым полностью автоматическим решением для аннотации центромер [18]. Он извлекает последовательности мономеров, выполняя кластеризацию блоков мономеров, полученных при запуске StringDecomposer на центромерной последовательности и одного консенсусного мономера. После этого CA решает задачу извлечения ПВП, формулируя ее как задачу сжатия данных. В CA основная идея решения задачи извлечения ПВП состоит в том, чтобы удалить избыточную информацию из центромерной последовательности путем сжатия идентичных частей. В свою очередь инструмент HORmon использует мономеры, полученные с помощью CentromereArchitect в качестве первого приближения решения задачи извлечения мономеров, однако далее HORmon пытается решить вместе задачу извлечения мономеров и задачу извлечения ПВП [19]. На данный момент трудно предоставить весомые аргументы, покрепленные данными, относительно того, какой подход, CA или HORmon, более оправдан с эволюционной точки зрения. Несмотря на это современное понимание того, как центромерные последовательности меняются с течением времени, резюмированное в постулате эволюции центромер [19] лучше отражается в процедуре HORmon.

3.2 Задача разложения строки

Задача разложения строки может быть сформулирована следующим образом. Дана строка R, представляющая центромерную последовательность, и набор строк блоков $Blocks$, каждая из которых соответствует одному мономеру. Конкатенация нескольких блоков из $Blocks$ определяется как *цепочка*. Тогда *оптимальная цепочка* для R определяется как цепочка, которая при глобальном выравнивании на R дает наибольшую идентичность по сравнению со всеми остальными возможными цепочками. Задача состоит в том, чтобы найти такую оптимальную цепочку для R.

3.2.1 Граф разложения строки

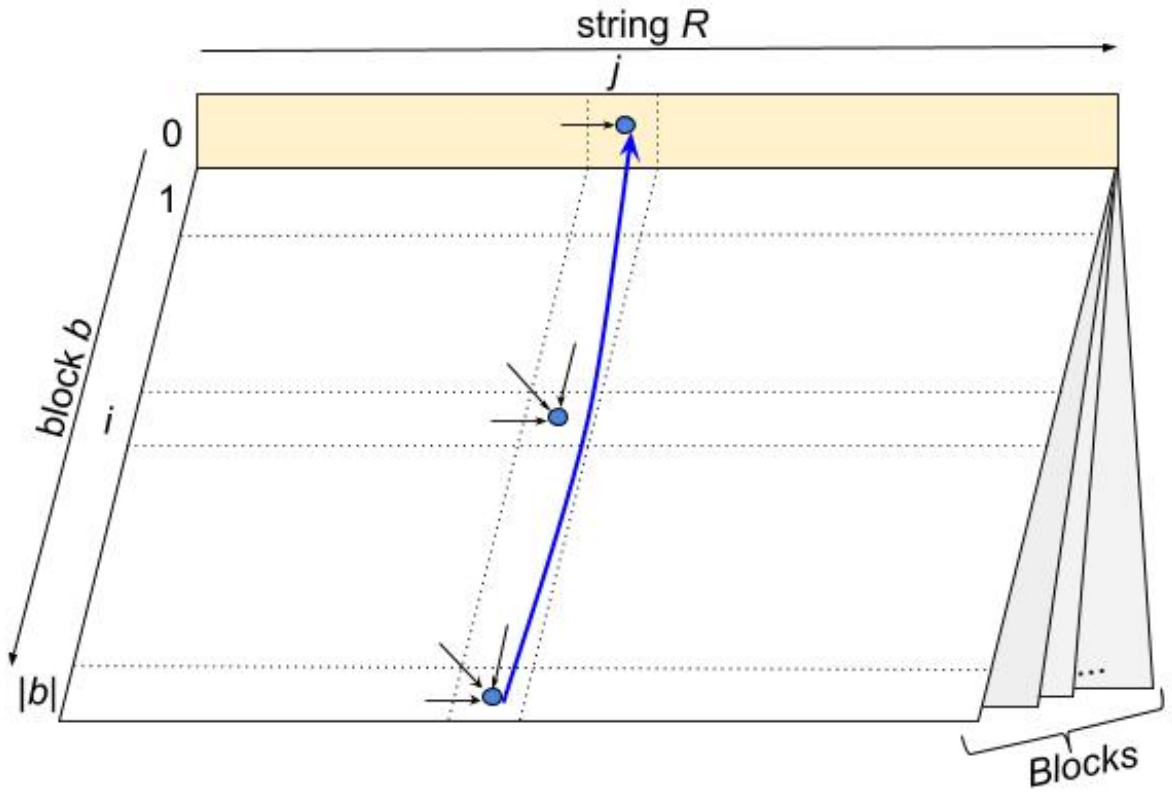

Рисунок 3.3 — Граф разложения строки можно представить в виде «книги», где каждая «страница» соответствует матрице выравнивания одного блока, а страницы связаны общей 0-й строкой. Ребра графа разложения строки представлены стрелками. В частности, ребро, соответствующее переключению между блоками, окрашено в синий цвет.

Рассмотрим стандартный *граф выравнивания* [124] двух строк: строки R и блока b из $Blocks$. Обычно он представляется в виде матрицы, где каждая ячейка с координатами (i,j) представляет собой вершину графа выравнивания и хранит вес оптимального выравнивания между префиксом R длины i и префиксом b длины j. Ориентированные ребра между вершиной (i,j) и вершинами $(i+1,j)$, $(i,j+1)$ и $(i+1,j+1)$ соответствуют вставкам, удалениям и совпадения/заменам соответственно.

Граф разложения строки для решения задачи декомпозиции строк можно представить в виде набора графов выравнивания для каждого блока b и строки R, «склеенных» по своим строкам номер 0 (рис. 3.3). За исключением вершин в 0-й строке, каждая вершина представлена в виде тройки (b,i,j), где b обозначает блок, i — позицию в блоке b, а j — позицию в R. Вершины в 0-й строке представляются парой $(0,j)$, где j представляет позицию в R. Кроме того, мы вводим новые ребра переключения блоков, ведущие из каждой ячейки последней строки $(b,|b|,j)$ в ячейку 0-й строки $(0,j)$ (синее ребро на рис. 3.3). Эти ребра переключения блоков имеют нулевой вес и обеспечивают переход от конца выравнивания одного блока к началу выравнивания следующего блока.

Более формальное определение графа разложения строки следующее. Каждая тройка (b,i,j), где b — блок, $1 \leq i \leq |b|$ и $0 \leq j \leq |R|$, определяет одну вершину. Аналогично каждая пара $(0,j)$, где $0 \leq j \leq |R|$ и 0 соответствует начальной позиции всех блоков, определяет одну вершину. Обычные ребра графа выравнивания между вершинами (b,i,j) (либо $(0,j)$) и $(b,i+1,j)$, $(b,i+1,j+1)$, $(b,i,j+1)$ представляют собой вставки, совпадения/замены и удаления соответственно. Ребра, соответствующие вставкам и удалениям, имеют вес $-\delta$, ребра-замены имеют вес $-\sigma$, а ребра-совпадения имеют вес 1. Ребра переключения блоков с нулевым весом соединяют вершину $(b,|b|,j)$ с вершиной $(0,j)$ для каждого блока b. Вершина (0,0) называется *истоком*, а вершина $(0,|R|)$ — *стоком* графа разложения строки. Поскольку все ориентированные циклы в графе разложения строки имеют отрицательные веса, самый тяжелый путь от истока к стоку в этом графе четко определен.

3.2.2 Алгоритм StringDecomposer

Для случая строки R и одного блока b алгоритм решения задачи разложения строки был предложен в [24] и был назван *алгоритм динамического программирования с зацикливанием* (wrap-around dynamic programming algorithm). Авторы другой работы [125] расширили этот алгоритм для произвольного количества блоков и реализовали решение для двух блоков в программном инструменте NTRFinder [126]. На данный момент NTRFinder не поддерживается. В этом разделе мы формально опишем основной алгоритм инструмента StringDecomposer, который решает задачу разложения строки для произвольного числа блоков.

Рассмотрим граф разложения строки для блоков $Blocks$ и строки R и схему подсчета весов за выравнивание со штрафом $-\delta$ за вставку, штрафом $-\sigma$ за замену и вознаграждением $+1$ за совпадение. Рассмотрим строку S и индекс i строки S (т. е. $0 \leq i \leq |S|$), тогда обозначение $S[:i]$ соответствует префиксу строки S длины i, а обозначение $S[i]$ соответствует i-му символу строки S. StringDecomposer применяет метод динамического программирования для вычисления $score(b,i,j)$ каждой вершины графа разложения строки, который определяется как вес оптимального глобального выравнивания для $R[:j]$ среди всех возможных цепочек блоков, заканчивающихся на $b[:i]$. За исключением вершин с входящими ребрами переключения блоков, $score(b,i,j)$ для каждой вершины определяется как максимум из трех значений: 1) $score(b,i-1,j)-\delta$, 2) $score(b,i,j-1)-\delta$, и 3) $score(b,i-1,j-1)-\sigma$, если $b[i]$ не равно $R[j]$, или $score(b,i-1,j-1)+1$, если $b[i]$ равно $R[j]$, как при решении обычной задачи выравнивания [127]. Значение $score(0,j)$ определяется как максимум среди всех $score(b,|b|,j)$. В самом начале инициализируются значения $score(b,0,j)=0$ для всех b и j и $score(b,i,0)=-i\cdot\delta$ для всех i.

После запуска алгоритма динамического программирования в переменной $score(0,|R|)$ хранится оценка самого тяжелого пути от истока к стоку. Самый тяжелый путь в этом графе определяет оптимальное выравнивание и, таким образом, решает задачу разложения строки. Каждый путь в графе разложения строки соответствует только одному возможному выравниванию цепочки блоков. И наоборот, каждое выравнивание цепочки блоков соответствует только одному пути в графе разложения строки. Для восстановления выравнивания,

рассмотрим последнее ребро переключения блоков из вершины (b,$|b|$,$|R|$) в сток (0,$|R|$), при переходе по которому был получен $score(0,|R|)$. Финальный блок оптимальной цепочки представлен блоком b с максимальным количеством очков. Для того чтобы восстановить другие блоки в этой цепочке, нужно осуществить проход в обратном направлении по графу разложения строки от стока (0,$|R|$) к истоку (0,0), каждый раз выбирая вершину и ребро с подходящим счетом.

Граф разложения строки не является ациклическим, но его можно преобразовать в эквивалентный ациклический граф, введя новые вершины (0,$j+1/2$) и заменив каждое ребро переключения блоков, заканчивающееся вершиной (0,j), парой ребер: ребром нулевой длины от начальной вершины данного ребра переключения блоков до новой вершины (0,$j+1/2$) и ребром нулевой длины от (0,$j+1/2$) до (0,$j+1$). Время работы и объем памяти алгоритма поиска самого тяжелого пути в ориентированном ациклическом графе пропорциональны количеству ребер в этом графе. Таким образом, время работы и объем памяти алгоритма StringDecomposer пропорциональны $O(|R| \cdot length(Blocks))$, где $length(Blocks)$ — общая длина всех блоков, а $|R|$ — длина строки.

3.2.3 Преобразование строк из нуклеотидного алфавита в алфавит блоков

Пусть заданы строка R и набор мономеров $Monomers$, StringDecomposer генерирует оптимальную цепочку мономеров $Monomers$ для R. Он возвращает начальную и конечную позиции $(s(m), e(m))$ и идентичность $Identity(m)$ для каждого выравнивания мономера m на R. Обозначим $R(m)$ как подстроку $R[s(m) : e(m)]$, соответствующую выравниванию мономера m на соответствующие позиции в R. При использовании StringDecomposer на реальных данных, когда алгоритм выравнивает набор мономеров либо на сборку, либо на длинное прочтение, StringDecomposer может выдавать выравнивание мономера m с низкой $Identity(m)$. Обычно это вызвано ошибками секвенирования в прочтениях или несовершенным набором входящих мономеров, который не включает все мономеры из заданной центромерной последовательности. Другая сложность заключается в том, что некоторые центромерные последовательности содержат подстроки, не образованные мономерами (например, вставки транспозонов

и LINE-элементы), и разложение таких подстрок не имеет смысла. Проблема ложной идентификации мономера становится первостепенной при обработке прочтений, содержащих высокий процент ошибок, например, прочтений ONT.

StringDecomposer использует следующую процедуру постобработки, чтобы выделить возможные ложные выравнивания. Рассмотрим выравнивание мономера m на центромерную цепочку R при разложении R на мономеры. Начальная и конечная позиции этого выравнивания ($s(m)$,$e(m)$), идентичность выравнивания $Identity(m)$. Рассмотрим выравнивание мономера m' на подстроку $R[s(m) : e(m)]$, которое имеет вторую лучшую идентичность выравнивания $Identity(m')$ на подстроку $R[s(m) : e(m)]$. StringDecomposer классифицирует каждое выравнивание мономера m как *надежное*/*ненадежное*, используя модель логистической регрессии с двумя параметрами: $Identity(m)$ и $IdentityDiff = Identity(m) - Identity(m')$. Мы сравнили эту модель с моделью с одним предиктором, основанной исключительно на $Identity(m)$, и показали ее превосходство.

Была обучена логистическая регрессия на 1000 ONT прочтениях, соответствующих подстрокам центромеры хромосомы X (называемых ONT_CENX), выровненных с помощью TandemMapper [118], и на 1000 ONT прочтениях, соответствующих другим областям сборки T2T хромосомы X v0.7, выровненных с помощью Minimap2 [22] (обозначается как ONT_NONCENX). Центромерные прочтения, содержащие LINE-элемент, были исключены из набора ONT_CENX. Как базовая модель, так и модель логистической регрессии с двумя предикторами были натренированы на обучающей выборке, построенной из 80% чтений ONT_CENX и ONT_NOTCENX, и протестированы на тестовой выборке, построенной из оставшихся 20%. Обе модели не дали ни одного ложноположительного ответа. Модель с двумя предикторами дала 198 ложноотрицательных выравниваний, в то время как модель с одним предиктором дала 241 ложноотрицательное выравнивание из 65 842 мономерных выравниваний тестовой выборки (см. рисунок 3.4).

В итоговом выводе StringDecomposer помечает ненадежные мономеры с помощью *символа пробела* (англ. gap symbol) «?». Таким образом при обработке прочтений StringDecomposer преобразует строку R в строку в *расширенном алфавите мономеров*, который включает все мономеры и символ пробела.

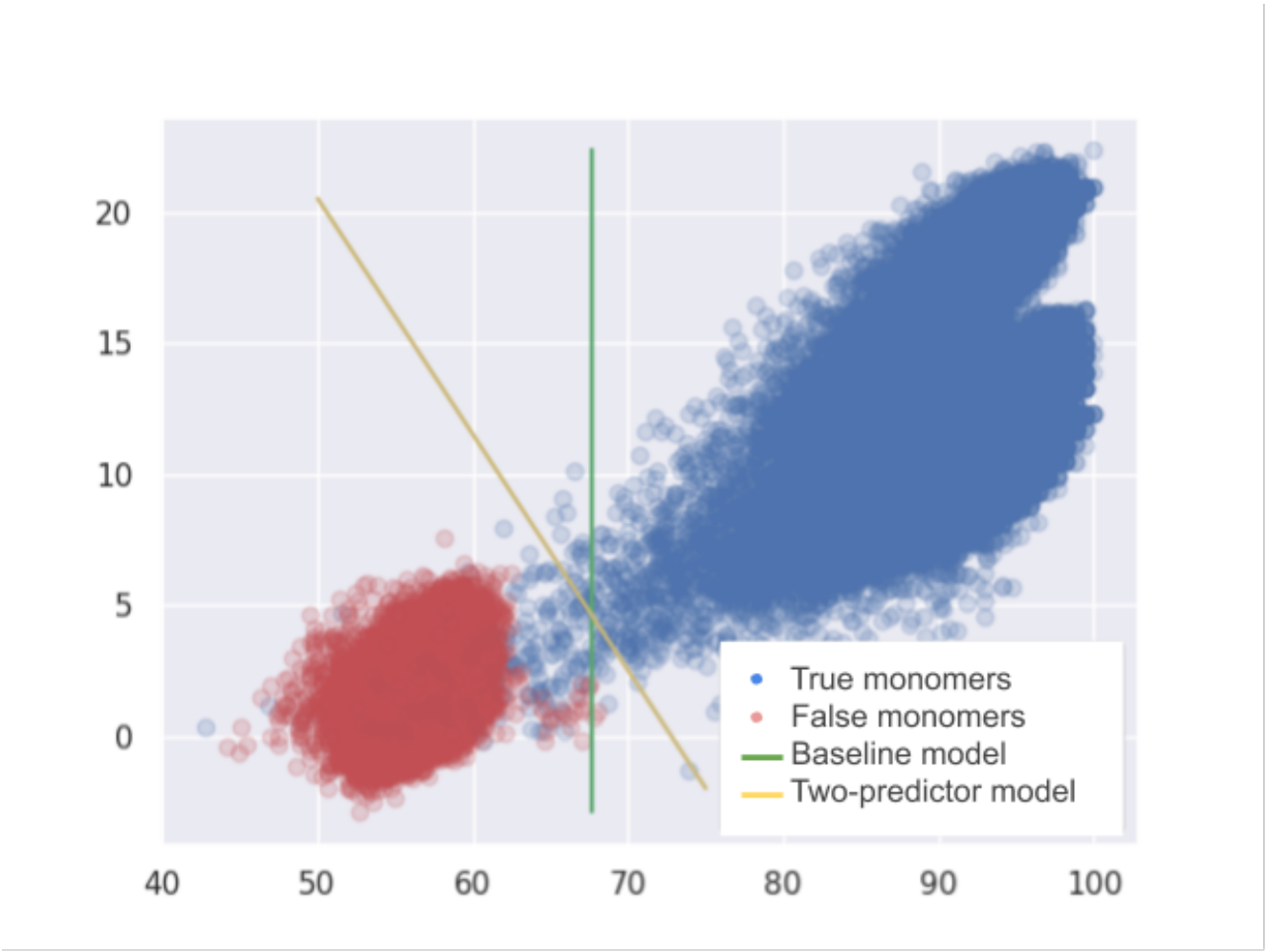

Рисунок 3.4 — Распределение Identity (ось OX) и IdentityDiff (ось OY) для выравнивания мономеров в центромерных областях. Каждая точка на графике представляет либо истинное выравнивание мономера (синие точки), либо ложное выравнивание мономера (красные точки). Зеленая линия разделяет истинные и ложные мономеры в соответствии с базовым классификатором на основе значения $Identity$ каждого выравнивания, а желтая линия разделяет истинные и ложные выравнивания мономеров в соответствии с логистической регрессией, обученной на значениях $Identity$ и $IdentityDiff$.

3.3 Результаты алгоритма StringDecomposer

3.3.1 Описание данных

В этом разделе мы сравниваем инструмент StringDecomposer (далее SD) с единственным существующим решением задачи разложения строки, AlphaCENTAURI (далее AC), на ONT прочтениях центромеры хромосомы X (да-

лее cenX). Для сравнительного анализа мы использовали сборку хромосомы X v0.7, первую полную сборку хромосомы, сгенерированную консорциумом T2T [116]. Набор мономеров был получен из консенсусной последовательности ПВП DXZ1* [12]. DXZ1* был разложен на двенадцать мономеров, обозначенных заглавными латинскими буквами ABCDEFGHIJKL, с помощью AlphaCENTAURI [120] в полуручном режиме.

В нашем исследовании было проанализировано 11 069 717 прочтений, полученных с помощью технологии Oxford Nanopore, сгенерированных в рамках инициативы консорциума T2T. Прочтения имеют общую длину 155 млрд нуклеотидов с покрытием 50x, и N50 равным 70 тыс. нуклеотидов. Как показано в [12] 2680 из этих прочтений (суммарной длиной 132,9 млн нуклеотидов) были сгенерированы из последовательности cenX. Из 2680 прочтений cenX 1442 были успешно выровнены на референс с помощью TandemMapper [118]. Полная суммарная длина 1442 успешно выровненных прочтений составляет 121 млн нуклеотидов, а суммарная длина выровненных фрагментов составляет 76 млн нуклеотидов. Невыровненные префиксы и суффиксы каждого прочтения были обрезаны, и для дальнейшего анализа использовалась только выровненная подстрока каждого прочтения, обозначаемая как *Read*. Набор всех таких подстрок будет далее обозначен как *Reads*, а строка в cenX, соответствующая выравниванию данного *Read* на cenX, будет обозначена как $origin(Read)$.

Все инструменты, используемые для анализа этих наборов данных, доступны по адресу https://github.com/TanyaDvorkina/sdpaper.

3.3.2 Тестирование StringDecomposer на прочтениях с высоким количеством ошибок

Рассмотрим центромерную последовательность (или последовательность прочтения) S и набор мономеров $Monomers$. Мы определим моноцентромеру (монопрочтение) $mono(S)$ как строку в мономерном алфавите, представляющую разложение S. Рассмотрим *Read* из *Reads* и его разложение $mono(Read)$ на мономеры. Для того чтобы убедиться в правильности его разложения, мы можем сравнить $mono(Read)$ с $mono(origin(Read))$, который представляет собой мономерное разложение соответствующей референсной подстроки. Такое

сравнение $mono(Read)$ с $mono(origin(Read))$ используется для тестирования подходов SD и AC.

Мы применили инструменты SD и AC к последовательности cenX длиной 3.1 млн нуклеотидов, чтобы получить моноцентромерную последовательность, обозначенную как $mono(cenX)$. И SD, и AC генерировали одну и ту же моноцентромерную последовательность со средним процентом идентичности мономерных выравниваний в последовательности cenX 98.8%. Результирующая последовательность $mono(cenX)$ состоит из 18 103 мономеров и 36 подряд идущих символов пробелов, что указывает на присутствие в регионе LINE-элемента. На рисунке 3.5 показано распределение идентичности в процентах для выравниваний двенадцати мономеров cenX и символа пробел.

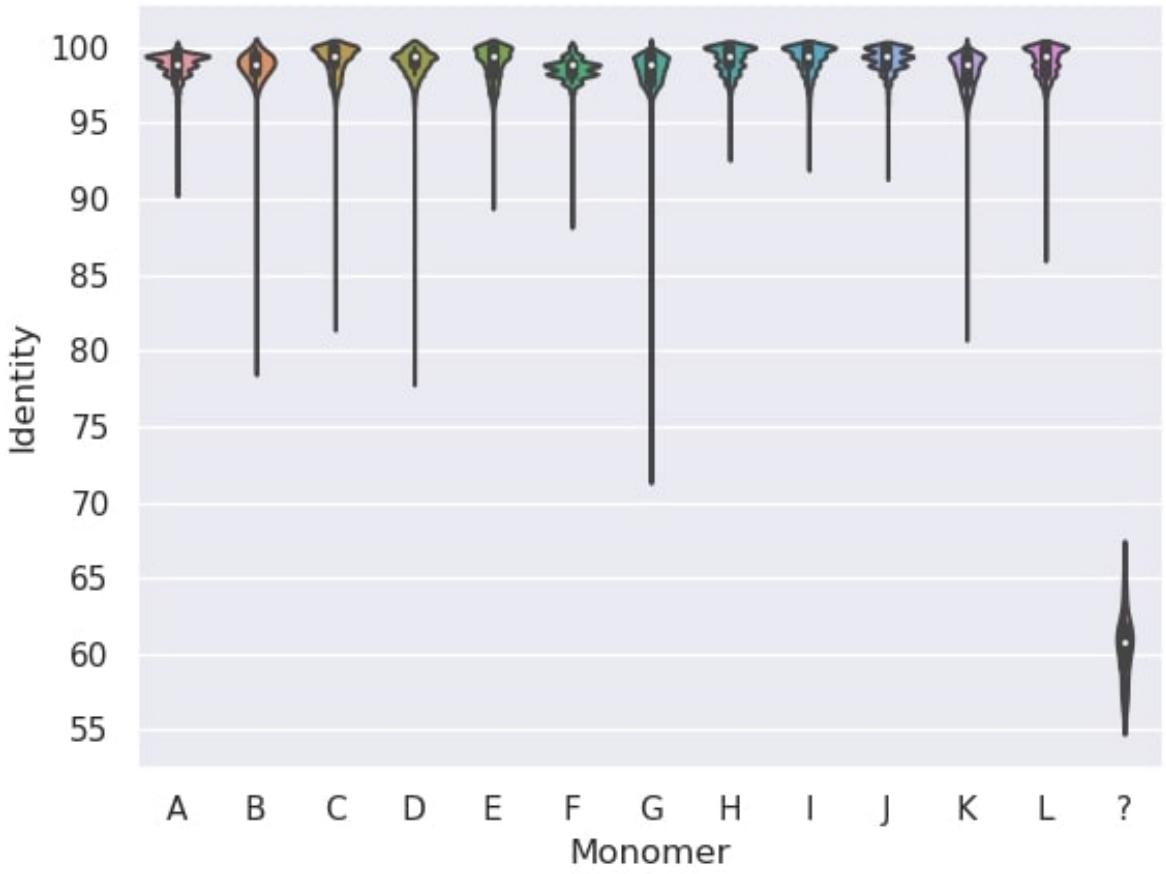

Рисунок 3.5 — Распределение идентичности в процентах для 12 мономеров cenX и символа пробела «?». Каждый график соответствует одному из 12 мономеров и показывает диапазон и частоту значений идентичности в процентах выравниваний данного мономерв в cenX. Центромера хромосомы X содержит LINE-элемент на позициях $CHM13 : X : 2773652\text{-}2779726$, который представлен 36 последовательными пробелами («?»).

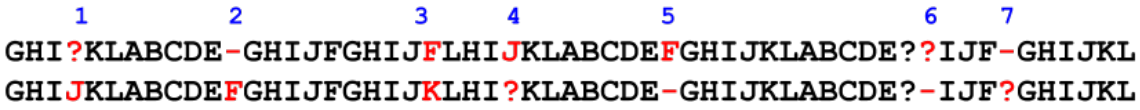

Рисунок 3.6 — Пример выравнивания монопрочтения к моноцетромере. Первая строка представляет $mono(origin(Read))$, вторая строка представляет $mono(Read)$. Совпадающие позиции показаны черным цветом, а позиции с ошибками - красным. Показаны следующие типы ошибок: замена мономера на пробел (1), вставка мономера (2), замена мономера на мономер (3), замена пробела на мономер (4), удаление мономера (5), удаление пробела (6), вставка пробела (7).

Инструменты SD и AC были запущены для преобразования каждого прочтения из $Reads$ в монопрочтение: $mono_{SD}(Read)$ и $mono_{AC}(Read)$ соответственно. Затем мы сопоставили каждое из этих монопрочтений с $mono(origin(Read))$, используя обычное редакционное расстояние для оценки похожести, со штрафами 1 за вставки, удаления и замены, и 0 за совпадение. Каждый столбец выравнивания состоял из пары символов, где первый символ соответствовал позиции в $mono(origin(Read))$, а второй символ соответствовал позиции в $mono(Read)$. Символы включали двенадцать мономеров, символ «?» для обозначения ненадежных мономеров и символ «-» для обозначения вставок и делеций в выравнивании. Пример выравнивания $mono(Read)$ и $mono(origin(Read))$ со всеми возможными результатами для определенного столбца показан на рисунке 3.6.

Общая статистика выравнивания $Reads$ для обоих инструментов представлена в таблице 7. Большинство ошибок для обоих инструментов связаны с заменами мономеров, которые обычно возникают из-за ошибок прочтения ONT. SD привел к трехкратному снижению ошибок по сравнению с AC (1183 против 3478), продемонстрировав лучшую обработку поврежденных участков при работе с прочтениями ONT.

В обоих случаях общий процент ошибок очень низок: 0.86% для AC и 0.32% для SD. Хотя разница между этими двумя подходами может показаться незначительной, для некоторых приложений, таких как сборка центромер, длина безошибочных k-мономеров в монопрочтениях может иметь решающее значение для результата. Например, если предположить, что ошибки распределены равномерно (что является нереалистичным предположением), SD предоставляет надежную информацию о 312-мономерах, в то время как AC предо-

	SD	AC	SD	AC	SD	AC	SD	AC
Read → cenX ↓	monomer		?		-		Total	
monomer	119	139	1183	3478	117	154	1419	3771
?	0	0	-	-	19	19	19	19
-	0	6	18	20	-	-	18	26
Total	119	145	1201	3498	136	173	1457 (0.32%)	3816 (0.86%)

Таблица 7 — Статистика ошибок выравниваний монопрочтений на моноцентромеру для инструментов AC (черный) и SD (синий) для 12 мономеров. Таблица показывает ошибки для 12 известных мономеров cenX. Используемые в таблице символы включают «мономер», который представляет собой один из двенадцати мономеров cenX, «?», который представляет собой символ пробела, и «-», который указывает на событие вставки или удаления в выравнивании $mono(Read)$ относительно $mono(origin(Read))$. Каждая ячейка (i, j) в таблице показывает количество раз, которое символ типа i в $mono(origin(Read))$ выравнивался с символом типа j в $mono(Read)$. Количество совпадений для разложения на 12 мономеров составляет 445 169 для SD и 442 783 для AC.

ставляет надежную информацию только о 116-мономерах в моночтениях, так как $100/0.32 = 312$ и $100/0.86 = 116$.

3.3.3 Определение первых гибридных мономеров

В [13] авторы проанализировали набор мономеров, сгенерированный референсными моделями ([25]), и выдвинули теорию о том, как мономеры могут возникать в результате конкатенации двух или более мономеров из исходного набора $Monomers$. Мономеры, получающиеся таким способом, были названы *гибридными*. Изучая замены одного мономера на другой, выявленные подходами SD и AC, мы обнаружили экземпляры *гибридных* мономеров в фактической сборке cenX.

В частности, из 119 (139) замен одного мономера на другой, выявленных с помощью SD (AC), 104 (115) являются заменами мономера K на мономер F. Эти несоответствия произошли в нестандартном 16-мономерном ПВП (ABCDEFGH**I**J**F**GHIJKL), где второе вхождение F часто заменяется на K. В нестандартном 11-мономерном ПВП (ABCDEFGH**I**J**K**), мономер K обычно

заменяется на L по аналогичной схеме. Эти выводы проиллюстрированы на тепловой карте, показанной на рисунке 3.7.

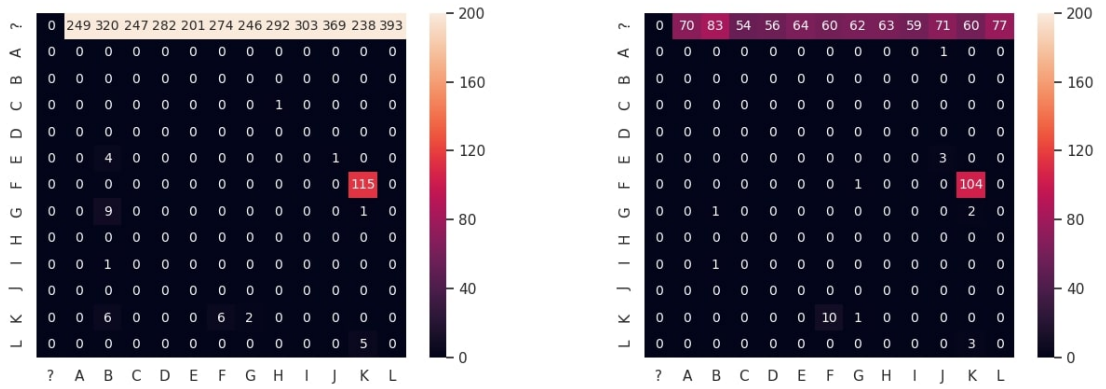

Рисунок 3.7 — Матрицы замен между мономерами для подходов AC (Слева) и SD (Справа). Каждая ячейка (X,Y) указывает количество случаев, когда символ X, который может быть мономером или символом пробела «?», в моноцентромере был заменен символом Y в монопрочтении.

Были рассмотрены все шесть вхождений нестандартного ПВП ABCDEFGHIJ**F**GHIJKL и все шесть вхождений нестандартного ПВП ABCDEFGHIJ**K**GHIJKL в моноцентромеру, далее извлечены нуклеотидные последовательности мономера **F** из каждого вхождения ABCDEFGHIJ**F**GHIJKL и нуклеотидные последовательности **K** из каждого вхождения ABCDEFGHIJ**K**GHIJKL (далее эти мономеры обозначаются как **FK**). По полученным последовательностям была построена матрица выравниваний (рис. 3.8 слева), где каждая ячейка показывает попарную идентичность двух нуклеотидных строк **FK**. StringDecomposer классифицировал выравнивания 1—4 и 11—12 как F, а выравнивания мономера 5—10 как K. Действительно, вхождения 1—4 практически совпадают с мономером F (идентичность 98—100%). Вхождения мономеров 5—12 очень похожи друг на друга, но при этом имеют низкие показатели идентичности как с F, так и с K.

Мы проанализировали попарные выравнивания между вхождениями 5—12 **FK** и мономерами F и K. Наш анализ показал, что 5—12 представляют собой вхождения *гибридного* мономера, образованного первой половиной мономера K и второй половиной мономера F. Если создать синтетический мономер **K**+**F** из первой части мономера K (заканчивается на 98 позиции **K**) и второй части мономера F (начинается с 101 позиции **F**), и построить его выравнивания

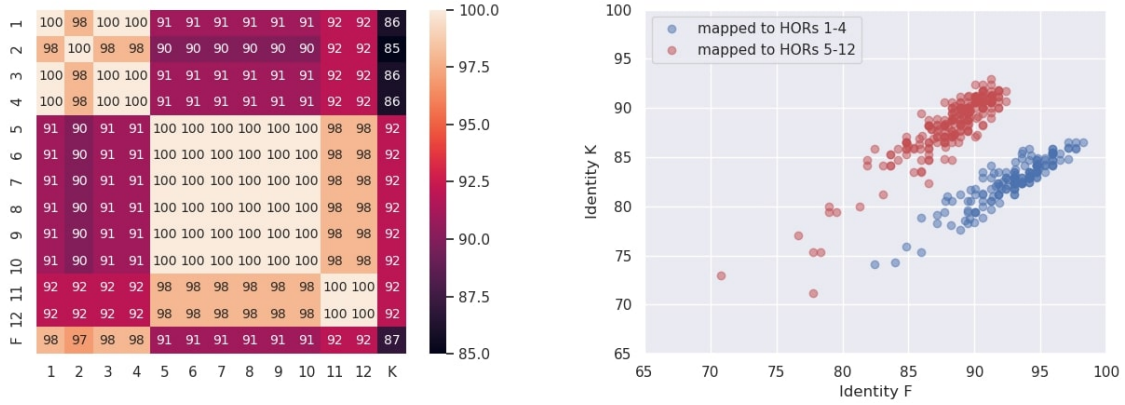

Рисунок 3.8 — Анализ нестандартных ПВП ABCDEFGHIJFGHIJKL. На левой панели показана тепловая карта, которая отображает идентичность между вхождениями **FK** в двенадцати нестандартных ПВП ABCDEFGHIJ**F**GHIJKL и ABCDEFGHIJ**K**GHIJKL в cenX, а также мономеров F и K. На правой панели показана идентичность 377 выравниваний мономера F (ось OX) и мономера K (ось OY) относительно соответствующей подстроки прочтения для всех 377 вхождений ABCDEFGHIJ**F**GHIJKL или ABCDEFGHIJ**K**GHIJKL в *Reads*. Синие (красные) точки обозначают вхождения мономера F или K, выровненные с **FK** из ПВП 1–4 (ПВП 5–12).

на вхождения 5—12 **FK**, выравнивания будут иметь идентичность около 97% (по сравнению с 91% с F и 92% с K).

Для того, что показать, что такой *гибридный* мономер поддерживается прочтениями, мы извлекли все монопрочтения, содержащие нестандартные ПВП ABCDEFGHIJ**F**GHIJKL (всего 272 монопрочтений) и ABCDEFGHIJ**K**GHIJKL (всего 105 монопрочтений). К первому кластеру было отнесено 154 вхождений мономера **FK**, т. е. последовательности соответствующие последовательностям 1—4 на рисунке 3.8 слева, а ко второму кластеру — 223, т. е. последовательности соответствующие последовательностям 5—12 на рисунке 3.8 слева. Мы сопоставили нуклеотидные последовательности **FK** из прочтений с мономерами K и F. Правая панель рисунка 3.8 ясно демонстрирует, что последовательности выравниваний в прочтениях можно разделить на два кластера на основе их идентичности к F и K, и эта классификация совпадает с группировкой на левой панели рисунка 3.8.

В таблице 8 показаны результаты SD и AC, пересчитанные на наборе из 12 исходных мономеров с добавлением мономера **K**+**F**. Моноцентромера была пересчитана с использованием 13 мономеров, и вхождения в нее 5—12 были классифицированы как новый мономер **K**+**F**, а не как мономеры K или F. Количе-

	SD	AC	SD	AC	SD	AC	SD	AC
Read → cenX ↓	monomer		?		-		Total	
monomer	11	38	1256	3481	119	154	1386	3673
?	0	0	-	-	19	19	19	19
-	1	6	17	21	-	-	18	27
Total	12	44	1273	3502	138	173	1423 (0.31%)	3719 (0.83%)

Таблица 8 — Статистика ошибок выравниваний монопрочтений на моноцентромеру для инструментов AC (черный) и SD (синий) для 13 мономеров. Таблица показывает ошибки разложения на 12 известных мономеров cenX и дополнительный новый мономер K+F. Используемые в таблице символы включают «мономер», который представляет собой один из двенадцати мономеров cenX или новый мономер K+F, «?», который представляет собой символ пробела, и «-», который указывает на событие вставки или удаления в выравнивании $mono(Read)$ относительно $mono(origin(Read))$. Каждая ячейка (i, j) в таблице показывает количество раз, которое символ типа i в $mono(origin(Read))$ выравнивался с символом типа j в $mono(Read)$. Количество совпадений для разложения на 13 мономеров составляет 445 206 для SD и 442 887 для AC.

ство несоответствий между мономерами уменьшилось со 119 (139) до 11 (47) для SD (AC), в то время как количество других ошибок не изменилось.

Эти результаты показали потенциальный механизм образования новых мономеров. По-видимому, при репликации ДНК вторая половина мономера **K** и некоторые расположенные дальше мономеры каким-то образом пропускались до тех пор, пока не приклеивалась вторая половина мономера **F**, продолжая с нового места регулярную репликацию. Стоит отметить, что для получения последовательности мономера **K+F** мономеры **K** и **F** были склеены по консервативной части одинаковой в обеих последовательностях, а длина полученного мономера составляет 171 нуклеотид.

3.4 Аннотация центромер

В данной секции описываются схемы работы двух инструментов для полной аннотации центромерных последовательностей, CentromereArchitect и HORmon. HORmon стал преемником CentromereArchitect.

3.4.1 Схема работы инструмента CentromereArchitect

CentromereArchitect (CA) — первый инструмент для автоматического аннотирования последовательностей центромер человека. CA принимает в качестве входных данных набор центромерных последовательностей *Centromeres* и последовательность *InitialMonomer*, консенсусный мономер для всех последовательностей альфа-сателлитов в геноме человека. Конвейер CA состоит из двух основных этапов: решения задачи извлечения мономеров и решения задачи извлечения ПВП. На этапе решения задачи извлечения мономеров CA строит набор мономеров, с помощью которых можно решить задачу разложения строки для всего набора Centromeres. Каждый мономер должен часто встречаться в разложении и иметь выравнивания высокой идентичности. На этапе решения задачи извлечения ПВП CA пытается идентифицировать ПВП, обрабатывая mono(Centromeres). Кроме того, CA пытается идентифицировать более крупные повторяющиеся строки, называемые далее *сверхПВП*. Наконец, CA выводит набор мономеров, набор ПВП и набор сверхПВП, а также разложения центромерных последовательностей на итоговые мономеры, ПВП и сверхПВП. На рисунке 3.9 показан полный цикл работы CA, далее следует краткое описание каждого шага:

- **Извлечение мономеров.** На данном шаге CA итеративно расширяет набор мономеров, начиная с набора, состоящего из одного *InitialMonomer*. CA использует StringDecomposer для получения разложения mono(Centromeres) с текущим набором мономеров. Выравнивания мономеров с низкой идентичностью в разложении кластеризуются с помощью односвязной кластеризации, и представитель самого большого кластера добавляется в набор мономеров. Этот процесс завершается после того, как 95% длины *Centromeres* покрыто выравниваниями мономерами с высокой идентичностью.
- **Идентификация гибридных мономеров.** Для каждого редкого мономера M CA пытается идентифицировать пару частых мономеров X,Y, из которых посредством склеивания префикса X и суффикса Y мог получиться мономер M.
- **Извлечение ПВП.** CA рассматривает задачу задачи извлечения ПВП как задачу сжатия данных. Основная идея состоит в том, что если цен-

тромера представляет собой повторяющуюся последовательность, то начальный представитель ПВП (или набор начальных ПВП) представляет собой короткую подпоследовательность, характеризующую центромерную последовательность. Кроме того, CA строит повторы еще более высокого уровня, называемые сверхПВП.

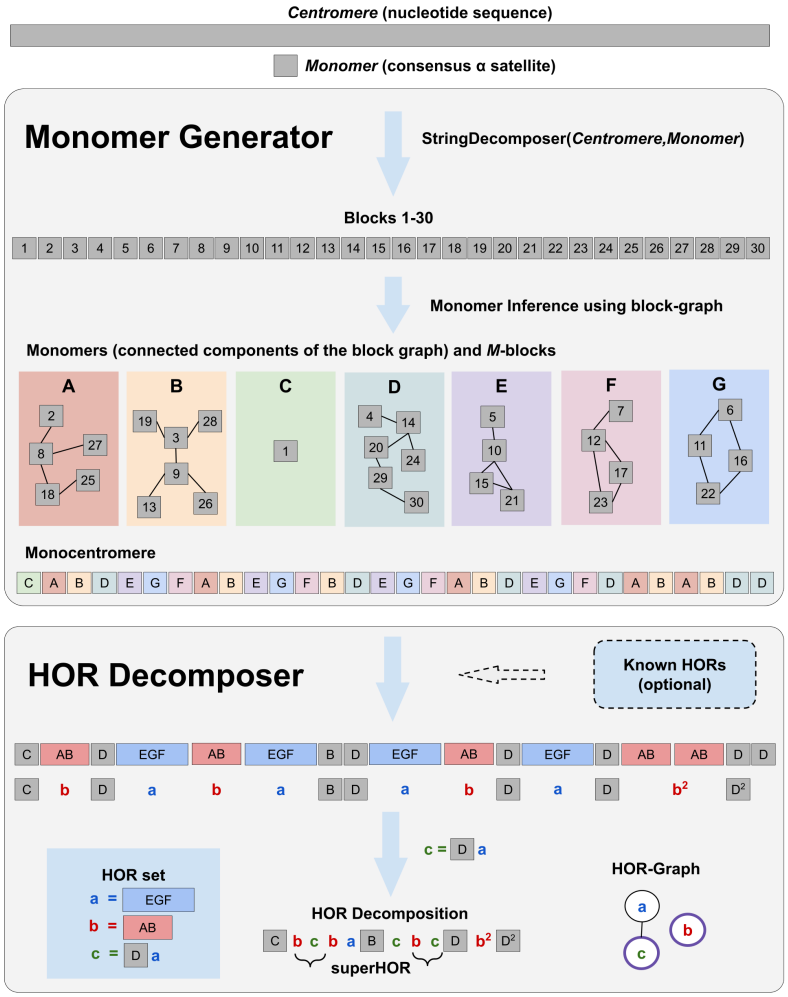

Рисунок 3.9 — Схема работы CentromereArchitect. В качестве входных данных CentromereArchitect принимает нуклеотидную последовательность центромеры Centromere и консенсусную альфа-сателлитную последовательность *InitialMonomer*. Алгоритм работы CentromereArchitect состоит из двух основных частей MonomerGenerator и HORDecomposer. MonomerGenerator строит граф блоков выравниваний, объединяя в одну компоненту похожие блоки выравнивания. Процесс повторяется итеративно, пока большая часть центромеры не будет покрыта выравниваниями мономеров с высоким процентом идентичности. Финальное разделение превращает центромеру в моноцентромеру. Далее HORDecomposer выводит набор ПВП и разделяет моноцентромеру на выделенные ПВП. В качестве дополнительных входных данных HORDecomposer может включать известные ПВП. Наконец, HORDecomposer извлекает сверхПВП из разложения центромеры на ПВП.

Анализ результатов CentromereArchitect на центромерных последовательностях человека продемонстрировал важность порядка мономеров в ПВП для решения задачи извлечения мономеров и позволил сформулировать более строгое определение канонического ПВП. Выполняя процедуру кластеризации мономеров, CA рассматривает каждое мономерное выравнивание отдельно от соседних выравниваний. Кроме того, CentromereArchitect определяет канонический ПВП, как наиболее часто повторяющуюся подстроку моноцентромеры, однако такое предположение оказывается верным не всегда. Для решения этих проблем и создания подхода, отражающего принципы постулата эволюции центромер, был разработан алгоритм HORmon.

3.4.2 Схема работы инструмента HORmon

Подобно CentromereArchitect, входными параметрами HORmon являются центромерные последовательности *Centromeres* и *InitialMonomer*. Конвейер HORmon начинается с запуска первого шага CentromereArchtect и рассматривает мономеры, созданные CA *CAMonomers*, как исходные мономеры для дальнейшей обработки. Далее инструмент HORmon выполняет следующие шаги (рис. 3.10):

- **Построение графа мономеров.** HORmon строит граф мономеров. Граф мономеров представляет собой ориентированный граф, каждая вершина которого соответствует одному мономеру из *CAMonomers*, а вес ребра от вершины мономера A до вершины мономера B показывает, сколько раз B следует за A в моноцентромере, образованной разложением на мономеры *CAMonomers*. Важно отметить, что граф мономеров представляет собой простой граф де Брюйна с $k = 2$.
- **Улучшение набора мономеров на основе информации об их положении.** HORmon выполняет итеративные операции слияния/разделения мономеров на основе их взаимного расположения и статистики частоты встречаемости, взятой из графа мономеров. Например, если мономеры X и X' всегда встречаются между одними и теми же двумя мономерами A и B (AXB и AX'B при разложении) и имеют высокое сходство, то, вероятно, они представляют собой один и тот же мономер,

и HORmon объединяет мономеры X и X' в один. С другой стороны, если мономер X обычно имеет два разных контекста, то есть иногда он встречается между мономерами A и B, а иногда между мономерами A' и B', то, вероятно, X следует разделить на два разных мономера. Набор мономеров, полученный на данном этапе, будет обозначаться $Monomers$.

- **Идентификация гибридных мономеров.** HORmon рассматривает все мономеры из конечного набора мономеров после операций разделения/слияния как потенциальные гибридные мономеры. Для каждого мономера A HORmon находит пару различных мономеров B и C, конкатенация префикса и суффикса которых наиболее похожа на A среди всех возможных конкатенаций префиксов и суффиксов других мономеров, и рассматривает A в качестве *кандидата* в гибридные мономеры, если он похож на конкатенацию B и C, но отличается от обоих B и C. Далее HORmon рассматривает как *гибридные мономеры* только гибридных кандидатов, которые могут быть представлены как конкатенация мономеров, не являющихся кандидатами в гибриды.
- **Дегибридизация.** Все обнаруженные гибридные мономеры заменяются в мономерном графе соответствующими ребрами (см. рисунок 3.10 дегибридизации мономера D/E) и удаляются из $Monomers$.
- **Извлечение ПВП.** HORmon строит упрощенный граф мономеров, состоящий только из ребер, соединяющих только мономеры канонического ПВП. Используя упрощенное представление графа мономеров, HORmon извлекает набор канонических ПВП.
- **Разложение ПВП.** Далее HORmon раскладывает моноцентромеру на канонические, частичные и вырожденные ПВП (см. рис. 3.10).

3.5 Аннотация последовательностей центромер генома человека

В этом разделе представлен анализ результатов HORmon v1.0 на центромерных последовательностях из последней на данный момент референсной сборки генома человека [9]. Точные координаты каждой центромерной последовательности приведены в [19] в дополнительной таблице S3.1. Идентифици-

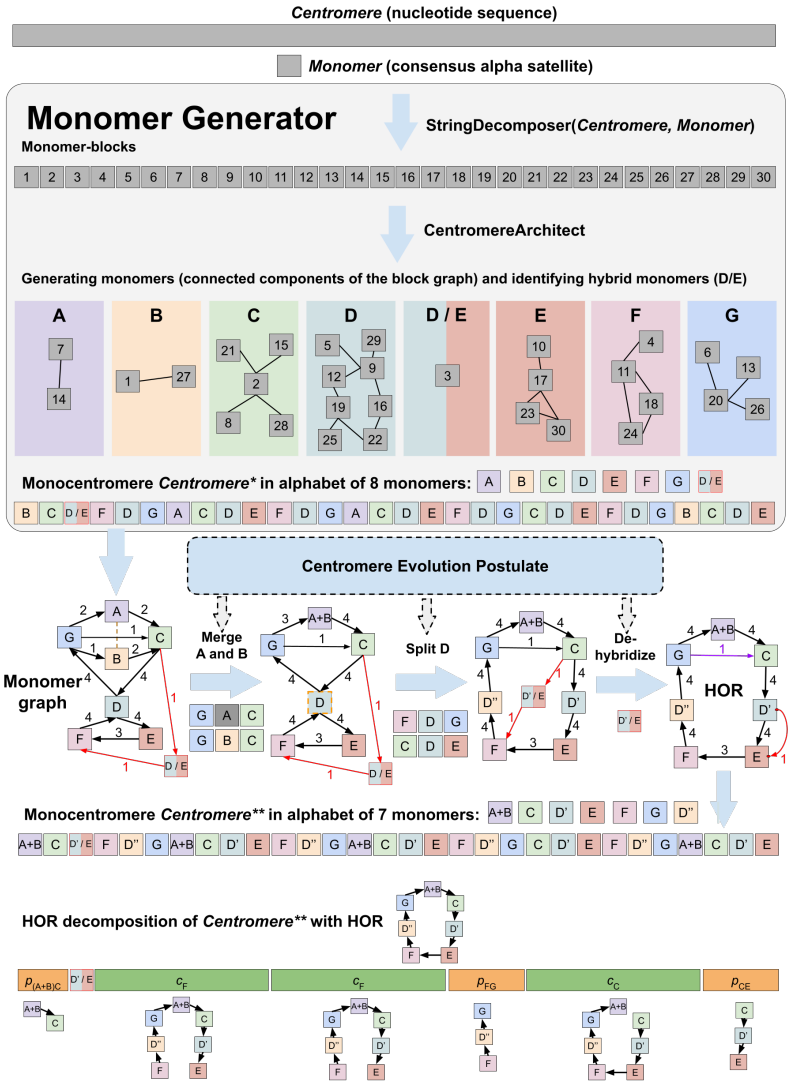

Рисунок 3.10 — Схема работы инструмента HORmon. HORmon разбивает центромерную последовательность Centromere, на блоки с помощью StringDecomposer [17]. Затем он использует CentromereArchitect [18] для идентификации похожих блоков, объединения их в мономеры, и преобразовывает Centromere в моноцентромеру, обозначаемую Centromere*. Полученный из разложения Centromere* граф мономеров используется для выполнения преобразований слияния/разделения и дегибридизации в соответствии с постулатом эволюции центромер. Разложение центромерной последовательности на финальный набор мономеров называется Centromere**. Черный цикл на мономерном графе представляет собой ПВП и используется для создания ПВП-разложения Centromere** на канонические, частичные и вырожденные ПВП.

рованные мономеры, ПВП и разложения центромерных последовательностей на мономеры и ПВП можно найти в дополнительных материалах [19].

Идентификация мономеров и канонических ПВП. На начальном этапе HORmon запускает модуль извлечения мономеров CentromereArchitect на *всех* центромерных последовательностях, чтобы идентифицировать моно-

меры, представленные на разных центромерах. Как было показано ранее в [13; 18; 115], большинство центромер представлено набором мономеров, отличным от набора мономеров любой другой центромеры, но пары центромер (хромосома 13 и хромосома 21) и (хромосома 14 и хромосома 22), а также тройка центромер (хромосома 1, хромосома 5 и хромосома 19), имеют одни и те же мономеры. Результаты модуля извлечения мономеров CentromereArchitect согласуются с этим распределением мономеров по центромерам.

Набор негибридных мономеров для каждой центромеры, полученных после улучшения набора мономеров на основе информации об их положении, совпадает с набором мономеров, полученным вручную в [20] (дополнительная таблица S4 в [19]) . Кроме того, HORmon обнаружил три *частых* гибридных мономера в центромерах 5 и 8: P5, R1/5/19 и L8. На рисунках 3.11 и 3.12 показаны мономерные графы для центромер хромосом 5 и 8 до и после процедуры дегибридизации.

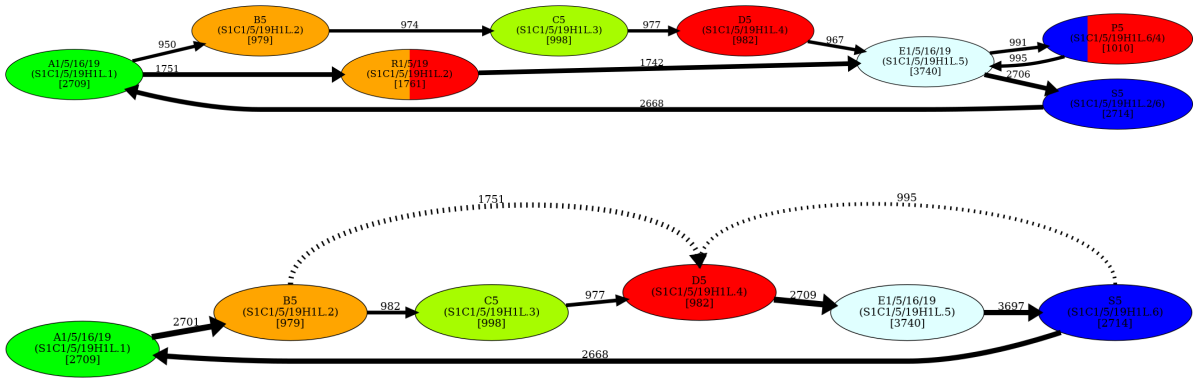

Рисунок 3.11 — Замена вершин гибридных мономеров в графе мономеров на соответствующие ребра для cen5. На рисунке показаны мономерный граф для cen5 (Сверху) и мономерный граф для cen5 после дегибридизации P5 и R1/5/19 (Снизу).

Структуры графов мономеров для центромер 3, 11, 14, 16, 17, 19, 20, 21, 22, X и Y, сгенерированные HORmon, изначально представляют собой циклы, соответствующие каноническим ПВП (рис. 3.13, левая панель). Граф мономеров для центромеры 17 содержит цикл ПВП D17Z1, который представляет собой основной ПВП, и цикл ПВП D17Z1-B для так называемой сестринской ПВП [128]. Для всех остальных центромер процедура вывода ПВП с построением упрощенного графа мономеров помогает выявить канонические ПВП.

Выделение канонического ПВП в центромере хромосомы 9 представляет собой сложный случай для HORmon и постулата эволюции центромер, поскольку повторяющаяся последовательность cen9, по-видимому, состоит из двух раз-

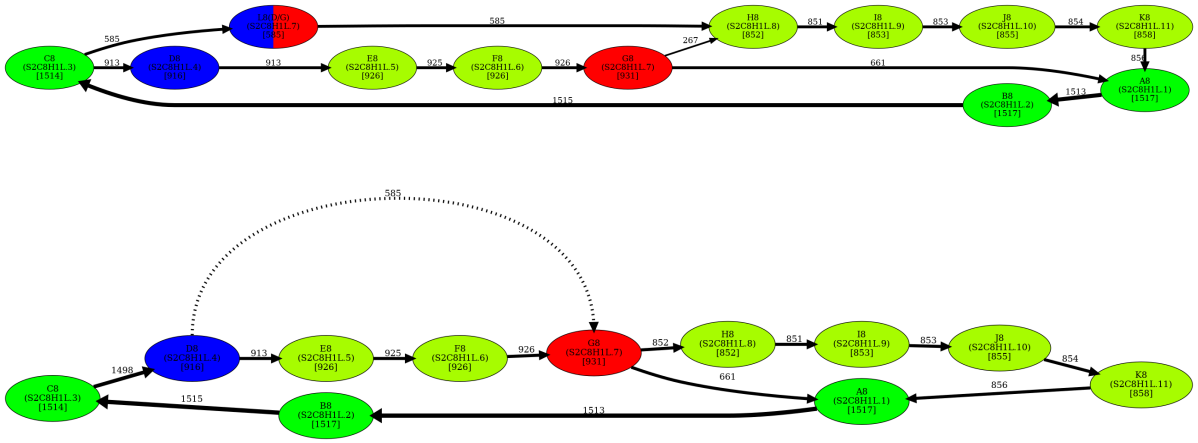

Рисунок 3.12 — Замена вершин гибридных мономеров в графе мономеров на соответствующие ребра для cen8. На рисунке показаны мономерный граф для cen8 (Сверху) и мономерный граф для cen8 после дегибридизации мономера L8 (Снизу).

ных ПВП (представленных синим и желтым циклами на правой панели рисунка 3.13). При этом каждый из двух ПВП содержит мономер, которого нет в другом ПВП и который не является гибридом двух других мономеров согласно текущим настройкам параметров. Единственный мономер, который есть только в желтом цикле ПВП (правая панель рисунка 3.13), отличается от самого близкого мономера Z4/9 на 12 нуклеотидов, что достаточно много. Из-за ограниченности доступных данных (пока что полностью собран только один геном человека) трудно восстановить реальный эволюционный сценарий и идентифицировать один канонический ПВП в данном случае.

Генерация разложения на ПВП. После извлечения мономеров и ПВП, HORmon провел разложения каждой центромеры на мономеры, а каждую полученную моноцентромеру разложил на канонические, частичные и врожденные ПВП. В то время как канонический ПВП обычно представляется в виде строки с фиксированным «первым» мономером, HORmon рассматривает его как цикл в графе мономеров без определенного начала. Рассмотрим канонический ПВП H, представленный в виде последовательности мономеров $M_1, \ldots, M_n$, каждый циклический сдвиг этой последовательности $M_i, \ldots, M_n, M_1, \ldots, M_{i-1}$ схлопывается в моноцентромере в символ c_i. Каждый частичный ПВП $M_i, \ldots, M_j$, представляющий подстроку строки $M_1, \ldots, M_n$, обозначается как p_{i-j}. Каждый вырожденный ПВП из одного мономера представлен заглавной буквой данного мономера. Для обозначения m последовательных вхождений одного и того же ПВП с одним и тем же сдвигом используется следующее написание: ${c_i}^m$ (${p_{i-j}}^m$).

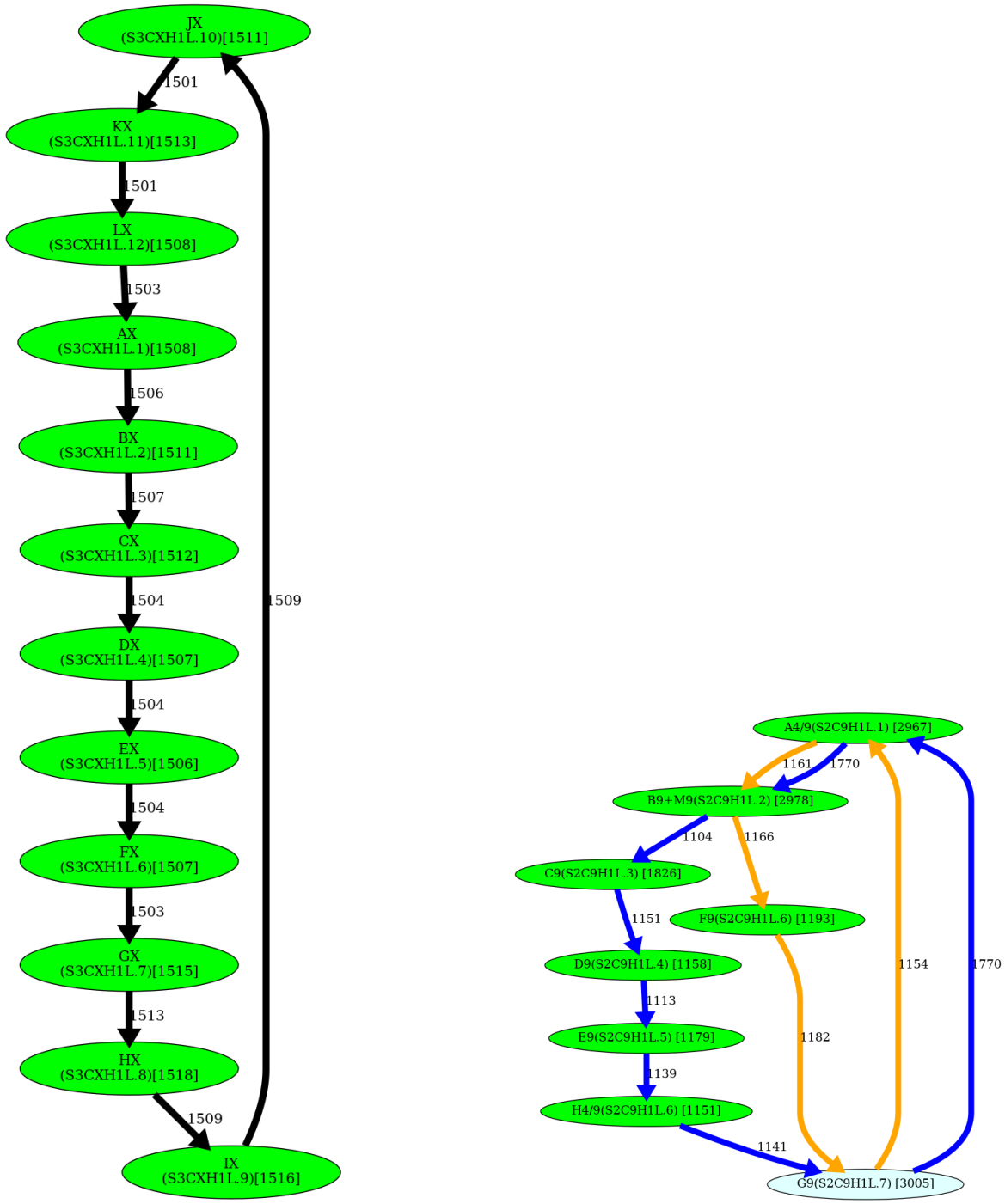

Рисунок 3.13 — Графы мономеров для cenX и cen9. На рисунке слева изображен граф мономеров для cenX. Он представляет собой один цикл, который соответствует каноническому ПВП в cenX. На рисунке справа изображен граф мономеров для cen9, в котором вычисленный вручную ПВП cen9 [115] показан синим циклом, а еще один часто встречающийся ПВП изображен желтым. Данный пример противоречит постулату эволюции центромер, так как часто проходимый желтый цикл содержит мономер, не принадлежащий синему циклу.

Разложение на ПВП центромеры хромосомы X, произведенное HORmon, показано на рисунке 3.14. Набор мономеров представлен двенадцатью исходными мономерами, обозначенными заглавными латинским буквами A,B,... K,L. Разложения на ПВП для всех других центромер можно найти в документе «Разложения ПВП». Описание процесса создания консенсусных нуклеотидных последовательностей всех мономеров и ПВП можно найти в дополнительном примечании 8 к [19]. Однако из-за недостатка данных неясно, являются ли эти консенсусные последовательности репрезентативными для других людей.

$p_{10\text{-}12}p_{5\text{-}6}IL p_{2\text{-}3}Ep_{9\text{-}12}p_{2\text{-}5}c_6^{22}p_{7\text{-}12}c_1^{3}p_{2\text{-}6}c_7^{3}p_{8\text{-}6}c_7p_{8\text{-}5}c_6^{95}p_{6\text{-}9}c_{10}^{26}Kc_{12}^{3}\underline{\text{LINE}}p_{1\text{-}7}c_8^{128}p_{11\text{-}7}c_8^{2}p_{11\text{-}7}c_8^{11}p_{11\text{-}7}c_8p_{11\text{-}7}c_8p_{1\text{-}7}c_8p_{11\text{-}7}c_8^{11}Kc_{12}^{8}p_{7\text{-}5}c_6^{2}p_{6\text{-}11}c_{12}p_{7\text{-}10}c_{11}^{50}p_{12\text{-}6}c_7^{174}p_{1\text{-}9}c_{10}^{240}p_{1\text{-}9}c_{10}^{8}p_{12\text{-}9}c_{10}^{18}p_{12\text{-}9}c_{10}^{8}p_{12\text{-}9}c_{10}^{19}p_{12\text{-}9}c_{10}^{8}p_{12\text{-}9}c_{10}^{13}p_{1\text{-}9}c_{10}p_{1\text{-}11}c_{12}^{41}p_{7\text{-}11}c_{12}^{6}p_{7\text{-}11}c_{12}^{5}p_{7\text{-}11}c_{12}^{7}p_{7\text{-}11}c_{12}^{16}p_{7\text{-}11}c_{12}^{4}p_{7\text{-}3}c_4^{13}p_{5\text{-}3}c_4^{21}Ec_6^{14}p_{7\text{-}3}c_4^{52}p_{5\text{-}2}c_3^{42}p_{11\text{-}5}c_6^{57}p_{6\text{-}12}c_1^{4}p_{8\text{-}11}c_{12}^{3}p_{8\text{-}11}c_{12}p_{8\text{-}12}c_1p_{8\text{-}4}c_5^{32}p_{6\text{-}2}c_3p_{5\text{-}1}c_2^{24}p_{3\text{-}4}c_5^{2}p_{6\text{-}4}^{4}p_{6\text{-}12}c_1^{94}p_{2\text{-}12}c_1^{11}p_{2\text{-}7}c_8^{14}p_{9\text{-}7}c_8^{5}p_{9\text{-}7}c_8^{21}p_{11\text{-}4}c_5^{28}p_{6\text{-}2}c_3^{87}Dp_{9\text{-}10}$

Рисунок 3.14 — Разложение cenX на ПВП. Канонический ПВП cenX, состоящий из 12 мономеров, обозначен $M_1 \dots M_{12} = AB \dots KL$. Каждое вхождение ПВП, которое начинается с определенного мономера M_i, обозначается c_i (показано красным), а каждое вхождение частичного ПВП, включающее мономеры от i до j, обозначается как p_{i-j}. Обозначение c_i^m (p_{i-j}^m) используется для представления m последовательных вхождений канонического (частичного) ПВП. Наиболее часто встречающиеся частичные мономеры p_{3-7}, p_{7-3} и p_{5-2} в cenX окрашены в синий, зеленый и коричневый цвета соответственно. Длина разложения cenX на ПВП составляет 72 знака, данное разложение включает в себя 1486 полных ПВП, которые образуют 34 кластера последовательных вхождений одинаковых ПВП. Только 257 из 18 089 (1.4%) мономерных блоков в cenX не покрыты полными ПВП. Положение LINE-элемента обозначено подписью «LINE».

Хотя некоторые пары и тройки центромер имеют похожие мономеры, анализ показал, что их ПВП различаются по крайней мере в клеточной линии CHM13. Например, редакционное расстояние между консенсусом ПВП в cen13 и cen21 составляет 20, что указывает на расхождение в 1%, в то время как редакционное расстояние между консенсусом ПВП в cen14 и cen22 намного ниже и составляет всего 3, что указывает на расхождение в 0.2%. В случае центромер 1, 5 и 19 канонический ПВП состоит из двух негибридных мономеров согласно [115], тогда как наш анализ выявил ПВП, состоящий из шести негибридных мономеров. Это несоответствие, вероятно, связано с отсутствием полной сборки генома на момент анализа, описанного в [115]. Примечательно, что редак-

ционное расстояние между ПВП в cen1 и cen5 велико: имеет разницу в 34 или расхождение в 3.3%.

3.6 Заключение

Недавние усовершенствования в технологиях секвенирования длинных прочтений произвели революцию в сборке генома человека и позволили проводить структурные и эволюционные исследования центромерных последовательностей человека на основе полных сборок центромер, а не отдельных прочтений или эталонных моделей [25].

StringDecomposer — это первый инструмент, разработанный специально для разложения центромерных сборок и прочтений на мономеры. Как было показано, StringDecomposer может точно преобразовывать даже подверженные ошибкам центромерные прочтения в монопрочтения.

CentromereArchitect и HORmon создают полные аннотации центромерных последовательностей, получая на вход только последовательности и консенсусный мономер без дополнительной информации о мономерном составе последовательности. В то время как CentromereArchitect представил строгие в вычислительном отношении определения мономеров и ПВП и обеспечил начальный шаг для процедуры вывода мономеров, инструмент HORmon был разработан для вывода аннотации центромер, которая согласуется с постулатом эволюции центромер и предыдущими усилиями по аннотации центромер вручную. HORmon не только извлекает мономеры и ПВП из «живых» альфа-сателлитных последовательностей центромер, но также создает их консенсусные последовательности. Аннотации центромер HORmon уже предоставили значительную информацию для анализа регионов CENP-A, полученных с помощью ChIP-seq, и метилирования ДНК в центромерах [19; 20].

В то же время HORmon тестировался только на «живых» центромерах, поэтому автоматические процедуры аннотирования других альфа-сателлитных доменов в настоящее время не описаны, и трудно предсказать результаты HORmon на мономерах с большей вариацией. Кроме того, опять же из-за отсутствия полных центромерных сборок, постулат эволюции центромер не был

должным образом проверен по сравнению с альтернативными моделями эволюции центромер [113; 129; 130].

Кроме того, как показано в [25; 110], последовательности центромер очень разнообразны в человеческой популяции, и мы полагаем, что концепция графа мономеров, введенная в [19], поможет в проблеме сравнения центромер нескольких индивидов.

Инструменты, описанные в этой главе, обещают внести свой вклад в продолжающиеся изучение центромерных участков и ускорить исследования центромер человека.

Заключение

Результат работы и практическая значимость. Несмотря на то, что общие методы сборки генома и анализа сборки геномов предоставляют большой массив ценной информации, до сих пор существуют сложные в обработке регионы последовательности генома. В первых двух частях диссертации были описаны алгоритмы для прикладывания биологических последовательностей на граф сборки, которые позволяют улучшить анализ регионов, фрагментированных в финальной сборке. В третьей части были предложены алгоритмы и инструменты для анализа таких сложных в сборке участков как последовательности центромер человека. До недавнего времени центромеры не были собраны в цельные последовательности, поэтому не было специальных инструментов для их анализа.

Инструмент SPAligner эффективно и точно прикладывает биологические последовательности к графам сборки, созданным SPAdes и другими популярными сборщиками для коротких прочтений. Этот подход помогает включать дополнительную информацию из длинных прочтений с большим количеством ошибок для улучшения качества сборки и информацию из баз данных белков для идентификации последовательностей белков, не собранных в финальной сборке. Кроме того, SPAligner был реализован как часть сборщика hybridSPAdes, а модуль выравнивания длинных прочтений SPAligner повысил производительность hybridSPAdes. В работе [16] представлено важное применение режима поиска белковых последовательностей SPAligner — поиск инсектицидных токсинов непосредственно в графах сборки, а не во фрагментированных (обычно метагеномных) сборках. Конвейер ORFograph был успешно использован для выявления ранее необнаруженных потенциальных ГИБ из графов сборки всех доступных наборов данных *B.thuringuses* Illumina из NCBI.

Инструменты для аннотации центромер StringDecomposer, CentromereArchitect и HORmon открыли новые возможности для анализа сборок и прочтений центромер, что было успешно продемонстрировано в ходе работы консорциума T2T [20]. В работе над StringDecomposer было показано, как могут образовываться гибридные мономеры. Инструменты CentromereArchitect и HORmon позволили создать и проверить аннотации впервые собранных центромерных последовательностей генома человека. CentromereArchitect и HORmon представ-

ляют собой полностью автоматические подходы, позволяющие выполнить анализ центромерных последовательностей на большом количестве сборок центромер человека.

Дальнейшее развитие данной работы. Недавние усовершенствования методов поиска зерен и алгоритмов выравнивания последовательностей на графы открыли огромный потенциал для улучшения реализации как SPAligner, так и ORFograph [131–133]. В то же время еще одним важным направлением будущей работы является адаптация методов SPAligner к выравниванию на граф сборки, построенный из точных и довольно длинных прочтений PacBio HiFi [134–136] и ультрадлинных, но подверженных ошибкам прочтений ONT. Недавно выпущенный гибридный сборщик Verkko [137] способен реконструировать 20 из 46 диплоидных хромосом без пробелов, комбинируя точные HiFi прочтения и сверхдлинные прочтения ONT. Сборщик Verkko использует GraphAligner [49] для выравнивания прочтений ONT на граф сборки, построенный на основе HiFi прочтений.

Валидация результатов HORmon на большем количестве сборок центромер человека имеет важное значение для анализа центромер человека. Это поможет понять границы и потенциальные улучшения постулата эволюции центромер и предлагаемой процедуры вывода ПВП. Кроме того, применение StringDecomposer, CentromereArchitect и HORmon представляет собой первые шаги к автоматическому аннотированию центромер не только у людей, но и у других видов.

Благодарности. Автор выражает благодарность своему научному руководителю, Павлу Певзнеру, всем коллегам, участвовавшим в совместных проектах и исследованиях — Ивану Александрову, Дмитрию Антипову, Антону Банкевичу, Андрею Бзикадзе, Антону Коробейникову, Ольге Кунявской и Сергею Нурку. Особая признательность коллегам и друзьям, прочитавшим черновой вариант диссертации и предоставившим полезные комментарии, которые помогли улучшить качество текста — Дмитрию Антипову, Андрею Бзикадзе, Татьяне Лихоманенко, Алексею Рогожникову и Яне Сафоновой. Работа была бы невозможна без поддержки со стороны моей семьи и близких, особенно моего мужа Михаила Дворкина.

Также благодарность выражается Российскому Научному Фонду и Санкт-Петербургскому государственному университету, предоставившим финансирование данной работы.

Список сокращений и акронимов

AAI	amino acid identity; аминокислотная идентичность
Bt	Bacillus thuringiensis
CA	CentromereArchitect
CDS	кодирующая последовательность
Cry	Crystal; кристаллические белки
Cyt	Cytolic; цитотоксические белки
NCBI	National Center for Biotechnology Information
NCRF	Noise-Cancelling Repeat Finder
NGS	next-generation sequencing
OLC	Overlap-Layout-Consensus
ONT	Oxford Nanopore Technology
PB, PacBio	Pacific Biosciences
SD	StringDecomposer
SRA	Short Read Archive; база данных коротких прочтений
T2T (Консорциум)	Консорциум Telomere-to-Telomere
TRF	TandemRepeatsFinder
Vip	вегетативный инсектицидный белок
ГИБ	гены инсектицидных белков
ДНК	дезоксирибонуклеиновая кислота
ОАГ	ориентированный ациклический граф
ОЗУ	оперативное запоминающее устройство
ОРС	открытая рамка считывания
ОКМД	одиночный поток команд, множественный поток данных
ПВП	повтор высокого порядка
ПЦР	полимеразная цепная реакция
РНК	рибонуклеиновая кислота
СММ	скрытая марковская модель

Словарь терминов

CDS — кодирующая последовательность.

***De novo* сборка** — процесс объединения прочтений в исходную последовательность без какой-либо предварительной информации, т. е. с нуля.

LINEs (long interspersed nuclear element) — длинные последовательности ДНК в геноме эукариот, представляющие собой ретротранспозоны, не содержащие длинных концевых повторов.

Overlap-Layout-Consensus — вычислительный метод, предназначенный для сборки из небольшого количества точных прочтений (изначально чтений Сэнгера).

Seed-and-extend подход — эвристический подход для решения задачи выравнивания. Сначала осуществляется поиск зерен (seeds) между запросом и целевой последовательностью. Затем каждое зерно расширяется (extend) и соединяется с выравниваниями других зерен в полное выравнивание.

Алгоритм Дейкстры — алгоритм поиска кратчайших путей между двумя вершинами в графе.

Ассемблер — программа, способная объединять чтения в более длинные точные последовательности, называемые контигами или каркасами.

Биопестициды — типы пестицидов, полученные из живых организмов или продуктов натурального происхождения.

Выравнивание — способ упорядочивания биологических последовательностей для выявления областей сходства.

Гаплотип — уникальная комбинация генетических вариаций, наследуемая одновременно от одного из родителей.

Гибридная сборка — процесс получения сборки генома с использованием данных различной природы (например, короткие и длинные прочтения).

Гибридный мономер — мономер, представляющий собой конкатенацию двух и более негибридных мономеров.

Граф вариаций — граф последовательностей, который отображает генетические вариации в сборках геномов нескольких индивидов.

Глобальное выравнивание последовательностей – выравнивание двух последовательностей по всей их длине, таким образом, чтобы каждому символу либо соответствовал символ другой строки, либо символ пробела.

Граф сборки — сжатый граф де Брюйна (в данной работе); также может обозначать любой граф последовательности, используемый в процессе сборки.

Граф де Брюйна — граф, построенный для одной или нескольких строк с использованием всех их подстрок длины k. Граф де Брюйна для строки S и параметра k определяется как ориентированный граф с множеством вершин $V = \{S_i^k| = 1 \dots |S| - k + 1\}$ и множество ребер $E = \{(u,v)|u \in V, v \in V, \exists i : u + v_k^1 = S_i^{k+1}\}$.

Граф последовательностей — граф, сконструированный из одной и более последовательностей, отображающий связи между ними.

Граф строк — граф последовательностей, где вершина представляет собой прочтение, а ребро — перекрытие между двумя прочтениями.

Длинные прочтения — прочтения, сгенерированные с помощью технологий третьего поколения.

Зерно — точный k-мер, который встречается и в запросе, и в целевой последовательности в решении задачи выравнивания.

Идентичность — метрика похожести двух последовательностей, которая считается как разница между длиной запроса (длиной выравнивания) и редакционным расстоянием, деленная на длину запроса (длину выравнивания).

Индекс — структура данных, позволяющая быстро находить конкретную информацию в большом наборе данных.

Канонический ПВП — циклическая последовательность частых негибридных мономеров, которая со временем эволюционировала в доступную на данный момент последовательность центромеры.

Кинетохор — белковый комплекс, отвечающий за деление клеток.

Кодон — тройка нуклеотидов, которая соответствует аминокислоте.

Контиг — непрерывный фрагмент ДНК или РНК, найденный ассемблером.

Короткие прочтения — прочтения, сгенерированные с помощью технологий секвенирования следующего поколения.

K-мер — последовательность длины K.

Метагеномика — изучение генетического материала, извлеченного непосредственно из образцов окружающей среды.

Метка — нуклеотидная последовательность, соответствующая ребру или вершине в графе сборки. Меткой пути называется конкатенация меток ребер пути в случае, когда в вершинах длина меток равна нулю.

Мономер (альфа-сателлит) — последовательность длины 171, которая представляет собой единицу строения ПВП в центромере человека.

Обход в ширину (в глубину) — алгоритмы обхода вершин графа в специальном порядке.

Оптимальное глобальное (полуглобальное) выравнивание — глобальное (полуглобальное) выравнивание последовательности запроса на целевую последовательность, которое соответствует самому низкому штрафу для преобразования запроса в целевую последовательность (подстроку целевой последовательности) в соответствии с заданной схемой штрафов.

Парные прочтения — прочтения, секвенированные с разных концов одного фрагмента ДНК.

Перекрытие — максимальный суффикс одного прочтения практически идентичный префиксу другого прочтения.

Повтор — фрагмент, который встречается несколько раз в геноме или другой нуклеиновой последовательности.

Повтор высшего порядка (ПВП) — нуклеотидная последовательность, которая формирует центромеру в конкретной хромосоме человека, повторяясь в ней тысячи раз.

Покрытие K-мерами — количество раз, когда интересующий k-мер встречается во всех чтениях. Для более длинной последовательности оно вычисляется как среднее число для всех k-меров в этой последовательности.

Полуглобальное выравнивание последовательностей — выравнивание двух последовательностей, запроса и целевой последовательности, когда необходимо наилучшее глобальное выравнивание для запроса и подстроки целевой последовательности среди всех возможных подстрок целевой последовательности.

Путь оптимального полуглобального выравнивание последовательности на граф сборки — путь, метка которого служит целевой последовательностью для наилучшего полуглобального выравнивания последовательности запроса по сравнению со всеми остальными метками путей графа сборки.

Прочтение — фрагмент ДНК, последовательность которого определяется в процессе секвенирования.

Размер вставки — длина фрагмента ДНК/РНК, который секвенируется с обоих концов и который «вставляется» между адаптерами.

Разрешение повторов — процесс разрешения неоднозначностей во время сборки последовательности, вызванных геномными повторами.

Реверскомплиментарная последовательность — последовательность, полученная из исходной с помощью следующих операций: каждый нуклеотид заменяется его комплементарным, и вся последовательность переворачивается.

Редакционное расстояние — схема штрафов, при которой операции замены, удаления и вставки символа имеют штраф 1.

Референсная сборка — процесс определения точной последовательности с использованием известной последовательности генома или транскриптома того же вида.

Сборщик — инструменты для соединения прочтений в более длинные и точные последовательности генома, называемые контигами.

Секвенирование — процесс определения порядка нуклеотидов в молекулах ДНК.

Сжатый граф де Брюйна — граф, полученный из графа де Брюйна заменой каждого неразветвляющегося пути одним ребром.

Скрытая марковская модель (СММ) — в данной работе статистическая модель, описывающая последовательности, принадлежащие определенному домену, встречающемуся в генах. СММ может быть представлена в виде ориентированного ациклического графа.

Старт-кодон — первый кодон матричной РНК, с которого начинается трансляция белка в рибосоме.

Стоп-кодон — кодон, кодирующий прекращение синтеза полипептидной цепи рибосомой.

Схема штрафов — набор штрафов за операции преобразования строки запроса в контексте решения задачи выравнивания последовательностей, например таких, как вставка, удаление и замена символа. Часто схема штрафов также задает вознаграждение за соответствие символов в выравнивании двух последовательностей.

Формат FASTA — текстовый формат для представления последовательностей нуклеотидов или аминокислот.

Хроматида — одна из двух одинаковый половин хромосомы, которые соединены в специальном месте, называемом центромерой.

Хромосома — длинная молекула ДНК, которая хранит часть генетического материала организма.

Центромера — область хромосомы, где клетка расщепляет свою ДНК в процессе клеточного деления с помощью специального белкового комплекса *кинетохора*, собранного в пределах той же области центромеры.

Якорь — области высокого сходства между последовательностью запроса и ребрами графа сборки в задаче выравнивания последовательности к графу.

Список литературы

1. *Peltola Hannu, Söderlund Hans, Ukkonen Esko.* SEQAID: A DNA sequence assembling program based on a mathematical model // *Nucleic Acids Research.* — 1984. — Vol. 12, no. 1Part1. — Pp. 307–321. — [Online; accessed 2022-12-06].

2. *Huang Xiaoqiu.* A contig assembly program based on sensitive detection of fragment overlaps // *Genomics.* — 1992. — 9. — Vol. 14, no. 1. — Pp. 18–25. — [Online; accessed 2022-12-06].

3. *Myers Jr Eugene W.* A history of DNA sequence assembly // *it - Information Technology.* — 2016. — jun 25. — Vol. 58, no. 3. — Pp. 126–132. — [Online; accessed 2022-09-03].

4. hybridSPAdes: An algorithm for hybrid assembly of short and long reads / Dmitry Antipov, Anton Korobeynikov, Jeffrey S. McLean, Pavel A. Pevzner // *Bioinformatics.* — 2015. — nov 20. — Vol. 32, no. 7. — Pp. 1009–1015. — [Online; accessed 2022-09-03].

5. *Salmela Leena, Rivals Eric.* LoRDEC: Accurate and efficient long read error correction // *Bioinformatics.* — 2014. — aug 26. — Vol. 30, no. 24. — Pp. 3506–3514. — [Online; accessed 2022-09-03].

6. Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads / Ryan R. Wick, Louise M. Judd, Claire L. Gorrie, Kathryn E. Holt // *PLOS Computational Biology.* — 2017. — jun 8. — Vol. 13, no. 6. — P. e1005595. — [Online; accessed 2022-09-03].

7. A graph-based approach to diploid genome assembly / Shilpa Garg, Mikko Rautiainen, Adam M Novak et al. // *Bioinformatics.* — 2018. — jun 27. — Vol. 34, no. 13. — Pp. i105–i114. — [Online; accessed 2022-09-03].

8. Initial sequencing and analysis of the human genome / Eric S. Lander, Lauren M. Linton, Bruce Birren et al. // *Nature.* — 2001. — feb 15. — Vol. 409, no. 6822. — Pp. 860–921. — [Online; accessed 2022-09-03].

9. The complete sequence of a human genome / Sergey Nurk, Sergey Koren, Arang Rhie et al. // *Science*. — 2022. — 4. — Vol. 376, no. 6588. — Pp. 44–53. — [Online; accessed 2022-09-20].

10. *HiFi Reads - Highly accurate long-read sequencing*. — https://www.pacb.com/technology/hifi-sequencing/. — 2019. — jul 19. — [Online; accessed 2023-01-14].

11. Nanopore sequencing and assembly of a human genome with ultra-long reads / Miten Jain, Sergey Koren, Karen H Miga et al. // *Nature Biotechnology*. — 2018. — jan 29. — Vol. 36, no. 4. — Pp. 338–345. — [Online; accessed 2023-01-14].

12. *Bzikadze Andrey V., Pevzner Pavel A.* Automated assembly of centromeres from ultra-long error-prone reads // *Nature Biotechnology*. — 2020. — jul 14. — Vol. 38, no. 11. — Pp. 1309–1316. — [Online; accessed 2022-09-20].

13. Classification and monomer-by-monomer annotation dataset of suprachromosomal family 1 alpha satellite higher-order repeats in hg38 human genome assembly / L.I. Uralsky, V.A. Shepelev, A.A. Alexandrov et al. // *Data in Brief*. — 2019. — 6. — Vol. 24. — P. 103708. — [Online; accessed 2022-12-03].

14. Assembling single-cell genomes and mini-metagenomes from chimeric MDA products / Sergey Nurk, Anton Bankevich, Dmitry Antipov et al. // *Journal of Computational Biology*. — 2013. — 10. — Vol. 20, no. 10. — Pp. 714–737. — [Online; accessed 2022-09-03].

15. SPAligner: Alignment of long diverged molecular sequences to assembly graphs / Tatiana Dvorkina, Dmitry Antipov, Anton Korobeynikov, Sergey Nurk // *BMC Bioinformatics*. — 2020. — 7. — Vol. 21, no. S12. — [Online; accessed 2022-08-27].

16. ORFograph: Search for novel insecticidal protein genes in genomic and metagenomic assembly graphs / Tatiana Dvorkina, Anton Bankevich, Alexei Sorokin et al. // *Microbiome*. — 2021. — jun 28. — Vol. 9, no. 1. — [Online; accessed 2022-08-30].

17. *Dvorkina Tatiana, Bzikadze Andrey V, Pevzner Pavel A*. The string decomposition problem and its applications to centromere analysis and assembly //

Bioinformatics. — 2020. — jul 1. — Vol. 36, no. Supplement_1. — Pp. i93–i101. — [Online; accessed 2022-09-20].

18. CentromereArchitect: Inference and analysis of the architecture of centromeres / Tatiana Dvorkina, Olga Kunyavskaya, Andrey V Bzikadze et al. // *Bioinformatics*. — 2021. — jul 1. — Vol. 37, no. Supplement_1. — Pp. i196–i204. — [Online; accessed 2022-09-20].

19. Automated annotation of human centromeres with HORmon / Olga Kunyavskaya, Tatiana Dvorkina, Andrey V. Bzikadze et al. // *Genome Research*. — 2022. — may 11. — Vol. 32, no. 6. — Pp. i1137–i1151.

20. Complete genomic and epigenetic maps of human centromeres / Nicolas Altemose, Glennis A. Logsdon, Andrey V. Bzikadze et al. // *Science*. — 2022. — 4. — Vol. 376, no. 6588. — [Online; accessed 2022-09-20].

21. *Li Heng*. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. — https://arxiv.org/abs/1303.3997. — 2013. — mar 16.

22. *Li Heng*. Minimap2: Pairwise alignment for nucleotide sequences // *Bioinformatics*. — 2018. — may 10. — Vol. 34, no. 18. — Pp. 3094–3100. — [Online; accessed 2022-09-03].

23. *Shlemov Alexander, Korobeynikov Anton*. Tech. Rep.: : Cold Spring Harbor Laboratory, 2019. — feb 27. — [Online; accessed 2022-08-27].

24. Identifying periodic occurrences of a template with applications to protein structure / Vincent A. Fischetti, Gad M. Landau, Jeanette P. Schmidt, Peter H. Sellers // Combinatorial Pattern Matching. — Berlin, Heidelberg: Springer Berlin Heidelberg, 1992. — Pp. 111–120. — [Online; accessed 2022-12-03].

25. Centromere reference models for human chromosomes X and Y satellite arrays / Karen H. Miga, Yulia Newton, Miten Jain et al. // *Genome Research*. — 2014. — apr 1. — Vol. 24, no. 4. — Pp. 697–707.

26. *Sanger F., Nicklen S., Coulson A. R.* DNA sequencing with chain-terminating inhibitors // *Proceedings of the National Academy of Sciences*. — 1977. — 12. — Vol. 74, no. 12. — Pp. 5463–5467. — [Online; accessed 2022-09-03].

27. Comparison of next-generation sequencing systems / Lin Liu, Yinhu Li, Siliang Li et al. // *Journal of Biomedicine and Biotechnology*. — 2012. — Vol. 2012. — Pp. 1–11. — [Online; accessed 2022-09-03].

28. Sanger confirmation is required to achieve optimal sensitivity and specificity in next-generation sequencing panel testing / Wenbo Mu, Hsiao-Mei Lu, Jefferey Chen et al. // *The Journal of Molecular Diagnostics*. — 2016. — 11. — Vol. 18, no. 6. — Pp. 923–932. — [Online; accessed 2022-09-03].

29. Long walk to genomics: History and current approaches to genome sequencing and assembly / Alice Maria Giani, Guido Roberto Gallo, Luca Gianfranceschi, Giulio Formenti // *Computational and Structural Biotechnology Journal*. — 2020. — Vol. 18. — Pp. 9–19. — [Online; accessed 2022-09-03].

30. Real-Time DNA sequencing from single polymerase molecules / John Eid, Adrian Fehr, Jeremy Gray et al. // *Science*. — 2009. — jan 2. — Vol. 323, no. 5910. — Pp. 133–138. — [Online; accessed 2023-01-14].

31. The Oxford Nanopore MinION: Delivery of nanopore sequencing to the genomics community / Miten Jain, Hugh E. Olsen, Benedict Paten, Mark Akeson // *Genome Biology*. — 2016. — nov 25. — Vol. 17, no. 1. — [Online; accessed 2023-01-14].

32. *Pop M.* Genome assembly reborn: Recent computational challenges // *Briefings in Bioinformatics*. — 2009. — may 29. — Vol. 10, no. 4. — Pp. 354–366. — [Online; accessed 2022-09-03].

33. *Myers Eugene W.* Toward simplifying and accurately formulating fragment assembly // *Journal of Computational Biology*. — 1995. — 1. — Vol. 2, no. 2. — Pp. 275–290. — [Online; accessed 2022-09-03].

34. *Pevzner Pavel A., Tang Haixu, Waterman Michael S.* An Eulerian path approach to DNA fragment assembly // *Proceedings of the National Academy of Sciences*. — 2001. — aug 14. — Vol. 98, no. 17. — Pp. 9748–9753. — [Online; accessed 2022-09-03].

35. *Zerbino Daniel R., Birney Ewan.* Velvet: Algorithms for de novo short read assembly using de Bruijn graphs // *Genome Research*. — 2008. — mar 18. — Vol. 18, no. 5. — Pp. 821–829. — [Online; accessed 2022-08-30].

36. ABySS: A parallel assembler for short read sequence data / Jared T. Simpson, Kim Wong, Shaun D. Jackman et al. // *Genome Research*. — 2009. — feb 27. — Vol. 19, no. 6. — Pp. 1117–1123. — [Online; accessed 2023-01-21].

37. SOAPdenovo2: An empirically improved memory-efficient short-read de novo assembler / Ruibang Luo, Binghang Liu, Yinlong Xie et al. // *GigaScience*. — 2012. — 12. — Vol. 1, no. 1. — [Online; accessed 2023-01-21].

38. SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing / Anton Bankevich, Sergey Nurk, Dmitry Antipov et al. // *Journal of Computational Biology*. — 2012. — 5. — Vol. 19, no. 5. — Pp. 455–477. — [Online; accessed 2022-08-27].

39. *Needleman Saul B., Wunsch Christian D.* A general method applicable to the search for similarities in the amino acid sequence of two proteins // *Journal of Molecular Biology*. — 1970. — 3. — Vol. 48, no. 3. — Pp. 443–453. — [Online; accessed 2022-09-03].

40. *Langmead Ben, Salzberg Steven L.* Fast gapped-read alignment with Bowtie 2 // *Nature Methods*. — 2012. — mar 4. — Vol. 9, no. 4. — Pp. 357–359. — [Online; accessed 2022-09-03].

41. *Chikhi Rayan, Rizk Guillaume*. Space-efficient and exact de Bruijn graph representation based on a Bloom filter // *Algorithms for Molecular Biology*. — 2013. — 1. — Vol. 8, no. 1. — [Online; accessed 2022-08-27].

42. MEGAHIT: An ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph / Dinghua Li, Chi-Man Liu, Ruibang Luo et al. // *Bioinformatics*. — 2015. — jan 20. — Vol. 31, no. 10. — Pp. 1674–1676. — [Online; accessed 2022-08-27].

43. Variation graph toolkit improves read mapping by representing genetic variation in the reference / Erik Garrison, Jouni Sirén, Adam M Novak et al. // *Nature Biotechnology*. — 2018. — 10. — Vol. 36, no. 9. — Pp. 875–879. — [Online; accessed 2022-09-03].

44. BrownieAligner: Accurate alignment of Illumina sequencing data to de Bruijn graphs / Mahdi Heydari, Giles Miclotte, Yves Van de Peer, Jan Fostier //

BMC Bioinformatics. — 2018. — sep 4. — Vol. 19, no. 1. — [Online; accessed 2022-09-03].

45. On the complexity of sequence to graph alignment / Chirag Jain, Haowen Zhang, Yu Gao, Srinivas Aluru // Lecture Notes in Computer Science. — Cham: Springer International Publishing, 2019. — Pp. 85–100. — [Online; accessed 2022-09-03].

46. Sequence alignment on directed graphs / Vaddadi Naga Sai Kavya, Kshitij Tayal, Rajgopal Srinivasan, Naveen Sivadasan // *Journal of Computational Biology*. — 2019. — 1. — Vol. 26, no. 1. — Pp. 53–67. — [Online; accessed 2022-09-03].

47. Read mapping on de Bruijn graphs / Antoine Limasset, Bastien Cazaux, Eric Rivals, Pierre Peterlongo // *BMC Bioinformatics*. — 2016. — jun 16. — Vol. 17, no. 1. — [Online; accessed 2022-09-03].

48. *Sirén Jouni*. Indexing variation graphs // 2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX) / Society for Industrial and Applied Mathematics. — Philadelphia, PA: 2017. — 1. — [Online; accessed 2022-09-03].

49. *Rautiainen Mikko, Marschall Tobias*. GraphAligner: Rapid and versatile sequence-to-graph alignment // *Genome Biology*. — 2020. — sep 24. — Vol. 21, no. 1. — [Online; accessed 2022-09-03].

50. MUMmer4: A fast and versatile genome alignment system / Guillaume Marçais, Arthur L. Delcher, Adam M. Phillippy et al. // *PLOS Computational Biology*. — 2018. — jan 26. — Vol. 14, no. 1. — P. e1005944. — [Online; accessed 2022-09-03].

51. *Rautiainen Mikko, Mäkinen Veli, Marschall Tobias*. Bit-parallel sequence-to-graph alignment // *Bioinformatics*. — 2019. — mar 9. — Vol. 35, no. 19. — Pp. 3599–3607. — [Online; accessed 2022-09-03].

52. *Li, Feng, Chu*. The design and construction of reference pangenome graphs with minigraph // *Genome Biology*. — 2020. — oct 16. — Vol. 21, no. 1. — Pp. 1–19.

53. MegaGTA: A sensitive and accurate metagenomic gene-targeted assembler using iterative de Bruijn graphs / Dinghua Li, Yukun Huang, Chi-Ming Leung et al. // *BMC Bioinformatics*. — 2017. — 10. — Vol. 18, no. S12. — [Online; accessed 2022-09-03].

54. *Myers Eugene W.* AnO(ND) difference algorithm and its variations // *Algorithmica*. — 1986. — 11. — Vol. 1, no. 1-4. — Pp. 251–266. — [Online; accessed 2022-09-04].

55. *Amir Amihood, Lewenstein Moshe, Lewenstein Noa.* Pattern matching in hypertext // *Journal of Algorithms*. — 2000. — 4. — Vol. 35, no. 1. — Pp. 82–99. — [Online; accessed 2022-09-04].

56. *Gotoh Osamu.* An improved algorithm for matching biological sequences // *Journal of Molecular Biology*. — 1982. — 12. — Vol. 162, no. 3. — Pp. 705–708. — [Online; accessed 2022-09-04].

57. *Navarro Gonzalo.* A guided tour to approximate string matching // *ACM Computing Surveys*. — 2001. — 3. — Vol. 33, no. 1. — Pp. 31–88. — [Online; accessed 2022-09-04].

58. *Šošić Martin, Šikić Mile.* Edlib: A C/C ++ library for fast, exact sequence alignment using edit distance // *Bioinformatics*. — 2017. — jan 31. — Vol. 33, no. 9. — Pp. 1394–1395. — [Online; accessed 2022-09-04].

59. *Daily Jeff.* Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments // *BMC Bioinformatics*. — 2016. — feb 10. — Vol. 17, no. 1. — [Online; accessed 2022-09-05].

60. *Pearson William R.* Selecting the right similarityscoring matrix // *Current Protocols in Bioinformatics*. — 2013. — 10. — Vol. 43, no. 1. — [Online; accessed 2022-09-05].

61. MinION-based long-read sequencing and assembly extends the *Caenorhabditis elegans* reference genome / John R Tyson, Nigel J O'Neil, Miten Jain et al. // *Genome research*. — 2018. — 2. — Vol. 28, no. 2. — Pp. 266–274.

62. *Nagarajan Niranjan, Pop Mihai.* Sequence assembly demystified // *Nature Reviews Genetics*. — 2013. — jan 29. — Vol. 14, no. 3. — Pp. 157–167. — [Online; accessed 2022-09-11].

63. Genome-resolved metagenomics identifies genetic mobility, metabolic interactions, and unexpected diversity in perchlorate-reducing communities / Tyler P. Barnum, Israel A. Figueroa, Charlotte I. Carlström et al. // *The ISME Journal*. — 2018. — feb 23. — Vol. 12, no. 6. — Pp. 1568–1581. — [Online; accessed 2022-09-11].

64. Accurate, multi-kb reads resolve complex populations and detect rare microorganisms / Itai Sharon, Michael Kertesz, Laura A. Hug et al. // *Genome Research*. — 2015. — feb 9. — Vol. 25, no. 4. — Pp. 534–543. — [Online; accessed 2022-09-11].

65. Characterization of metagenomes in urban aquatic compartments reveals high prevalence of clinically relevant antibiotic resistance genes in wastewaters / Charmaine Ng, Martin Tay, Boonfei Tan et al. // *Frontiers in Microbiology*. — 2017. — nov 16. — Vol. 8. — [Online; accessed 2022-09-11].

66. Tech. Rep.: / Michael Feldgarden, Vyacheslav Brover, Daniel H. Haft et al.: Cold Spring Harbor Laboratory, 2019. — feb 15. — [Online; accessed 2022-09-11].

67. Basic local alignment search tool / Stephen F. Altschul, Warren Gish, Webb Miller et al. // *Journal of Molecular Biology*. — 1990. — 10. — Vol. 215, no. 3. — Pp. 403–410. — [Online; accessed 2022-09-11].

68. Bandage: Interactive visualization ofde novogenome assemblies: Fig. 1. / Ryan R. Wick, Mark B. Schultz, Justin Zobel, Kathryn E. Holt // *Bioinformatics*. — 2015. — jun 22. — Vol. 31, no. 20. — Pp. 3350–3352. — [Online; accessed 2022-08-30].

69. What are Biopesticides? — https://www.epa.gov/ingredients-used-pesticide-products/what-are-biopesticides. — 2015. — aug 31. — [Online; accessed 2022-08-23].

70. Molecular and Insecticidal Characterization of a Novel Cry-Related Protein from Bacillus Thuringiensis Toxic against Myzus persicae / Leopoldo Palma, Delia Muñoz, Colin Berry et al. // *Toxins*. — 2014. — nov 1. — Vol. 6, no. 11.

71. Genetically engineered crops help support conservation biological control / Jörg Romeis, Steven E. Naranjo, Michael Meissle, Anthony M. Shelton //

Biological Control. — 2019. — 3. — Vol. 130. — Pp. 136–154. — [Online; accessed 2022-08-23].

72. *Ohba Michio, Mizuki Eiichi, Uemori Akiko*. Parasporin, a New Anticancer Protein Group from Bacillus thuringiensis // *Anticancer Research*. — 2009. — jan 1. — Vol. 29, no. 1. — Pp. 427–433.

73. Field-evolved resistance by western corn rootworm to multiple Bacillus thuringiensis toxins in transgenic maize / Aaron J. Gassmann, Jennifer L. Petzold-Maxwell, Eric H. Clifton et al. // *Proceedings of the National Academy of Sciences*. — 2014. — mar 17. — Vol. 111, no. 14. — Pp. 5141–5146. — [Online; accessed 2022-08-23].

74. *Höfte H, Whiteley H R*. Insecticidal crystal proteins of Bacillus thuringiensis // *Microbiological Reviews*. — 1989. — 6. — Vol. 53, no. 2. — Pp. 242–255. — [Online; accessed 2022-08-23].

75. Prediction of insecticidal activity of Bacillus thuringiensis strains by polymerase chain reaction product profiles / N B Carozzi, V C Kramer, G W Warren et al. // *Applied and Environmental Microbiology*. — 1991. — 11. — Vol. 57, no. 11. — Pp. 3057–3061. — [Online; accessed 2022-08-24].

76. Screening and identification ofvipgenes inBacillus thuringiensisstrains / C.S. Hernández-Rodríguez, A. Boets, J. Van Rie, J. Ferré // *Journal of Applied Microbiology*. — 2009. — 7. — Vol. 107, no. 1. — Pp. 219–225. — [Online; accessed 2022-08-24].

77. *Juárez-Pérez V M, Ferrandis M D, Frutos R*. PCR-based approach for detection of novel Bacillus thuringiensis cry genes // *Applied and Environmental Microbiology*. — 1997. — 8. — Vol. 63, no. 8. — Pp. 2997–3002. — [Online; accessed 2022-08-24].

78. *Lin Yi, Fang Guangwei, Peng Kun*. Characterization of the highly variable cry gene regions of Bacillus thuringiensis strain ly4a3 by PCR-SSCP profiling and sequencing // *Biotechnology Letters*. — 2006. — dec 7. — Vol. 29, no. 2. — Pp. 247–251. — [Online; accessed 2022-08-24].

79. *Noguera Pedro A., Ibarra Jorge E*. Detection of New cry Genes of Bacillus thuringiensis by Use of a Novel PCR Primer System // *Applied and Environ-*

mental Microbiology. — 2010. — sep 15. — Vol. 76, no. 18. — Pp. 6150–6155. — [Online; accessed 2022-08-24].

80. *Besemer J., Borodovsky M.* GeneMark: Web software for gene finding in prokaryotes, eukaryotes and viruses // *Nucleic Acids Research.* — 2005. — jul 1. — Vol. 33. — Pp. W451–W454. — [Online; accessed 2022-08-25].

81. Prodigal: Prokaryotic gene recognition and translation initiation site identification / Doug Hyatt, Gwo-Liang Chen, Philip F LoCascio et al. // *BMC Bioinformatics.* — 2010. — mar 8. — Vol. 11, no. 1. — [Online; accessed 2022-08-25].

82. Identifying bacterial genes and endosymbiont DNA with Glimmer / Arthur L. Delcher, Kirsten A. Bratke, Edwin C. Powers, Steven L. Salzberg // *Bioinformatics.* — 2007. — jan 19. — Vol. 23, no. 6. — Pp. 673–679. — [Online; accessed 2022-08-25].

83. *Zhu Wenhan, Lomsadze Alexandre, Borodovsky Mark.* Ab initio gene identification in metagenomic sequences // *Nucleic Acids Research.* — 2010. — apr 19. — Vol. 38, no. 12. — Pp. e132–e132. — [Online; accessed 2022-08-25].

84. Gene and translation initiation site prediction in metagenomic sequences / Doug Hyatt, Philip F. LoCascio, Loren J. Hauser, Edward C. Uberbacher // *Bioinformatics.* — 2012. — jul 12. — Vol. 28, no. 17. — Pp. 2223–2230. — [Online; accessed 2022-08-25].

85. Gene prediction with Glimmer for metagenomic sequences augmented by classification and clustering / David R. Kelley, Bo Liu, Arthur L. Delcher et al. // *Nucleic Acids Research.* — 2011. — nov 17. — Vol. 40, no. 1. — Pp. e9–e9. — [Online; accessed 2022-08-25].

86. BtToxin_Digger: A comprehensive and high-throughput pipeline for mining toxin protein genes from Bacillus thuringiensis / Hualin Liu, Jinshui Zheng, Dexin Bo et al. // *Bioinformatics.* — 2021. — jul 9. — Vol. 38, no. 1. — Pp. 250–251. — [Online; accessed 2022-08-24].

87. UniProt: The universal protein knowledgebase in 2021 / Alex Bateman, Maria-Jesus Martin, Sandra Orchard et al. // *Nucleic Acids Research.* — 2020. — nov 25. — Vol. 49, no. D1. — Pp. D480–D489. — [Online; accessed 2022-08-27].

88. Introduction to algorithms / Thomas H.. Cormen, Thomas H Cormen, Charles E Leiserson et al. — MIT Press, 2001. — [Online; accessed 2022-08-27].

89. *Eddy S. R.* Profile hidden Markov models // *Bioinformatics*. — 1998. — oct 1. — Vol. 14, no. 9. — Pp. 755–763. — [Online; accessed 2022-08-27].

90. A structure-based nomenclature for Bacillus thuringiensis and other bacteria-derived pesticidal proteins / Neil Crickmore, Colin Berry, Suresh Panneerselvam et al. // *Journal of Invertebrate Pathology*. — 2021. — 11. — Vol. 186. — P. 107438. — [Online; accessed 2022-08-29].

91. The Pfam protein families database in 2019 / Sara El-Gebali, Jaina Mistry, Alex Bateman et al. // *Nucleic Acids Research*. — 2018. — oct 24. — Vol. 47, no. D1. — Pp. D427–D432. — [Online; accessed 2022-08-29].

92. metaSPAdes: A new versatile metagenomic assembler / Sergey Nurk, Dmitry Meleshko, Anton Korobeynikov, Pavel A. Pevzner // *Genome Research*. — 2017. — mar 15. — Vol. 27, no. 5. — Pp. 824–834. — [Online; accessed 2022-08-30].

93. Plasmid detection and assembly in genomic and metagenomic data sets / Dmitry Antipov, Mikhail Raiko, Alla Lapidus, Pavel A. Pevzner // *Genome Research*. — 2019. — may 2. — Vol. 29, no. 6. — Pp. 961–968. — [Online; accessed 2022-08-30].

94. No more tears: Mining sequencing data for novel bt cry toxins with cryprocessor / Anton E. Shikov, Yury V. Malovichko, Rostislav K. Skitchenko et al. // *Toxins*. — 2020. — mar 23. — Vol. 12, no. 3. — P. 204. — [Online; accessed 2022-08-30].

95. Critical Assessment of Metagenome Interpretation—a benchmark of metagenomics software / Alexander Sczyrba, Peter Hofmann, Peter Belmann et al. // *Nature Methods*. — 2017. — oct 2. — Vol. 14, no. 11. — Pp. 1063–1071. — [Online; accessed 2022-08-29].

96. ART: A next-generation sequencing read simulator / Weichun Huang, Leping Li, Jason R. Myers, Gabor T. Marth // *Bioinformatics*. — 2011. — dec 23. — Vol. 28, no. 4. — Pp. 593–594. — [Online; accessed 2022-08-29].

97. *Buchfink Benjamin, Xie Chao, Huson Daniel H.* Fast and sensitive protein alignment using DIAMOND // *Nature Methods*. — 2014. — nov 17. — Vol. 12, no. 1. — Pp. 59–60. — [Online; accessed 2022-08-29].

98. Revision of the Nomenclature for the Bacillus thuringiensis Pesticidal Crystal Proteins / N. Crickmore, D. R. Zeigler, J. Feitelson et al. // *Microbiology and Molecular Biology Reviews*. — 1998. — 9. — Vol. 62, no. 3. — Pp. 807–813. — [Online; accessed 2022-08-29].

99. *Edgar.* MUSCLE: A multiple sequence alignment method with reduced time and space complexity // *BMC Bioinformatics*. — 2004. — aug 19. — Vol. 5, no. 1. — Pp. 1–19.

100. *Price Morgan N., Dehal Paramvir S., Arkin Adam P.* FastTree 2 – approximately maximum-likelihood trees for large alignments // *PLoS ONE*. — 2010. — mar 10. — Vol. 5, no. 3. — P. e9490. — [Online; accessed 2022-08-30].

101. Lineage-specific plasmid acquisition and the evolution of specialized pathogens inBacillus thuringiensisand theBacillus cereusgroup / Guillaume Méric, Leonardos Mageiros, Ben Pascoe et al. // *Molecular Ecology*. — 2018. — 4. — Vol. 27, no. 7. — Pp. 1524–1540. — [Online; accessed 2022-08-29].

102. *Jeong Haeyoung, Choi Soo-Keun, Park Seung-Hwan.* Genome Sequences of Bacillus thuringiensis Serovar kurstaki Strain BP865 and B. thuringiensis Serovar aizawai Strain HD-133 // *Genome Announcements*. — 2017. — feb 2. — Vol. 5, no. 5. — [Online; accessed 2022-08-29].

103. *Coil David, Jospin Guillaume, Darling Aaron E.* A5-miseq: An updated pipeline to assemble microbial genomes from Illumina MiSeq data // *Bioinformatics*. — 2014. — oct 22. — Vol. 31, no. 4. — Pp. 587–589. — [Online; accessed 2022-08-29].

104. *Seemann T.* Prokka: Rapid prokaryotic genome annotation // *Bioinformatics*. — 2014. — mar 18. — Vol. 30, no. 14. — Pp. 2068–2069. — [Online; accessed 2022-08-29].

105. NCBI prokaryotic genome annotation pipeline / Tatiana Tatusova, Michael DiCuccio, Azat Badretdin et al. // *Nucleic Acids Research*. — 2016. — jun 24. — Vol. 44, no. 14. — Pp. 6614–6624. — [Online; accessed 2023-02-20].

106. Geospatial resolution of human and bacterial diversity with city-scale metagenomics / Ebrahim Afshinnekoo, Cem Meydan, Shanin Chowdhury et al. // *Cell Systems*. — 2015. — 7. — Vol. 1, no. 1. — Pp. 72–87. — [Online; accessed 2022-08-30].

107. Recovery of nearly 8,000 metagenome-assembled genomes substantially expands the tree of life / Donovan H. Parks, Christian Rinke, Maria Chuvochina et al. // *Nature Microbiology*. — 2017. — sep 11. — Vol. 2, no. 11. — Pp. 1533–1542. — [Online; accessed 2022-08-30].

108. *McKinley Kara L., Cheeseman Iain M.* The molecular basis for centromere identity and function // *Nature Reviews Molecular Cell Biology*. — 2015. — nov 25. — Vol. 17, no. 1. — Pp. 16–29. — [Online; accessed 2022-09-26].

109. Sequences associated with centromere competency in the human genome / Karen E. Hayden, Erin D. Strome, Stephanie L. Merrett et al. // *Molecular and Cellular Biology*. — 2013. — feb 15. — Vol. 33, no. 4. — Pp. 763–772. — [Online; accessed 2022-09-20].

110. *Suzuki Yuta, Myers Eugene W., Morishita Shinichi.* Rapid and ongoing evolution of repetitive sequence structures in human centromeres // *Science Advances*. — 2020. — dec 11. — Vol. 6, no. 50. — [Online; accessed 2022-09-20].

111. *Smith George P.* Evolution of repeated DNA sequences by unequal crossover // *Science*. — 1976. — feb 13. — Vol. 191, no. 4227. — Pp. 528–535. — [Online; accessed 2023-03-11].

112. *Malik Harmit S., Henikoff Steven.* Major evolutionary transitions in centromere complexity // *Cell*. — 2009. — 9. — Vol. 138, no. 6. — Pp. 1067–1082. — [Online; accessed 2023-03-11].

113. *Rice William R.* Tech. Rep.: : Cold Spring Harbor Laboratory, 2019. — aug 10. — [Online; accessed 2022-12-03].

114. *Waye J S, Willard H F.* Chromosome-specific alpha satellite DNA: Nucleotide sequence analysis of the 2.0 kilobasepair repeat from the human X chromosome. // *Nucleic Acids Research*. — 1985. — apr 25. — Vol. 13, no. 8.

115. *McNulty Shannon M, Sullivan Beth A.* Alpha satellite DNA biology: Finding function in the recesses of the genome // *Chromosome research : an international journal on the molecular, supramolecular and evolutionary aspects of chromosome biology.* — 2018. — 9. — Vol. 26, no. 3. — Pp. 115–138.

116. Telomere-to-telomere assembly of a complete human X chromosome / Miga, Koren, Rhie et al. // *Nature.* — 2020. — jul 14. — Vol. 585, no. 7823. — Pp. 79–84.

117. *Harris Robert S, Cechova Monika, Makova Kateryna D.* Noise-cancelling repeat finder: Uncovering tandem repeats in error-prone long-read sequencing data // *Bioinformatics.* — 2019. — jul 10. — Vol. 35, no. 22. — Pp. 4809–4811. — [Online; accessed 2022-12-03].

118. TandemTools: Mapping long reads and assessing/improving assembly quality in extra-long tandem repeats / Mikheenko, Bzikadze, Gurevich et al. // *Bioinformatics.* — 2020. — jul 13. — Vol. 36, no. Supplement_1. — Pp. i75–i83.

119. *Benson G.* Tandem repeats finder: A program to analyze DNA sequences // *Nucleic Acids Research.* — 1999. — jan 1. — Vol. 27, no. 2. — Pp. 573–580. — [Online; accessed 2022-12-03].

120. Alpha-CENTAURI: Assessing novel centromeric repeat sequence variation with long read sequencing / Volkan Sevim, Ali Bashir, Chen-Shan Chin, Karen H. Miga // *Bioinformatics.* — 2016. — feb 24. — Vol. 32, no. 13. — Pp. 1921–1924. — [Online; accessed 2022-12-03].

121. Interspersed repeats are found predominantly in the “old“ alpha-satellite families / Alexei E Kazakov, Valery A Shepelev, Irina G Tumeneva et al. // *Genomics.* — 2003. — Vol. 82, no. 6. — Pp. 619–627. — URL: `https://www.sciencedirect.com/science/article/pii/S0888754303001824`.

122. ColorHOR–novel graphical algorithm for fast scan of alpha satellite higher-order repeats and HOR annotation for GenBank sequence of human genome / V. Paar, N. Pavin, M. Rosandic et al. // *Bioinformatics.* — 2004. — oct 27. — Vol. 21, no. 7. — Pp. 846–852. — [Online; accessed 2022-12-03].

123. Organization and evolution of primate centromeric DNA from whole-genome shotgun sequence data / Can Alkan, Mario Ventura, Nicoletta Archidiacono

et al. // *PLoS Computational Biology*. — 2007. — sep 28. — Vol. 3, no. 9. — P. e181. — [Online; accessed 2022-12-03].

124. *Compeau Phillip, Pevzner Pavel*. Bioinformatics algorithms: An active learning approach. — 1986. — 6. — [Online; accessed 2022-12-03].

125. *Matroud Atheer A., Tuffley Christopher P., Hendy Michael D.* An algorithm to solve the motif alignment problem for approximate nested tandem repeats in biological sequences // *Journal of Computational Biology*. — 2011. — 9. — Vol. 18, no. 9. — Pp. 1211–1218. — [Online; accessed 2022-12-03].

126. *Matroud A. A., Hendy M. D., Tuffley C. P.* NTRFinder: A software tool to find nested tandem repeats // *Nucleic Acids Research*. — 2011. — nov 25. — Vol. 40, no. 3. — Pp. e17–e17. — [Online; accessed 2022-12-03].

127. *Gusfield Dan*. Algorithms on strings, trees, and sequences: Computer science and computational biology. — Cambridge University Press, 1997. — may 28. — [Online; accessed 2022-12-03].

128. *Miga Karen H., Alexandrov Ivan A.* Variation and evolution of human centromeres: A field guide and perspective // *Annual Review of Genetics*. — 2021. — nov 23. — Vol. 55, no. 1. — Pp. 583–602. — [Online; accessed 2022-12-03].

129. Evolution of satellite DNAs from the genus Palorus–experimental evidence for the "library"hypothesis / N. Mestrovic, M. Plohl, B. Mravinac, D. Ugarkovic // *Molecular Biology and Evolution*. — 1998. — aug 1. — Vol. 15, no. 8. — Pp. 1062–1068. — [Online; accessed 2022-12-03].

130. *Henikoff Steven, Ahmad Kami, Malik Harmit S.* The centromere paradox: Stable inheritance with rapidly evolving DNA // *Science*. — 2001. — aug 10. — Vol. 293, no. 5532. — Pp. 1098–1102. — [Online; accessed 2022-12-03].

131. *Ivanov Pesho, Bichsel Benjamin, Vechev Martin*. Tech. Rep.: : Cold Spring Harbor Laboratory, 2021. — nov 8. — [Online; accessed 2022-12-10].

132. Tech. Rep.: / Jun Ma, Manuel Cáceres, Leena Salmela et al.: Cold Spring Harbor Laboratory, 2022. — jan 7. — [Online; accessed 2022-12-10].

133. *Chandra Ghanshyam, Jain Chirag*. Tech. Rep.: : Cold Spring Harbor Laboratory, 2022. — sep 1. — [Online; accessed 2022-12-10].

134. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm / Cheng, Concepcion, Feng et al. // *Nature Methods*. — 2021. — feb 1. — Vol. 18, no. 2. — Pp. 170–175.

135. HiCanu: Accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads / Sergey Nurk, Brian P. Walenz, Arang Rhie et al. // *Genome Research*. — 2020. — aug 14. — Vol. 30, no. 9. — Pp. 1291–1305. — [Online; accessed 2022-12-10].

136. Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads / Anton Bankevich, Andrey V. Bzikadze, Mikhail Kolmogorov et al. // *Nature Biotechnology*. — 2022. — feb 28. — Vol. 40, no. 7. — Pp. 1075–1081. — [Online; accessed 2023-02-18].

137. Telomere-to-telomere assembly of diploid chromosomes with Verkko / Mikko Rautiainen, Sergey Nurk, Brian P. Walenz et al. // *Nature Biotechnology*. — 2023. — feb 16. — [Online; accessed 2023-02-18].

Список рисунков

1.1 Алгоритм инструмента SPAligner для выравнивания нуклеотидной последовательности на граф сборки 19
1.2 Алгоритм SPAligner для выравнивания аминокислотной последовательности на граф сборки 24
1.3 Подграф метагеномной сборки с реконструированной последовательностью белка бета-лактамазы 31

2.1 Гистограммы длин инсектицидных белков Cry и Vip и всех бактериальных генов из базы данных Uniprot 35
2.2 Общая схема конвейера ORFograph 36
2.3 Противоречащие и непротиворечащие пути в графе сборки. 39
2.4 Разнообразие Cry1-последовательностей, идентифицированных в данных изолированных бактерий 47
2.5 Подграф графа метагеномной сборки с выравниваниями Cry4Aa и Cry4Ba генов 48

3.1 Устройство центромеры хромосомы X 51
3.2 Разложения на ПВП для центромеры хромосомы X и центромеры хромосомы 8. 52
3.3 Представление графа разложения строки в виде «книги» 57
3.4 Распределение Identity (ось OX) и IdentityDiff (ось OY) для выравнивания мономеров в центромерных областях. 62
3.5 Распределение идентичности в процентах для 12 мономеров cenX и символа пробела «?». 64
3.6 Пример выравнивания монопрочтения к моноцетромере 65
3.7 Матрицы замен между мономерами для подходов AC и SD 67
3.8 Анализ нестандартных ПВП ABCDEFGHIJFGHIJKL 68
3.9 Схема работы CentromereArchitect 71
3.10 Схема работы инструмента HORmon 74
3.11 Замена вершин гибридных мономеров в графе мономеров на ребра для cen5 75
3.12 Замена вершин гибридных мономеров в графе мономеров на ребра для cen8 76

3.13 Графы мономеров для cenX и cen9 77
3.14 Разложение cenX на ПВП 78

Список таблиц

1 Данные для тестирования SPAligner и GraphAligner 27
2 Результаты SPAligner и GraphAligner при выравнивании прочтений PacBio на графы сборки из коротких прочтений 28
3 Результаты SPAligner и GraphAligner при выравнивании прочтений ONT на графы сборки из коротких прочтений 29
4 Информация о семи симулированных наборах данных 43
5 Информация о процентной идентичности между шестью выбранными Cry-белками 44
6 Результаты тестирования конвейера ORFograph на пяти симулированных наборах данных разной сложности 45
7 Статистика ошибок выравниваний монопрочтений на моноцентромеру для инструментов AC и SD для 12 мономеров . . 66
8 Статистика ошибок выравниваний монопрочтений на моноцентромеру для инструментов AC и SD для 13 мономеров . . 69

ST.PETERSBURG STATE UNIVERSITY

as a manuscript

Tatiana Dvorkina

DEVELOPMENT OF ALGORITHMS FOR ANALYSIS OF GENOME ASSEMBLY GRAPHS AND ASSEMBLED GENOMES

Scientific specialty 1.5.8
Mathematical biology, bioinformatics

Dissertation is submitted for the degree of
Candidate of Physico-Mathematical Sciences

Translated from Russian

Scientific advisor:
Candidate of Science in Physics and Mathematics
Pavel A. Pevzner

Saint Petersburg — 2023

Table of contents

P.

Introduction 113

Chapter 1. Alignment of biological sequences to assembly graph . . 119
1.1 Introduction 119
1.1.1 Sequencing methods 119
1.1.2 Genome assembly and assembly graphs 120
1.1.3 Alignment of two sequences 121
1.1.4 Sequence-to-graph alignment problem in assembly graphs . . 122
1.2 Alignment of long reads to assembly graphs 124
1.2.1 Algorithm overview 124
1.2.2 Sequence to graph alignment via alignment graphs 126
1.3 Alignment of amino acid sequences to assembly graphs 127
1.3.1 Amino acid sequence alignment pipeline 128
1.3.2 Amino acid sequence alignment scoring scheme 129
1.3.3 Alignment graph for amino acid sequence alignment 130
1.4 Results 131
1.4.1 Long reads to graph alignment benchmarking 131
1.4.2 Amino acid alignment benchmarking 133
1.5 Conclusions 135

Chapter 2. Search for insecticidal protein genes in assembly graph 136
2.1 Introduction 136
2.1.1 *Bacillus thuringiensis* insecticidal proteins 136
2.1.2 Insecticidal protein genes search 137
2.2 ORFograph pipeline overview 138
2.3 Aligning known IPGs and domain HMMs to the assembly graph . . 140
2.4 Start and stop codons search 140
2.5 Generation of complete CDSs 141
2.5.1 Analyzing HMM positions within gene sequences 141

P.

2.5.2 Filtering putative proteins that conflict with contigs 142
2.6 Clustering potential proteins and selecting representatives 143
2.7 Results 143
2.7.1 ORFograph benchmarking overview 143
2.7.2 Simulated datasets 144
2.7.3 Isolate IPG-containing bacteria samples from the NCBI database 147
2.7.4 New York City subway metagenome 149
2.8 Conclusions 150

Chapter 3. Automatic analysis of centromere sequence assemblies 151
3.1 Introduction 151
3.1.1 Human centromeres and their structure 151
3.1.2 Centromere Evolution Postulate 152
3.1.3 Towards automatic centromere annotation 154
3.2 String Decomposition Problem 157
3.2.1 String Decomposition Graph 157
3.2.2 The StringDecomposer algorithm 158
3.2.3 Transformation from the nucleotide alphabet to the block alphabet 160
3.3 StringDecomposer Results 162
3.3.1 Datasets 162
3.3.2 Benchmarking StringDecomposer on error-prone reads 163
3.3.3 Identifing first hybrid monomers 164
3.4 Centromere annotation 168
3.4.1 CentromereArchitect pipeline overview 168
3.4.2 HORmon pipeline overview 171
3.5 Annotation of human centromeres 173
3.6 Conclusions 177

Conclusions 178

List of abbverations and acronyms 180

P.

Glossary 182

References 187

Table of figures 203

Table of tables 204

Introduction

Motivation. The problem of obtaining a complete genomic sequence is paramount for biological research and various applied fields, from medical studies to agriculture. The process of determining the nucleic acid sequence of a genome is called *sequencing*. Sequences provided by a sequencing procedure are called *reads*. The length of reads is typically much shorter than the lengths of explored genomes. The process of concatenating reads into longer genomic sequences, *contigs*, is called *genome assembly*.

There are two main types of genome assembly — reference-based assembly and *de novo* assembly. The reference-based assembly utilizes information from already assembled genome sequences of the reference organism from the same species. The *de novo* assembly solves a more complicated problem of obtaining contigs without any prior knowledge about the order of reads.

First *de novo* genome assemblers used the Overlap-Layout-Consensus (OLC) approach to obtain contigs [1; 2]. With the rise of the second-generation sequencing (also known as the next-generation sequencing or NGS) technologies in 2005, the number of reads increased while their length decreased from around 1000 base pairs (bp) to 100-350 bp, so the OLC approach has been replaced by algorithms based on sequence graphs [3]. In the process of genome assembly, a set of reads and their potential connections are usually represented as a graph structure called *assembly graph*.

Reads from the second-generation sequencing technologies (or short reads) became the basis for automatic genome assembly and genome analysis, but their short length makes it challenging to untangle complex regions with a high number of repeats. Only third-generation sequencing (TGS) technologies, first described in 2008-2009, provided long enough reads to capture the structure of some complex repetitive parts of the genome sequence. However, in comparison to short NGS reads, first TGS reads (or long reads) had very low accuracy (10-15% error rate) and again did not allow performing assembly of long complex regions using solely TGS data. In order to improve assembly quality, automatic methods based on mixing data from different sources started to spread out [4; 5].

Assembly graphs contain more information about genome sequence rather than final contigs as they store additional connections between reads. And in combination

with other sources of genomic information, e.g., other types of reads or protein sequences, they may be used in various applications, from straightforward read error correction [5] to the improvement of complex assemblies [4; 6] and reference-free haplotype reconstruction [7]. These applications usually require to find the potential location of a read or a protein in the assembly graph, or, more formally, find the best alignment of nucleotide or amino acid sequence to assembly graphs. The first two chapters of this dissertation describe the solution to the problem of how to align biological sequences to assembly graphs (specifically, an assembly graph built from short reads) and its application to the problem of computing potential coding amino acids directly from the assembly graphs.

Centromere is a specific part of the chromosome where two chromatids (two copies of DNA that form the chromosome) are attached to each other. For over two decades after the publication of the first human reference genome [8], centromere sequences remained the longest unassembled regions of the human genome due to their large length and highly repetitive structure [9]. Arrival of long reads [10] with low error rates and ultralong reads [11] resulted in the first full human centromere assemblies [9;12]. In order to analyze new centromere assemblies, scientists need tools to describe their structure, especially to decompose centromeres into repetitive units [13]. This problem is crucial for the centromere assembly and centromere evolution studies. The third chapter of the dissertation is dedicated to the algorithms and instruments for centromere analysis.

The main **goals of this work** include:

- design and implementation of algorithms for alignment of nucleotide and amino acid sequences to assembly graphs;
- design and implementation of a pipeline to extract protein sequences from assembly graphs;
- design and implementation of algorithms for human centromere structure analysis.

The practical value and the novelty. The following instruments have been made available to the public as a result of this work:

- SPAligner: standalone tool for aligning biological sequences to de Bruijn assembly graphs built from NGS data (based on SPAdes [14] infrastructure) (`cab.spbu.ru/software/spaligner/`);

- ORFograph: the pipeline that extracts proteins sequences scattered over several contigs from de Bruijn assembly graphs built from NGS data (`https://github.com/ablab/orf-search`);
- StringDecomposer: standalone tool to accurately decompose centromeric sequences into given monomers (`cab.spbu.ru/software/stringdecomposer/`);
- HORmon: the pipeline to annotate human centromeres (`cab.spbu.ru/software/HORmon/`);
- CentromereArchitect: the first pipeline for human centromere annotation, including a module for initial monomer extraction (now a part of HORmon) (`cab.spbu.ru/software/stringdecomposer/`).

New methods for solving the problem of aligning sequences to an assembly graph are proposed in [15]. SPAligner [15] is the only tool that allows to align both nucleotide and amino acid sequences to assembly graphs. The quality of read alignment using SPAligner is comparable to alternative instruments.

ORFograph [16] is the only pipeline that allows extracting potential coding sequences of an arbitrary protein family using assembly graphs. In the course of work on the ORFograph tool, new methods for filtering potential protein sequences based on information from graph connections and contigs were proposed. It has also been shown that ORFograph can be used on already analyzed data to extract new potential genes not found in the original studies.

StringDecomposer, CentromereArchitect, and HORmon played the key roles in obtaining and analyzing the first human centromere sequence assemblies produced by the Telomere-to-Telomere (T2T) Consortium effort [17]. The work on these tools made it possible to understand the principles of the formation of centromeric sequences, such as the formation of so-called hybrid monomers, and to describe these principles in the form of a set of rules. The HORmon pipeline is the first and so far the only existing tool to automatically analyze human centromere assemblies and large sets of centromeric reads, guided by current understanding of the evolution of centromere sequences.

Methods. The main sequence-to-graph alignment algorithm in SPAligner tool [15] is based on hybridSPAdes read-to-graph alignment algorithm described in [4]. It uses the classical seed-and-extend idea as in common sequence-to-sequence alignment methods, such as BWA-MEM [18] or Minimap2 [19]. SPAligner uses BWA-MEM [18] to find short accurate alignments between a query sequence and assembly graph edges, and performs dynamic programming to extend these short alignments

into full alignment of the query onto the assembly graph. In [15] we described a new modification of dynamic programming approach to extend short alignments that lowers time complexity, and extended the initial dynamic programming algorithm proposed in [4] for amino acid-to-graph alignment case.

ORFograph pipeline [16] combines the sequence-to-graph alignment tool, SPAligner [15], and the tool PathRacer [20], that aligns Hidden Markov Models to assembly graphs, to extract potential coding sequences, or open reading frames (ORFs), from complex assembly graphs and to perform ORF filtering to identify potential genes fragmented across multiple contigs. In [16] new algorithms to search for ORFs in assembly graphs and to perform ORF filtering based on the assembly graph structure are described.

StringDecomposer [21] takes a set of monomers and centromeric sequences as input and uses wrap-around dynamic programming [22] to find the best partition of sequences into monomers. It was the first tool developed as a part of the T2T Consortium effort to automate centromere sequence analysis, as many centromeric monomers had already been extracted from short reads and artificially built Centromeric Reference Models [13; 23].

HORmon [24] represents the final pipeline for full centromere sequence analysis. It uses CentromereArchitect [25] to extract the initial set of monomers and StringDecomposer to divide the centromere into monomers. After that, HORmon extracts high-order repeats (HORs) using de Bruijn graph concept and partitions the centromere into monomers and HORs.

The SPAligner tool was implemented in C++, while all other tools were predominantly implemented in Python.

Work presentation. The results of this work were presented in oral and poster presentations at several international conferences:

1. HORmon: automated annotation of human centromeres, Poster, BiATA 2021, virtual conference;
2. CentromereArchitect: inference and analysis of the architecture of centromeres, Talk, ISMB 2021, virtual conference;
3. The string decomposition problem, Talk, ISMB 2020, virtual conference;
4. SPAligner: A tool for alignment of biological sequences to assembly graph, Talk, BiATA 2019, St. Petersburg, Russia.
5. GAligner: A tool for alignment of long reads to assembly graph, Poster, BiATA 2018, St. Petersburg, Russia.

In addition to poster and oral presentations at international conferences, these works were presented at scientific seminars of various laboratories.

Publications. In addition to listed presentations and talks, the results were published as five papers [15; 16; 21; 24; 25] in journals indexed in Web of Science and Scopus databases. All five papers are published in journals currently ranked as Q1.

Main results of the dissertation.

- The SPAligner tool was developed based on the module for aligning long reads to the assembly graph of the hybridSPAdes assembler [4]. The SPAligner tool not only proposed and implemented new algorithms for aligning long reads, but also extended the functionality to align amino acid sequences. Our benchmarking has shown that SPAligner's long read alignment mode gives the same or better results with respect to the state-of-art solutions.
- ORFograph is the only method that performs protein extraction directly from assembly graphs. It can be used to identify new proteins and to extract long proteins that can not be assembled.
- The StringDecomposer, CentromereArchitect, and HORmon tools are the first tools for automated analysis of centromere sequences. The StringDecomposer implements the only algorithm that *always* finds the optimal decomposition of the centromeric sequence into given blocks. The CentromereArchitect and HORmon tools were successfully used to analyze the structure of the first complete human centromere assemblies and helped to form a clearer understanding of the principles of evolution of centromere sequences.

Personal contribution. In the SPAligner project, the author was the main contributor to the algorithm development and implementations. The author obtained the primary results and carried out all testing pipelines. Although this project was based on hybridSPAdes assembler [4] sequence-to-graph alignment module and its codebase, the implementation was substantially redesigned improving the quality and speed of previous efforts.

In the ORFograph project, the author developed and implemented all algorithms, performed all tests and experiments.

The author is the sole contributor to the StringDecomposer project and is a key contributor to the CentromereArchitect and HORmon projects. In

CentromereArchitect and HORmon projects, the author developed and implemented HOR decomposition procedures.

In all projects, the author generated results and actively participated in publication preparation.

Structure of the work. The dissertation includes an introduction, 3 chapters, and a conclusion. The text of the dissertation contains 96 pages, including 22 figures and 8 tables. References include 137 citations.

The first chapter formulates the problem of aligning long reads and amino acid sequences to assembly graphs from short reads and describes the algorithms and the pipeline of the SPAligner tool presented in [15].

In **the second chapter** the idea of extracting proteins from complex assembly graphs is discussed. The chapter describes the ORFograph pipeline that extracts coding sequences of already known proteins and sequences of potential proteins, and demonstrates its results on an important class of insecticide toxins. This chapter is based on [16].

The third chapter describes the core ideas and methods implemented in StringDecomposer, CentromereArchitect, and HORmon tools and presents the results of these tools on the latest human genome assembly [9]. The material of these chapter is based on [21; 24; 25].

The conclusion discusses the influence of algorithms and instruments proposed in previous chapters on the scientific community and presents potential directions for their development.

Chapter 1. Alignment of biological sequences to assembly graph

1.1 Introduction

1.1.1 Sequencing methods

The first sequencing method was proposed by British biochemist Frederick Sanger and his colleagues in 1977 [26] and was named the Sanger sequencing method (or chain breaking method). Using this method, regions of DNA up to 900 bp long could be sequenced. Sanger sequencing was used in the Human Genome Project [8], which was aimed at building the first accurate human genome assembly. Produced reads were aligned on top of each other based on overlapping parts to assemble sequences of large DNA fragments.

The main disadvantage of this method is its high cost relative to other technologies ($2 400 000 for sequencing 1 billion bp) [27]. Genomes are now sequenced using less expensive and faster methods. However, Sanger sequencing is sometimes used to verify the results of a genome assembly produced by other technologies, as well as to sequence individual DNA fragments, such as fragments used in DNA cloning [28].

Sanger sequencing was replaced by next-generation sequencing (NGS) technologies. One of the most popular approaches among next-generation sequencing methods is the technology proposed by Illumina/Solexa in 2005. In addition to Illumina sequencers, there are many other NGS methods, and though each of them uses its own unique technology, most of them share a set of properties that distinguish them from Sanger sequencing [27]:

- **Small length of the resulting DNA fragment:** e.g., on the Illumina platform the length of one read varies from 50 to 350 bp.
- **Cheap:** NGS genome sequencing costs $5-$150 per 1 billion bp.
- **High speed:** Many sequencing reactions occur at the same time, so results are obtained much faster.

In 2008-2009, third-generation sequencing (TGS) technologies emerged [29]. Their main difference from NGS methods is that, with completely new technologies

for reading the genomic sequence, they produce much longer reads. At the moment, the length of one read can be up to several millions nucleotides. The two most common technologies were proposed by Pacific Biosciences (PacBio) [30] and Oxford Nanopore Technologies (ONT) [31]. Until recently, the main disadvantage of reads from such sequencing technologies was a high percentage of errors (around 10-15% error rate).

1.1.2 Genome assembly and assembly graphs

Assembly of a DNA sequence is a complex combinatorial problem that arose in the early 1980s with the first DNA sequencing projects. The first fully automated assembly attempts resulted in the Overlap-Layout-Consensus (OLC) paradigm [32], where the assembly was performed in three stages:

- Find all overlaps between reads with an accuracy consistent with the expected error rate of a sequencing technology.
- Next, using overlaps, order the reads and create a mosaic assembly of the genome.
- Create a consensus sequence of reads covering a given area.

With the advent of second-generation technologies, the number of reads and their length have changed a lot, and the OLC approach has gradually been replaced by algorithms based on sequence graphs. The two main paradigms were the string graph approach proposed in [33] and the de Bruijn graph approach proposed in [34]. In string graphs, each vertex represents a read and an edge represents the overlap of two reads. String graphs became the basis for Celera and Newbler [3; 32] assembly algorithms.

At the same time, de Bruijn graphs have become more popular for assembly construction and are used in well-known assemblers such as Velvet [35], ABySS [36], SOAPdenovo [37], SPAdes [38], etc. De Bruijn assembly graph is built by extracting all k-mers (substrings of length k) from reads and connecting k-mers that go one after another in at least one read. The final de Bruijn graph has k-mers as edges and $k-1$-mers as nodes. The *compacted* de Bruijn graph can be constructed from the ordinary de Bruijn graph by replacing all non-branching paths with single edges. Each vertex and each edge can be described by its own nucleotide sequence, called

label. The length of a vertex sequence is predefined within one graph and equals $k - 1$, while the length of an edge sequence can vary. Genome assembly contigs represent a set of paths in the assembly graph. The SPAligner tool [15] presented in this chapter aligns biological sequences to compacted de Bruijn graphs.

1.1.3 Alignment of two sequences

Consider two nucleotide (or amino acids) sequences, Q and T. *Alignment* of two sequences can be considered as the placement of one sequence under the other in order to find similar areas. *Global alignment* of Q and T is an alignment of the sequences along the entire length, so that each symbol of one sequence corresponds to either a symbol of another sequence or a symbol of the gap. *Semi-global alignment* of sequence Q to sequence T is a global alignment of sequence Q to an arbitrary substring of T.

Consider *scoring scheme* that sets a penalty for each operation to change the sequence Q, for example, such as insertion, deletion or replacement of a symbol. *Optimal global (semi-global) alignment* of Q to T is an alignment that reflects the lowest total penalty of converting Q to T (to substring of T). Frequently, the scoring scheme also sets a reward for matching symbols in the alignment of two sequences. The most common scoring scheme, called *edit distance*, assigns a penalty of 1 for insertion, deletion or replacement of a symbol. Operations as well as values of penalties in the scoring scheme can vary depending on the task.

Identity between two sequences is usually calculated as a difference between the query length (or alignment length) and the edit distance divided by the query length (or alignment length).

Alignment of two sequences is an essential problem in the analysis of genomic data. Straightforward alignment algorithms based on dynamic programming procedure [39] require a lot of computational time and memory resources to process a large number of long biological sequences. Practical solutions that perform fast and accurate alignment of short and long reads to assembly contigs (BWA-MEM [18], Minimap2 [19], Bowtie2 [40]) use a heuristic approach called *seed-and-extend* to speed up the alignment. The key idea of the seed-and-extend approach is to find exact k-mers, called *seeds*, shared between a query sequence and a target sequence

to approximate which part of the query is aligned to which part of the target, and then *extend* seed alignments to obtain a full alignment.

1.1.4 Sequence-to-graph alignment problem in assembly graphs

Most short-read assemblers [14; 41; 42] construct either string graphs or de Bruijn graphs during assembly and thus can provide not only a set of contigs, but also assembly graphs. Assembly graphs contain excessive number of connections between reads and some of these connections reflect true connections, lost during contig construction step.

In order to utilize this information, some problems can be solved at the level of assembly graphs rather than at the level of final contigs. The sequence-to-graph alignment problem is the basic problem that should be solved for this purpose [43–47]. Identifying alignments of long error-prone reads (such as PacBio and ONT reads) is particularly important in hybrid assembly [4; 6], read error correction [5], and haplotype separation [7]. Alignment of amino acid sequences to assembly graphs is useful for analysis of complex metagenomic datasets.

Current solutions to the sequence-to-graph alignment problem utilize the seed-and-extend idea, but with some modifications. While the seeding step is similar to the seeding step in the linear case, the seed extension step faces new challenges, such as multiple potential extensions of seeds due to the non-linear graph structure and graph loops.

Such assemblers as hybridSPAdes [4] and Unicycler [6] perform long reads alignment to short-read assembly graph as one of the steps of their pipeline to improve assembly graph and resolve complex parts of the assembly graph.

In addition, there are instruments specifically designed to solve the sequence-to-graph alignment problem. The vg alignment pipeline [43] was primarily developed to align reads to variation graphs (sequence graphs to compare variations across individuals) rather than assembly graphs. In the first step, it searches for super maximal equal matches (SMEMs) using the GCSA2 library [48], then filters and chains the SMEMs. Next vg "unrolls" cycles in the graph, thus transforming it into a directed acyclic graph (DAG) and uses SIMD-accelerated banded dynamic programming to link SMEMs.

GraphAligner [49] has become a widely-used tool for aligning long read sequences onto general sequence graphs. GraphAligner $v1.0.4$ (tested in the original paper [15]) used MUMmer4 [50] to identify potential alignment seeds, while the most recent version of GraphAligner $v1.0.16$ performs a minimizer-based seeding procedure, which is faster but less accurate. After that, seeds are extended separately with a Myers' bit-vector algorithm, specifically improved for sequence-to-graph alignment [51].

In August 2022, the prerelease version of sequence-to-graph aligner within Minigraph toolkit [52] was announced. The Minigraph aligner uses ideas and implementation from Minimap2 [19]. However, for now Minigraph aligner works only with sequence graph where overlaps between two edges (or k value in de Bruijn graphs) are equal to 0, so it can not be used on assembly graphs without additional graph transformations.

To the best of our knowledge, no tool supports alignment of amino acid sequences. PathRacer [20] and MegaGTA [53] support alignment of Hidden Markov Models (HMMs) to assembly graph, which theoretically can be considered as a generalization of sequence-to-graph alignment. However, these tools are more time consuming due to assumption that they perform HMM alignment (basically, aligning a directed acyclic graph (DAG) to another graph).

The result presented in this chapter is a solution to the sequence-to-graph alignment problem for both nucleotide and amino acid sequences: we first improved reads-to-graph aligner module of hybridSPAdes assembler [4] and then proposed a new algorithm and a tool SPAligner. SPAligner is based on algorithms and implementation of hybridSPAdes alignment module.

In the original study [15] SPAligner was benchmarked against vg [43] and GraphAligner [49]. In this chapter we decided to omit benchmarking against the vg toolkit sequence-to-graph aligner since it was generally tuned for short-read alignment to variation graphs and has very slow runtime for long reads alignment to assembly graphs [15; 49].

1.2 Alignment of long reads to assembly graphs

1.2.1 Algorithm overview

Consider a nucleotide sequence S and an assembly graph G. In order to omit technical details we will consider assembly graph G as directed graph with nucleotide sequences (labels) on its edges and with zero overlaps between adjacent edges (i. e. vertices' labels have zero size). Path P in the assembly graph G is described by a list of edges $e_1, e_2, \ldots, e_n$, where each edge is an incoming edge for a start vertex of the next edge. So, if $Label(e_1), Label(e_2), \ldots, Label(e_n)$ are labels of edges in the path P, then label $Label(P)$ of path P is a concatenation of strings $Label(e_1), Label(e_2), \ldots, Label(e_n)$. *The path of optimal semi-global alignment* between a sequence S (query) and an assembly graph G is a path P in G that results in the lowest possible semi-global alignment cost between the sequence S and the label of P, compared to all other paths in G [15; 45; 49].

SPAligner applies a heuristic approach to semi-global sequence-to-graph alignment problem [15]. Initially, it builds an index of all graph edges' labels using BWA-MEM library [18]. For each sequence S it starts by searching for *anchors*, i.e. regions of high similarity between S and edges of G. We decided to use anchors, local alignments found by BWA-MEM, instead of exact seeds, in this work to utilize BWA-MEM ability to filter and extend seeds into reliable local alignments in case of linear sequences. After that SPAligner filters these anchors and tries to find best alignment of S that goes through final anchors. The pipeline for the alignment of S to assembly graph G consists of four main steps [15] described below and shown in Figure 1.1.

Anchor search. High-scored local alignments between sequence S and edges of G reported by BWA-MEM are considered as initial anchors. Each anchor a is described by its alignment position on S, alignment edge, and an alignment position on that edge.

Anchor filtering. SPAligner filters anchors by their size, their position on edge, and their mutual position on the sequence S.

Anchor chaining. In an assembly graph G, a pair of anchors is considered *compatible* if the difference between their minimal distance in G is

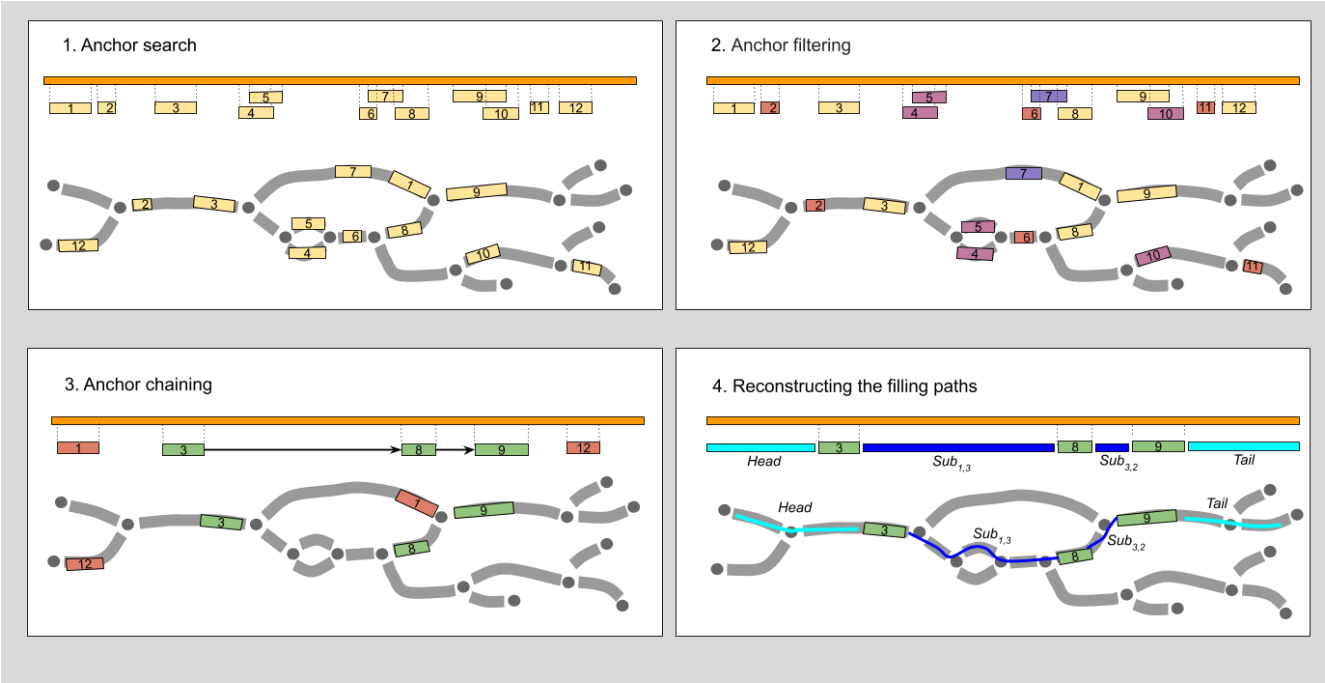

Figure 1.1 — SPAligner pipeline for nucleotide sequence alignment to assembly graph. The process of aligning the query sequence S (represented as an orange bar) to the assembly graph G (represented by grey bars, directed from left to right) involves four steps. **Upper-left: Anchor search.** Anchor alignments are identified between S and the labels of the edges in G using BWA-MEM. **Upper-right: Anchor filtering.** The identified anchors are filtered based on certain criteria such as their length being shorter than K(anchors 2, 6, and 11), or if they are located in the middle of a long edge (anchor 7), or if they are mostly covered by other anchors (anchors 4, 5, and 10). **Bottom-left: Anchor chaining.** The heaviest chain of compatible anchors is then identified using dynamic programming procedure. **Bottom-right: Reconstructing the filling paths.** The filling paths are reconstructed by aligning the substrings of S between consecutive chain anchors, as well as the prefix and suffix substrings.

not $min\_stretching$ ($min\_stretching = 1.5$ by default) times large than their distance on the sequence S. Anchors are assigned weights equal to their lengths in the query sequence S. SPAligner searches for the heaviest chain of compatible anchors using dynamic programming and considers the resulting chain as a skeleton of the final alignment [4].

Reconstructing the filling paths. At this stage SPAligner tries to connect pairs of neighbouring anchors in the skeleton by finding path P_{Sub} with such $Label(P_{Sub})$ that have minimal alignment cost to corresponding region in the query sequence. First, SPAligner tries to find best-scoring path using brute-force approach implemented in initial hybridSPAdes assembler [4], i.e. SPAligner generates upto X paths between two neighbouring anchors and finds the path with the best score. If there are more than X paths between two anchors, then SPAligner tries to

find the best alignment using dynamic programming approach. We reformulated the alignment problem as a minimal-weight path problem in the corresponding extended alignment graph [51; 54] following classical approach used in previous studies [4; 51; 54; 55] (see the next section for more precise alignment graph description). There may be situations where SPAligner is unable to connect adjacent anchors, resulting in separate alignments that cover different segments of the query sequence without overlap. Additionally, SPAligner tries to extend the alignment beyond the first and last anchors (i.e., prefix and suffix) to produce a complete alignment of the query sequence.

1.2.2 Sequence to graph alignment via alignment graphs

Consider a query subsequence Sub, an assembly graph G, and a linear gap penalty scoring scheme (i.e. gap penalty $\sigma > 0$ and mismatch penalty $\mu > 0$). Position pos_G в G is described by a pair $< e, pos_e >$, where e is an edge id and pos_e is a position on the edge e. We define edges of the alignment graph $SG(G,Sub)$ with vertices corresponding to all pairs of a position in G and a position in Sub as follows:

1. $< pos_G, pos_{Sub} > \rightarrow < pos'_G, pos_{Sub} >$ of weight σ,
2. $< pos_G, pos_{Sub} > \rightarrow < pos_G, pos_{Sub} + 1 >$ of weight σ,
3. $< pos_G, pos_{Sub} > \rightarrow < pos'_G, pos_{Sub} + 1 >$, of weight 0 if $Sub[pos_{Sub} + 1]$ matches $G[pos'_G]$, and μ otherwise,
4. $< posEnd_G, pos_{Sub} > \rightarrow < posStart_G, pos_{Sub} >$ of weight 0 if $posEnd_G$ and $posStart_G$ are located on the same vertex in G (represents connection between adjacent edges),

where variable pos_G runs through every position in the graph G, while pos'_G runs through all graph positions that extend pos_G, and pos_{Sub} runs through all possible positions in the subsequence Sub.

In order to find minimal alignment path P_{Sub} between two anchors a_i and a_{i+1} in the skeleton chain with graph coordinates pos_{Gi} and pos_{Gi+1} respectively, one need to find minimal-weighted path connecting vertices $< pos_{Gi}, 0 >$ and $< pos_{Gi+1}, |Sub| >$ in $SG(G, Sub)$ according to Lemma 5 in [55]. As edges $SG(G, Sub)$ have non-negative weights, this path can be found by running Dijkstra algorithm [4;

55] in $SG(G, Sub)$. In addition, the method being suggested can accommodate an affine gap scoring system (i.e. gap openning penalty $\sigma > 0$, gap extension penalty $\gamma > 0$, and mismatch penalty $\mu > 0$) [56].

Time complexity of Dijkstra algorithm in alignment graph $SG(G,S)$ that searches for optimal alignment between the query S and assembly graph G is $O(|G| \cdot |S| \cdot \log(|G| \cdot |S|))$, , where $|G|$ is a total length of all edges in G, and $|S|$ is a length of S. At the same time, Navarro et al. [57] and Rautiainen et al. [51] desribed algorithms with the time complexity of $O(|G| \cdot |S|)$ for classical edit distance scoring scheme, where μ and σ equal to 1.

Notable, this time complexity can be achieved by a simple modification of Dijkstra algorithm, which can be implemented more straightforwardly than algorithms from [51; 57]. Within classical edit distance scoring scheme the edges of $SG(G,S)$ have weights of either 0 or 1. It is not difficult to show that in this case the priority queue of Dijkstra algorithm contains only elements that have no more than two distinct values, that can only differ by 1, at any moment of the algorithm execution. Thus the priority queue can be replaced by a deque: all the vertices across 0-edges (1-edges) are appended along with the corresponding distance to the beginning (to the end) of the deque. Deque implementation provides all necessary operations in $O(1)$ time, improving the overall running time to $O(|G| \cdot |S|)$.

Recently in [45] authors suggested a similar extension with $O(|G| \cdot |S|)$ time complexity for affine gap penalty scoring scheme.

In this work we decided to take advantage of the existing highly optimized solutions for aligning sequences to each other [58]. In the current implementation SPAligner tries to align suffix of Sub to full edge with the Edlib library [58] omitting positions within the edge while searching for the shortest path using Dijkstra algorithm.

1.3 Alignment of amino acid sequences to assembly graphs

Consider a protein sequence S_p and an assembly graph G with nucleotide sequences on edges and zero length overlaps between adjacent edges. A path P in

G with label $Label(P)$ is considered as *valid* if $Label(P)$ can be represented as a protein sequence $Translated(Label(P))$, i.e. it can be translated into valid protein sequence without stop codons inside. The path of the optimal semi-global alignment between sequence S_p and the assembly graph G is considered as a valid path P in G, where the cost of aligning S_p to $Translated(Label(P))$ is the lowest compared to all valid paths in G.

1.3.1 Amino acid sequence alignment pipeline

The SPAligner protein sequence alignment procedure is quite similar to its nucleotide sequence alignment procedure with some differences highlighted below (Figure 1.2).

Anchors search. First, SPAligner searches for anchor alignments between nucleotide representation of S_p and the pre-indexed six-frame translated edge labels. To achieve this, each amino acid in S_p is substituted by the lexicographically minimal codon, as is each triplet in the edge label.

Anchor filtering. The filtering procedure performed on this step is similar to nucleotide sequence anchors filtering. There is no chaining step in protein sequence alignment procedure as protein sequences tend to be shorter and usually can be reconstructed from one anchor.

Alignment extension. SPAligner tries to extend each resulting anchor alignment through the assembly graph to obtain the path that has the best alignment to sequence S_p (see "Alignment graph for amino acid sequence alignment").

Alignments post-processing. For the protein sequence S_p SPAligner can find more than one optimal path through the graph G as it extends each anchor separately. The final output contains all unique paths that longer than a certain fraction of S_p length (0.8 by default) rescored against S_p with Parasail library [59] to obtain alignment weight according to amino acid substitution matrices (BLOSUM or PAM).

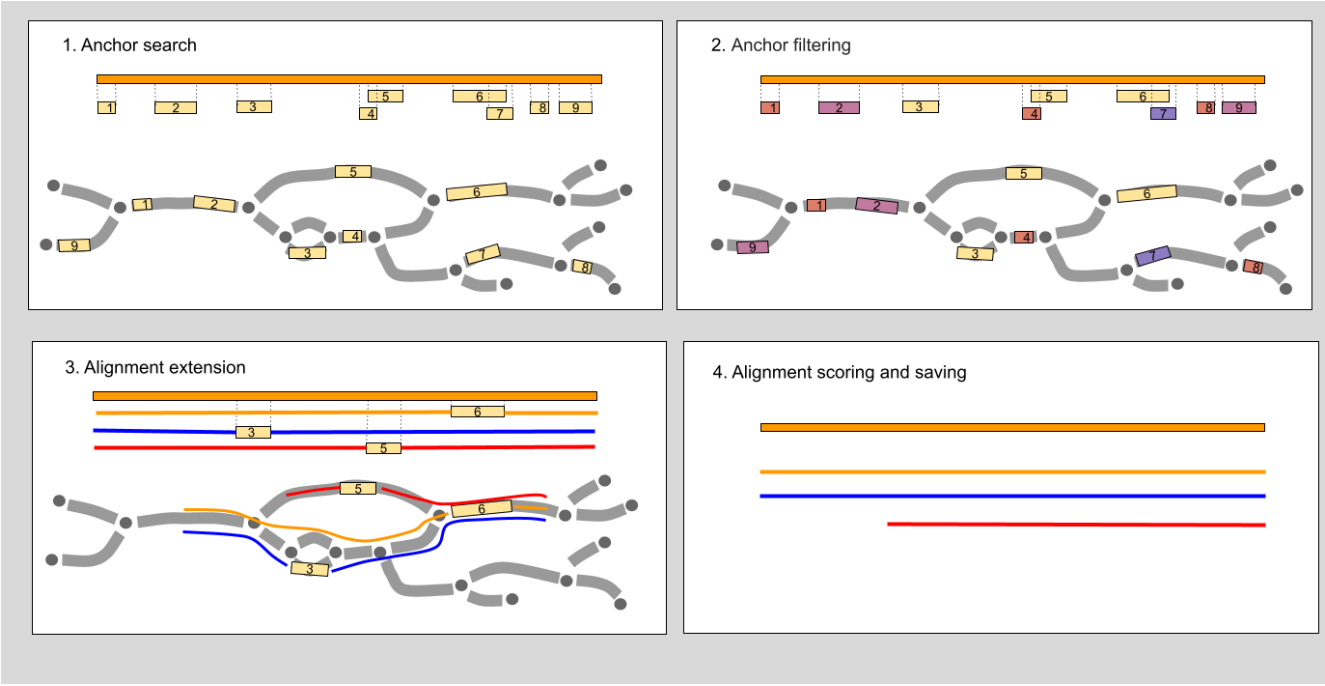

Figure 1.2 — SPAligner pipeline for amino acid sequence alignment to assembly graph. The process of aligning the query sequence S_p (represented as an orange bar) to the assembly graph G (represented by grey bars, directed from left to right) involves four steps. **Upper-left: Anchor search.** Anchor alignments are identified between S_p and pre-indexed six-frame translated labels of the edges in G using BWA-MEM. **Upper-right: Anchor filtering.** The identified anchors are filtered based on certain criteria such as their length being shorter than K (anchors 1, 4, 8), or if they are located in the middle of a long edge (anchors 2, 9), or if they are mostly covered by other anchors (anchor 7). **Bottom-left: Anchor extension.** Each anchor is extended to obtain the best scoring alignment. **Bottom-right: Alignment scoring and saving.** The resulting alignments are filtered based on their length (red alignment is discarded) and are rescored against S_p with Parasail library [59].

1.3.2 Amino acid sequence alignment scoring scheme

To align long reads SPAligner uses conventional scoring scheme for nucleotide sequence alignment with fixed mismatch penalties, while aligning protein sequences SPAligner extends anchors using specialized amino acid substitution matrices, i.e. BLOSUM or PAM matrices [60]. Scores in such matrices reflect relative frequencies of amino acids and substitution rates between different amino acids.

The standard substitution matrix M assumes that a higher cumulative score indicates a better alignment. To employ M in SPAligner's cost-minimization approach to the alignment problem, we treat the elements of the matrix $M' = -M$

as a match/mismatch score (with $M = BLOSUM90$). In addition, we use linear gap penalties with a coefficient of σ (with $\sigma = 5$ being used in our experiments).

1.3.3 Alignment graph for amino acid sequence alignment

To align a query subsequence Sub_p to an assembly graph G using an amino acid substitution matrix and gap penalty coefficient σ, we create an alignment graph SG_p. Each vertex in SG_p is represented by a triplet $< pos_G, pos_{Sub_p}, fs >$, where pos_G is a position in the graph G, pos_{Sub_p} is a position in Sub_p, and fs is a current codon substring of length 0-2 (with an empty string represented by ε). We construct SG_p vertices for all possible triplets as described above, and then connect them with edges using the following rules:

1. $< pos_G, pos_{Sub_p}, fs > \rightarrow < pos'_G, pos_{Sub_p}, fs + G[pos'_G] >$ of 0 weight if $|fs| < 2$,
2. $< pos_G, pos_{Sub_p}, fs > \rightarrow < pos'_G, pos_{Sub_p}, \varepsilon >$ of weight σ if $|fs| = 2$,
3. $< pos_G, pos_{Sub_p}, fs > \rightarrow < pos'_G, pos_{Sub_p} + 1, \varepsilon >$ of weight $M'[Translate(fs + G[pos'_G]), Sub_p[pos_{Sub_p} + 1]]$, if $|fs| = 2$,
4. $< pos_G, pos_{Sub_p}, \varepsilon > \rightarrow < p, pos_{Sub_p} + 1, \varepsilon >$ of weight σ,
5. $< posEnd_G, pos_{Sub_p}, fs > \rightarrow < posStart_G, pos_{Sub_p}, fs >$ of weight 0, if $posEnd_G$ and $posStart_G$ both correspond to the same vertex in G (represent a connection between adjacent edges),

where pos_G is a position in G, pos'_G is a position in G extending pos_G.

The alignment path for a fixed subsequence Sub_p, flanked by anchor alignments, can be obtained from the minimal weight path in the alignment graph SG_p. However, the Dijkstra algorithm cannot be applied directly to SG_p due to the presence of negative weights in matrix M' and, consequently, negative edges in SG_p. To address this issue, we add a constant C (greater than the maximum value in M) to the weights of edges in rules 3 and 4 and apply Dijkstra algorithm to the resulting graph SG'_p, which has only non-negative weights.

The minimum weight path between any two nodes $s =< x, 0, \varepsilon >$ and $e =< y, |Sub_p|+1, \varepsilon >$ in SG'_p is also the minimum weight path between s and e in SG_p. If we take an arbitrary path P in SG_p between s and e with weight w_P, its weight after transformation (in SG'_p) is $w_p + C \cdot |Sub_p|$, as P contains exactly $|Sub_p|$ edges with

modified weights. Thus, all paths between s and e will have their weights increased by the same constant $|Sub_p| \cdot C$, and we can obtain the semi-global alignment path of Sub_p in SG_p by running the Dijkstra algorithm on the modified graph.

1.4 Results

1.4.1 Long reads to graph alignment benchmarking

We benchmarked SPAligner against recent version of the widely used GraphAligner tool $v1.0.16$ [49]. As in the original study [15], we evaluated GraphAligner and SPAligner on PacBio and ONT reads from three different organisms: *E. coli* strain K12, *S. cerevisiae* strain S288C, and *C. elegans* strain Bristol N2 1. Assembly graphs were build with SPAdes $v3.12$ [14] assembler.

SPAligner was benchmarked in two modes: the basic mode (SPAligner), that was fully described above; the fast mode (SPAligner w/o AG), where SPAligner tries to close gaps using only brute-force approach implemented in hybridSPAdes [4] without running Dijkstra algorithm on alignment graph (parameters $run\_dijkstra = false$, $restore\_ends = false$). GraphAligner was also benchmarked in two modes: the initial one (GraphAligner); and more precise mode (GraphAligner-MUM), where we used MUMers instead of minimizers and applied wide bands on dynamic programming restrictions ("`-seeds-mum-count -1 -C 500000 -max-cluster-extend 5 -b 35`").

	E. coli K12	*S. cerevisiae* S288C	*C. elegans* Bristol N2
Reference ID	U00096.2	GCA_000146045.2	GCA_000002985.3
Illumina	ERA000206	ERP016443	-
PacBio	PacBio DevNet	ERR1655118	PacBio DevNet
ONT	Loman Lab R9 data	ERP016443	PRJEB22098

Table 1 — Datasets for benchmarking GraphAligner and SPAligner. Due to the lack of freely available up-to-date ONT data for strain Bristol N2, reads sequenced from closely-related wild-type *C. elegans* strain VC2010 [61] were used (see original study for discussion of their relatedness).

Tools were compared based on the following metrics:

– *Number of mapped reads* (MR): We consider a read to be mapped if 80% of its length is covered by a single alignment path.
– *Alignment identity (%)*(AI): Mean nucleotide identity across the longest continuous alignments of the mapped reads.
– *Execution time (h:m:s)*(T): Wall-clock time with 16 threads.
– *Memory (Gb)*(M): Peak RAM usage with 16 threads.

Results of comparison are presented in Table 2 for PacBio reads and Table 3 for ONT reads.

	Aligner	MR, %	AI,%	T, h:m:s	M, Gb
E. coli	GA v1.0.16	81.94	87	0:00:08	0.3
	GA-MUM v1.0.16	83.5	87	0:00:31	0.2
	SPAligner w/o AG	82.19	87	0:02:47	0.3
	SPAligner	88.27	86	0:03:23	0.3
S. cerevisiae	GA v1.0.16	50.37	87	0:00:13	0.4
	GA-MUM v1.0.16	51.36	87	0:00:44	0.3
	SPAligner w/o AG	49	86	0:02:15	0.4
	SPAligner	56.51	85	0:48:49	0.6
C. elegans	GA v1.0.16	79.45	87	0:00:49	1.5
	GA-MUM v1.0.16	82.12	87	0:02:41	1.2
	SPAligner w/o AG	74.01	87	0:06:27	0.6
	SPAligner	84.43	86	0:43:57	1.0

Table 2 — The performance of SPAligner and GraphAligner on aligning PacBio reads to short-read assembly graphs, constructed by SPAdes with k-mer size of 77, was evaluated using three datasets. Each dataset comprised 10 000 reads longer than 2Kbp. All the runs were executed using 16 threads. To overcome the high fragmentation of the assembly graph resulting from the available *C. elegans* Bristol N2 Illumina sequencing data, simulated Illumina reads were used to obtain the *C. elegans* assembly graph. The alignment quality was measured using four metrics: the number of mapped reads (MR), the alignment identity (%)(AI), the execution time (h:m:s)(T), and the peak RAM usage (Gb)(M).

All four tools' configurations produced similar results on the average identity and number of mapped reads (see Tables 2, 3). SPAligner tends to map more reads than other configurations without large average identity drawdown. The analysis of reads mapped by more than 80% of their length only by SPAligner showed that it produces longer paths as it is able to restore an alignment in parts where

	Aligner	MR, %	AI,%	T, h:m:s	M, Gb
E. coli	GA *v*1.0.16	61.19	87	0:00:10	0.5
	GA-MUM *v*1.0.16	63.25	87	0:00:36	0.4
	SPAligner w/o AG	65.3	86	0:02:20	0.4
	SPAligner	67.65	86	0:02:33	0.4
S. cerevisiae	GA *v*1.0.16	57.4	83	0:00:16	0.4
	GA-MUM *v*1.0.16	58.37	83	0:00:37	0.3
	SPAligner w/o AG	55.43	83	0:00:55	0.3
	SPAligner	61.55	82	0:20:47	0.5
C. elegans	GA *v*1.0.16	60.09	86	0:00:57	1.4
	GA-MUM *v*1.0.16	64.06	86	0:02:53	1.2
	SPAligner w/o AG	60.01	87	0:05:56	0.8
	SPAligner	65.82	86	0:39:09	1.1

Table 3 — The performance of SPAligner and GraphAligner on aligning ONT reads to short-read assembly graphs, constructed by SPAdes with k-mer size of 77, was evaluated using three datasets. Each dataset comprised 10 000 reads longer than 2 Kbp, except for the *E. coli* ONT dataset, which contained approximately 7 000 reads of appropriate length. All the runs were executed using 16 threads. To overcome the high fragmentation of the assembly graph resulting from the available *C. elegans* Bristol N2 Illumina sequencing data, simulated Illumina reads were used to obtain the *C. elegans* assembly graph. The alignment quality was measured using four metrics: the number of mapped reads (MR), the alignment identity (%)(AI), the execution time (h:m:s)(T), and the peak RAM usage (Gb)(M).

reads and graph edges have lower similarity. Consequently, SPAligner is much more time-consuming than other configurations. At the same time, SPAligner w/o AG mode has similar time execution to GA and GA-MUM. Thus SPAligner approach is competable with GraphAligner, the state-of-art solution for sequence-to-graph alignment.

1.4.2 Amino acid alignment benchmarking

Metagenomic assemblies tend to be highly fragmented due to within-species and inter-species repeats caused by horizontally transferred genes, conservative regions, or closely-related strains [62]. Contig fragmentation can result in the

fragmentation of gene-coding segments within a genome assembly, which can significantly impact downstream functional analysis of the microbial community [63;64]. SPAligner sets itself apart from other sequence-to-graph aligners by allowing for the search of amino acid sequences in assembly graphs, making it possible to identify fragmented genes.

In this section, we demonstrate the application of SPAligner in the search for known antibiotic resistance genes in a wastewater metagenomic dataset. In the experiment, only the best scoring amino acid alignment among all produced alignments for a particular query is considered.

Beta-lactamases represent enzymes produced by various bacteria that provide resistance against widely used beta-lactam antibiotics. Identification of beta-lactamases' genes in environment samples is essentially important for tracking antibiotic resistance spread. In [65] authors searched for antimicrobial resistance (AMR) proteins, including beta-lactamases, in wastewater and urban surface water metagenomic datasets. The hospital wastewater discharge dataset has the highest number of beta-lactamase genes among all datasets presented in the study according to the original paper. The dataset (SRA acc. no. SRR5997548) consisting of 3.3 million 2×250 bp Illumina reads (mean insert size of 350 bp) was assembled by metaSPAdes $v3.12$ with default settings.

In order to show that SPAligner is able to find more protein sequences than the conventional approach, we analyze AMR genes found in assembled contigs by the state-of-the-art tool AMRFinder [66] and AMR genes found by SPAligner in the corresponding metagenomic assembly graph. Both AMRFinder and SPAligner aligned 4810 protein sequences from Bacterial Antimicrobial Resistance Reference Gene Database [66]: partial predictions that cover less than 75% of the query sequence length were discarded and only alignments with more than 90% identity were considered. Final sequences found by AMRFinder and SPAligner were clustered using single-linkage clustering with a 90% similarity threshold. Out of the resulting 89 clusters one contains only sequences found by AMRFinder, and four clusters contain only sequences provided by SPAligner. Sequences from the single cluster with AMRFinder alignments represent border case and span 90.7% of the query length, while SPAligner found a little bit shorter alignment that span 89.7% of the query length and they were filtered out. We searched for representatives of four clusters found by SPAligner in contigs using tBLASTx [67]. All of them were only partially ($< 75\%$ of the query length) covered by contigs or had low identity ($< 70\%$).

At the same time SPAligner reconstructed sequences of seven (bla_{KPC}, bla_{CTX-M}, bla_{SHV}, bla_{TEM}, bla_{IMP}, bla_{VIM} and bla_{OXA}) out of nine beta-lactamase protein families from the original study. Two remaining families were poorly covered ($\sim$ 20X) compared to other families ($\sim$ 150X) [65]. Figure 1.3 shows the alignment of beta-lactamase IMP-4 scattered over several contigs.

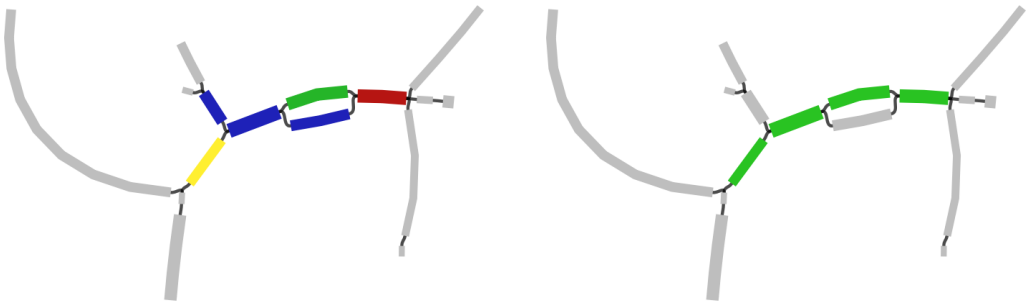

Figure 1.3 — Metagenomic assembly subgraph with colored contigs (Left) and a reconstructed fragmented beta-lactamase protein sequence (Right). (Left) Assembly subgraph visualized by Bandage software [68]. The different colors (blue, red, green, and brown) represent four different contigs of the metaSPAdes assembly. (Right) The alignment path (green edges) for the IMP-4 gene, which is 246 amino acids long and has an alignment identity of 98%.

1.5 Conclusions

In this chapter we demonstrated that SPAligner is an effective instrument to align both long error-prone reads and amino acid sequences to short-read assembly graphs. The work on SPAligner produced algorithms and approaches that can help to accelerate development of sequence-to-graph alignment algorithms. At the same time, it worth to note that its implementation is highly dependent on SPAdes codebase and its restrictions (e.g., SPAligner can be applied only to de Bruijn graphs).

SPAligner tool is competitive to other long read-to-graph aligners. In addition, SPAligner is the only tool that can be used to successfully recover full protein sequences directly from metagenomic assembly graphs. While the extraction of all possible protein sequence alignments from assembly graphs represents only the initial step of the coding sequence identification, ORFograph pipeline [16] presented in the next chapter uses information from contigs and graph structure to get most reliable protein sequence candidates.

Chapter 2. Search for insecticidal protein genes in assembly graph

2.1 Introduction

2.1.1 *Bacillus thuringiensis* insecticidal proteins

Biopesticides are usually defined as types of pesticides derived from living organisms or natural products [69]. Their main advantages over synthetic chemical pesticides include:

- high specificity (i.e., affects only target and closely related organisms);
- effectiveness in small doses (i.e., have no negative environmental impact because they dissolve quickly);
- less toxicity.

Plant-incorporated pesticides [69] are a type of biopesticide that is widely used in agriculture. Usually they represent pesticidal proteins from genes that have been incorporated to the DNA of a plant. *Bacillus thuringiensis* (Bt) bacteria has gained significance as a valuable source of pesticidal proteins. It has been isolated from diverse ecosystems, such as soil, water, dead insects, and human tissues. During its sporulation and stationary growth phases, Bt produces crystal (Cry) and cytotoxic (Cyt) insecticidal proteins, while vegetative insecticidal proteins (Vip) are produced during the vegetative growth phase. These proteins have been shown to be effective against various insects belonging to the orders Lepidoptera, Coleoptera, Hemiptera, and Diptera [70], and are widely used as a crucial element in insect pest management strategies [71]. Some of them have also been shown in recent studies to be active against human cancer cells [72].

Bt resistance in insects evolved as a result of the active use of Bt insecticidal proteins [73]. Over the last thirty years, the number of known Cry-encoding genes has increased from 14 to over 700 [74]. The constant need for new insecticidal protein genes (IPGs) requires new methods of IPGs identification.

2.1.2 Insecticidal protein genes search

Initially, detection of Cry, Cyt, and Vip genes was performed using PCR-based methods [75;76]. E-PCR [77], PCR-RFLP [76] and PCR-SSCP [78] systems enabled the identification of potentially new IPGs. Their success was mainly based on a primer selection, and their attempts to identify new IPGs were mostly experimental and limited to the identification of three-domain Cry genes [79].

With development of sequencing technologies new cheaper computational methods for IPGs detection has become available. Gene prediction tools GeneMark [80], Prodigal [81], Glimmer [82] and their metagenomic versions metaGeneMark [83], metaProdigal [84], metaGlimmer [85] can perform gene search in reads and contigs. In order to find potential IPGs, the resulting gene candidates can be compared against databases of already known IPGs. In addition, tools that perform specific Bt toxin search in reads and contigs emerged, such as BtToxin_scanner and BtToxin_Digger [86]. They receive genomic sequences and domain HMMs as input and predict Cry, Cyt, or Vip gene sequences using machine learning techniques. All of these methods, however, can only reconstruct genes that reside within a single contig.

This chapter introduces ORFograph, a pipeline designed to extract open reading frames (ORFs) from assembly graphs. By utilizing ORFograph, candidate IPGs can be identified even if their parts are located on several contigs, providing more potential IPGs for subsequent processing steps in the whole IPG identification pipeline, which provides a significant advantage over previous approaches.

ORFograph is a tool that takes as input de Bruijn assembly graph and a set of domain HMMs or protein sequences, searches for similar ORFs in the assembly graph, and performs initial candidate filtering. Thus, it can identify not only IPGs, but any proteins similar to those provided as input. In this work we concentrate on Cry, Cyt, and Vip toxins because their identification is practically important and because their sequences tend to be longer in comparison to sequences of other proteins (Figure 2.1). So, their sequences are usually fragmented and located on several contigs.

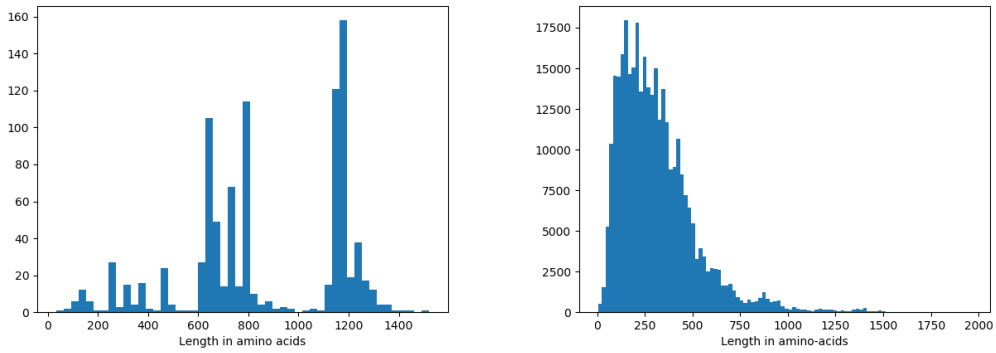

Figure 2.1 — Distribution of the lengths for 941 known Cry and Vip insecticide proteins (Left), and bacterial genes from the Uniprot database [87] (Right). The Cry and Vip insecticide proteins are characterized by a substantial proportion (84%) of genes that have a length of over 600 amino acids. Conversely, only a small proportion (7%) of all genes from the Uniprot database have a length exceeding 600 amino acids.

2.2 ORFograph pipeline overview

ORFograph accepts as input a set of domain HMMs, a set of protein sequences, and an assembly graph built by one of the modern genome assemblers [38; 41; 42] that construct de Bruijn graphs from a set of short reads. It outputs ORFs found in the assembly graph that contain either domains, described by input HMMs, or ORFs similar to input protein sequences. ORFograph clusters these sequences and tries to find the most reliable representative for each cluster. ORFograph pipeline can be divided into four steps outlined below (Figure 2.2).

Aligning known insecticide proteins/HMMs to the assembly graph. ORFograph uses PathRacer [20] to align HMMs to the assembly graph and keeps all alignments with an e-value less than 10^{-9} and a length greater than 90% of the HMM length. Input protein sequences are aligned to the assembly graph using SPAligner [15]. These alignments are further considered as partial ORFs, or *anchor paths*, that have to be extended to the full ORFs. These alignments can be represented as a part of an edge, or as a path that goes through one or more edges in the assembly graph.

Start and stop codon search. ORFograh then attempts to extend each anchor path to the full ORF. It takes the anchor path end position in the

assembly graph and explores all paths to the nearest stop codon in it. After that, ORFograph performs a similar procedure to find potential start codons by exploring all "backward" paths from its start position to the previously appearing stop codons and identifies putative start codons between them.

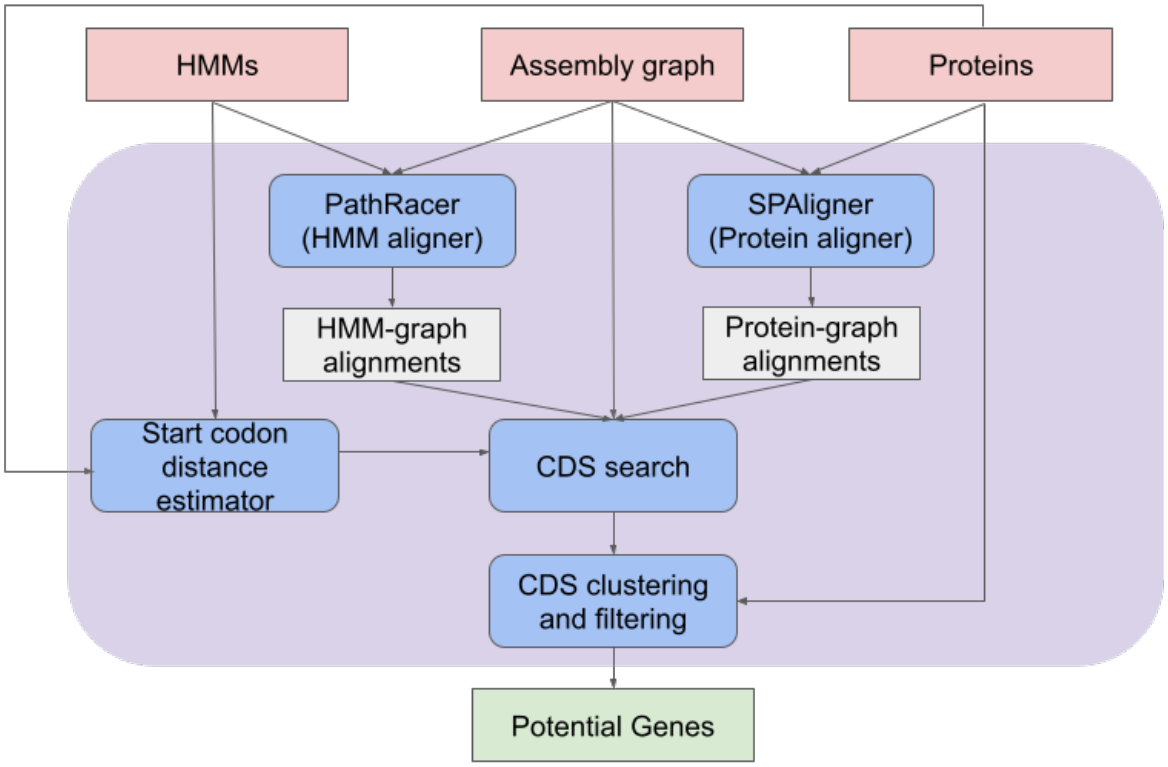

Figure 2.2 — The ORFograph pipeline. ORFograph accepts de Bruijn assembly graph, HMMs and protein sequences as input. On the first step, it builds alignments of HMMs and proteins to the assembly graph using PathRacer and SPAligner, respectively. After that ORFograph performs initial ORF search and filtering using anchor paths generated on the previous step. Finally, ORFograph performs clustering and filtering of the resulted ORF sequences.

Generation of complete coding sequences (CDSs). For each identified pair of start and stop codons for a specific anchor path, ORFograph runs a exhaustive search to generate all prefix paths leading from the start codon to the beginning of the anchor path and all suffix paths leading from the end of the anchor path to the stop codon. A limit of generating a maximum of 1000 paths is imposed for running time speed-up. Each pair of prefix and suffix paths are concatenated with the anchor path resulting in potential gene sequence. Resulting potential gene sequence are filtered based on information about contig paths. The selected paths are then translated into amino acid sequences, and identical sequences are removed.

Clustering potential proteins and selecting representatives. Since the number of potential protein sequences can be very large, ORFograph clusters the resulting sequences and selects a set of representatives in each cluster based on their mutual position in the assembly graph.

In the next four sections, these four steps are described in more detail.

2.3 Aligning known IPGs and domain HMMs to the assembly graph

ORFograph constructs anchor paths by performing the HMM-to-graph alignment [20] or sequence-to-graph alignment [15] to detect anchor paths and further extends each anchor path into a complete ORF. With HMM alignment, ORFograph often finds a large number of short paths, since gene-related HMMs usually represent a short domain rather than a full gene sequence. ORFograph removes such short alignment-paths from further consideration if they represent subpaths of longer (and thus more informative) alignment-paths with the same frameshift.

2.4 Start and stop codons search

For each anchor path, ORFograph runs a breadth-first search (BFS) [88] on the assembly graph from the start (end) position of the anchor path to identify all potential start (stop) codons. During this search, each new visited vertex in the BFS graph is described not only by its position in assembly graph, but also by the frameshift (of length 1, 2, or 3) at which it was reached, thus storing the last passed codon. The BFS algorithm does not continue its search from vertices for which the last codon traversed is a stop codon.

2.5 Generation of complete CDSs

Once the start and stop codon sets are calculated for each partial ORF alignment, the ORFograph generates the corresponding CDSs. The path of each CDS is divided into three parts: prefix, middle, and suffix, which correspond to the path from the selected start codon to the leftmost position of the anchor path, the anchor path and the path from the rightmost position of the anchor path to the selected stop codon, respectively. The ORFograph performs two separate exhaustively searches to generate all potential prefixes and suffixes and combines them with the middle part corresponding to the anchor path sequence to form a set of complete paths representing putative proteins. To speed up CDS generation, ORFograph first calculates a set of edges that can be traversed from the identified start codons to the beginning of the anchor path (and from the end of the anchor path to identified stop codons) using a depth-first search algorithm (DFS) [88]. ORFograph only iterates over these edges to create prefix and suffix paths. The ORFograph then combines each prefix, anchor path, and each suffix sequences to create a putative protein. As the number of potential full-length genes can be large, ORFograph filters and ranks the constructed sequences based on the criteria described below.

2.5.1 Analyzing HMM positions within gene sequences

At this step, ORFograph tries to estimate the most probable position of HMM relative to the beginning or end of the gene sequence using input sequences and HMMs. The HMMer [89] tool was used to obtain statistics on the position of input HMMs in potential genes. Using this tool for each HMM, the distribution of the distance from the beginning (end) of the gene sequence to the position of the HMM was calculated. ORFograph uses this distribution to estimate the most probable distance from the start (stop) codon to the HMM position in the sequences of a given protein family and determines the most probable prefixes (suffixes).

2.5.2 Filtering putative proteins that conflict with contigs

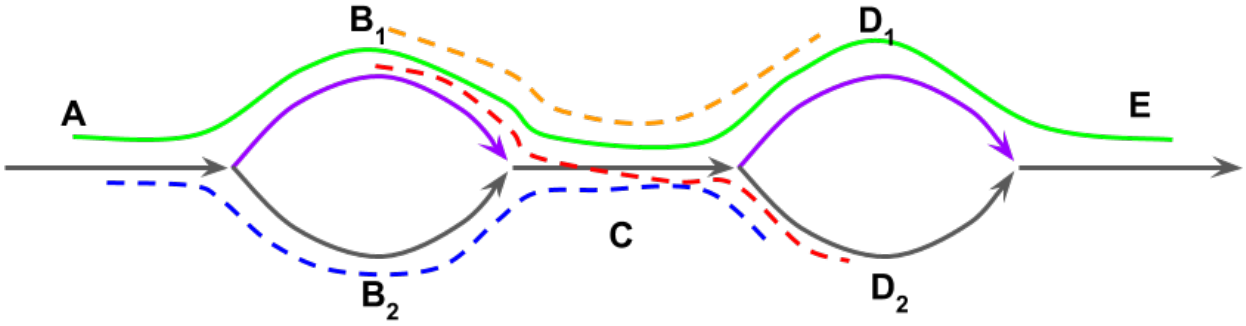

Figure 2.3 — Conflicting and non-conflicting paths in the assembly graph. There are two unique edges B_1 and D_1. All other edges are classified as non-unique edges. Edges B_2 and D_2 have length that is less than minLength threshold, while edges A, C, and E have inappropriate indegrees and/or outdegrees. The green path is considered as a contig paths, and red, blue, and yellow paths are as potential genes' paths. The green path contains both unique edges, while red path contains only one of them, so they conflict with each other. The green path doesn't intersect with yellow path by unique edges and contains the same unique edges as blue path, so there is no conflict in these cases.

ORFograph evaluates each putative protein path against the set of contigs and filters protein paths that conflict with the contig-set paths. ORFograph identifies *unique* edges with labels met only once in the genome sequence, using a straightforward set of heuristics. Specifically, an edge is considered unique if its length exceeds the threshold *minLength* (default is 300 nucleotides), its start vertex has an outdegree greater than 1, and its end vertex has an indegree greater than 1.

ORFograph considers the path of each putative protein and finds all contig paths that intersect this path by at least one unique edge. If there is a unique edge that exists only in the putative protein path, but not in the contig path, or, conversely, only in the contig path, but not in the path of the considered putative protein, then ORFograph filters out this putative protein, since it contradicts the contig path (see Figure 2.3).

2.6 Clustering potential proteins and selecting representatives

To further minimize the number of candidates, ORFograph applies single linkage clustering to all the sequences generated in the previous stage. ORFograph clusters two sequences together if one sequence is a substring of the other, or if their percent identity exceeds a predetermined threshold (default value is 90%). This step helps to simplify the analysis by grouping highly similar sequences into clusters.

ORFograph outputs a set of representatives for each cluster by following a set of rules. Firstly, it identifies all edges in the assembly graph that belong to at least one protein path in the cluster and have a length exceeding 1 kbp. Then, for each path in the cluster, ORFograph considers a triplet of information consisting of the start codon distance likelihood, the coverage by reads, and the path length. It sorts the triplets of different paths in a lexicographically increasing order, and considers a path P to be more reliable than another path P' if its triplet is lexicographically larger. ORFograph scans the protein sequences in the lexicographically non-decreasing order of their triplets and adds a potential protein to the list of representatives if it contains a long edge that was not presented in any of the previously considered sequences from the list.

2.7 Results

2.7.1 ORFograph benchmarking overview

We obtained 941 IPG sequences (811 Cry and Cyt toxins as well as 130 Vip toxins) from the NCBI database by their accession numbers listed in the Bt nomenclature list [90]. Additionally, we downloaded 66 HMMs representing domains commonly found in IPGs from the Pfam database [91], with the number of hidden states ranging from 25 to 350.

To evaluate the performance of ORFograph, we conducted benchmarking on simulated metagenomic datasets, real isolated Bt bacteria datasets, and a metagenomic dataset containing Bt samples. All datasets are publicly available

at https://figshare.com/s/f20604a5333bbe4514c9. We utilized SPAdes [38] or metaSPAdes [92] (for metagenomic dataset) to construct the assembly graph and then ran the ORFograph pipeline, running SPAligner on the 941 IPG sequences and PathRacer on the 66 downloaded HMMs.

It is known that Cry and Cyt genes usually reside on Bt bacteria plasmids. We tried to use plasmidSPAdes [93] to extract full plasmid sequences. We found that this method has limitations that prevent us from using it in our benchmarking [16].

CryProcessor [94] is a tool designed for Cry protein discovery that utilizes PathRacer [20] to extract Cry sequences from assembly graphs. However, it has limited functionality as it can only identify three-domain Cry toxins, and is not suitable for general gene discovery from assembly graphs. It is not possible to compare CryProcessor with ORFograph because CryProcessor only considers the sequences identified by PathRacer, without taking into account the ORF paths that contain those sequences.

2.7.2 Simulated datasets

In order to test ORFograph on simulated data, we used the CAMI challenge "TOY dataset" simulated from 30 genomes [95]. The dataset contains 74 million paired-end reads of length 100 bp (mean insert size 180 bp). The errors in reads were modeled after Illumina HiSeq reads.

We took two complete assemblies of the Bt genomes from the NCBI database: *BtA* (NCBI ID=GCF_002025105) and *BtB* (NCBI ID=GCF_001598095) and aligned them to 941 extracted IPG sequences. We excluded plasmids with toxin-like sequences to have control over the Cry gene sequences that may be identified and to vary the complexity of the datasets. We excluded the plasmid NZ_CP020005.1 from *BtA* and the plasmids NZ_CP014852.1, NZ_CP014853.1 from *BtB* that contain partial or full alignments of toxins from the IPG database, forming the simulated datasets BtA_{pure} and BtB_{pure}.

We varied the complexity of the IPG identification problem by enriching BtA_{pure} and BtB_{pure} with gradually increasing numbers of CDSs (0, 1, 2, 6) and with the gradually decreasing similarity between CDSs that varied from 45% to 92% (Tables 4 and 5).

Simulated Genome	Description
BtA_{pure}	BtA without plasmid NZ_CP020005.1
BtB_{pure}	BtB without plasmids NZ_CP014852.1, NZ_CP014853.1
BtA: Cry1Aa1	BtA_{pure} with added Cry1Aa1 CDS in NZ_CP020003.1 at position 27668
BtA: Cry1Aa1 + Cry1Ab1	*BtA*: Cry1Aa1 with added Cry1Ab1 CDS in NZ_CP020004.1 at position 24976
BtA: Cry1Aa1 + Cry1Ac1	*BtA*: Cry1Aa1 with added Cry1Ac1 CDS in NZ_CP020004.1 at position 12488
BtA: Cry1Aa1 + Cry1Ab1 + Cry1Ac1	*BtA*: Cry1Aa1 + Cry1Ab1 with added Cry1Ac1 CDS in NZ_CP020004.1 at position 12488
BtB: Cry1Ca1 + Cry2Aa1 + Cry2Ac1	BtB_{pure} with added Cry1Ca1 CDS in NZ_CP020003.1 at position 14716, Cry2Aa1 CDS in NZ_CP014848.1 at position 8614, and Cry2Ac1 CDS at NZ_CP014849.1 position 19166

Table 4 — Information about seven simulated datasets enriched with Cry genes.

We formed five datasets by mixing "TOY dataset" with Illumina reads simulated by ART-simulator [96] with parameters "`hs20 -l 100 -f 20 -m 180 -s 10`" from BtA_{pure} and BtB_{pure} enriched with IPG CDSs (Table 4).

We launched ORFograph on the SPAdes v3.14.1 assembly graphs and the contigs generated from the simulated read-sets. To avoid false positive alignments, we decided to exclude alignments of domain HMMs that describe a wide range of toxins (rather than Cry proteins only) from consideration when running ORFograph on these datasets. Table 6 illustrates ORFograph results.

On the NoCry dataset, ORFograph identified no IPGs. The dataset Cry1Aa1 with only one Cry gene was assembled so that this Cry gene was located on a single contig, so ORFograph easily identified this gene.

When we considered datasets with two Cry genes, SPAdes failed to assemble the regions where these genes were located, while ORFograph found all Cry proteins. In the Cry1Aa1+Cry1Ab1 dataset with two similar CDSs, ORFograph printed out

Cry-genes (NCBI ID)	Cry-genes					
	Cry1Aa1	Cry1Ab1	Cry1Ac1	Cry1Ca1	Cry2Aa1	Cry2Ab1
Cry1Aa1 (aaa22353)	100	92	88	77	45	45
Cry1Ab1 (aaa22330)		100	88	76	45	45
Cry1Ac1 (aaa22331)			100	75	45	45
Cry1Ca1 (caa30396)				100	45	46
Cry2Aa1 (aaa22335)					100	89
Cry2Ab1 (aaa22342)						100

Table 5 — Information about the percent identity between six analyzed Cry proteins. Each cell shows the percent identity for between Cry1Aa1, Cry1Ab1, Cry1Ac1, Cry1Ca1, Cry2Aa1, and Cry2Ab1.

both genes, but also two chimeric sequences. In the case of a dataset with two distant Cry genes, ORFograph separated them, but also identified a substring of Cry1Aa1 as an ORF.

In the case where six Cry CDSs presented in the metagenome, ORFograph identified four clusters of ORFs: three of them represent Cry1-like proteins' CDSs and one represents Cry2-like proteins' CDSs.

Our analysis of the simulated dataset demonstrated that ORFograph successfully identifies Bt genes but may "glue" very similar sequences. It may also generate spurious genes when (i) there is limited information available from contigs, or (ii) due to imperfect representative selection procedure.

Dataset Name	TOY dataset + simulated reads from:	Unassembled by SPAdes	ORFograph results
NoCry	BtA_{pure} and BtB_{pure}	-	No Cry genes identified
Cry1Aa1	*BtA*: Cry1Aa1	-	One Cry1A-like CDS
Cry1Aa1 + Cry1Ab1	*BtA*: Cry1Aa1 + Cry1Ab1	Cry1Aa1, Cry1Ab1	Four sequences similar to Cry1Aa1 and Cry1Ab1 CDS
Cry1Aa1 + Cry1Ac1	BtA: Cry1Aa1 + Cry1Ac1	Cry1Aa1, Cry1Ac1	Three sequences: one is identical to Cry1Ac1 CDS, and the other two are similar to Cry1Aa1
AllCry	*BtA*: Cry1Aa1 + Cry1Ab1 + Cry1Ac1 and *BtB*: Cry1Ca1 + Cry2Aa1 + Cry2Ac1	Cry1Aa1, Cry1Ab1, Cry1Ac1, Cry1Ca1	Four clusters of ORFs: one Cry2-like (which belongs to one contig consisting of three edges) and three clusters of Cry1-like proteins scattered across several contigs.

Table 6 — Benchmarking ORFograph on five simulated datasets of varying complexity. In the result of AllCry dataset the second cluster contains Cry1Aa1 and Cry1Ab1-like sequences, the third cluster contains Cry1Aa1 and Cry1Ac1-like sequence, and the fourth cluster represents the Cry1Ca sequence.

2.7.3 Isolate IPG-containing bacteria samples from the NCBI database

We extracted 2749 datasets from the NCBI database and found datasets that potentially could contain Cry, Cyt or Vip toxin genes by aligning reads from each dataset to the extracted IPG sequences with the Diamond tool [97]. We selected a dataset for further analysis if there is IPG with at least 50% of its length covered by the dataset reads and the alignment identity of the reads is greater than 80%.

We used ORFograph to analyze the assemblies of the 342 datasets obtained by running SPAdes. Among these datasets, ORFograph identified 72 that contained IPG alignments distributed across multiple contigs. As a result, ORFograph reconstructed 419 novel cluster representatives from the selected datasets.

The Bacterial Pesticidal Protein Resource Center (BPPRC) typically considers only sequences with at least 95% amino acid identity (AAI) to existing IPGs as candidate IPGs. However, we chose to use a less strict criterion and considered all representatives with at least 90% identity to known IPGs that were not present in the non-redundant protein BLAST database. This led to the identification of 232 candidates.

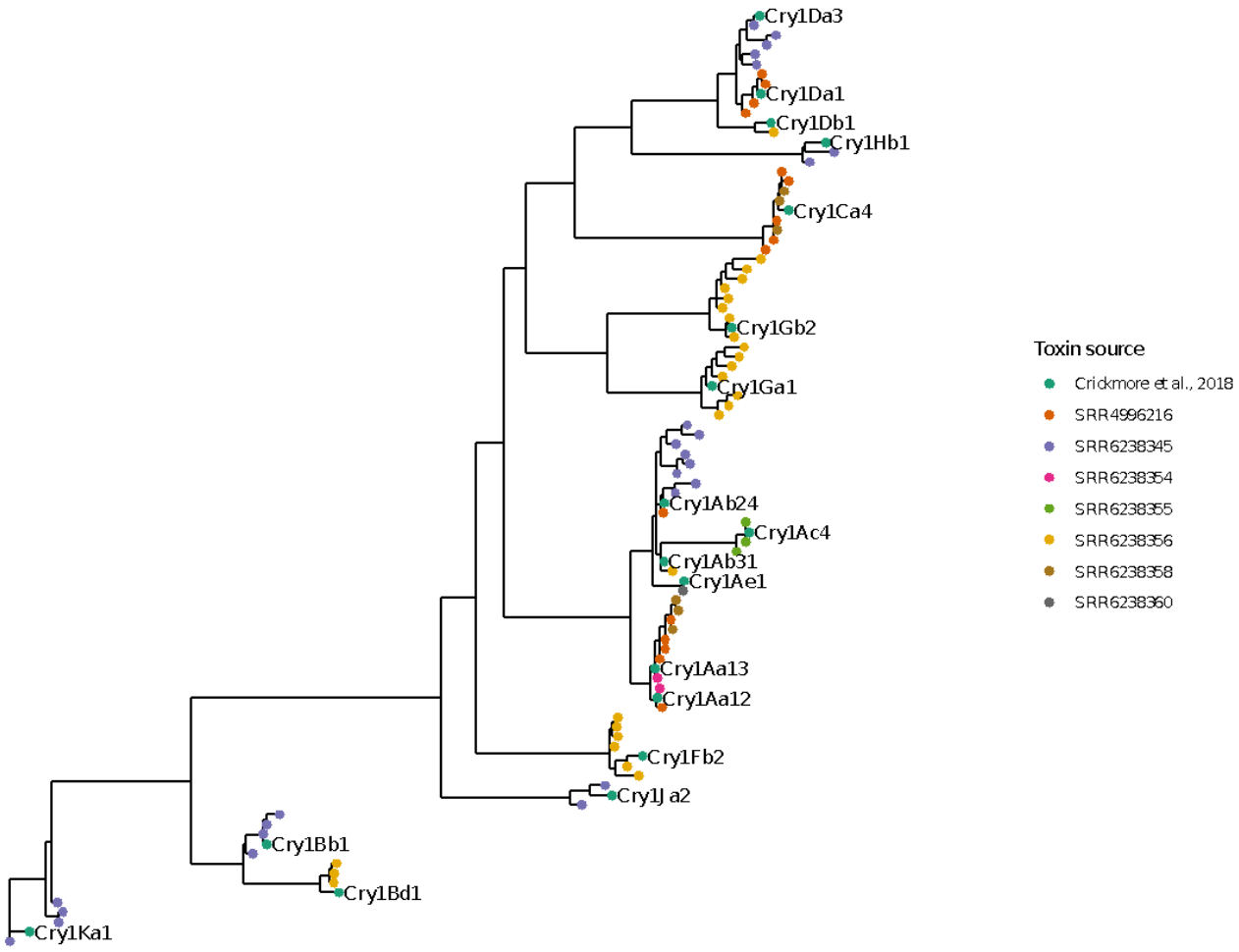

Figure 2.4 — Diversity of the reliable Cry1 sequences identified from isolated bacteria datasets. ORFograph was used to identify putative Cry1 novel IPGs, which were then aligned to sequences from the Bt nomenclature list [98] using Muscle v3.8.31 [99]. A maximum likelihood phylogeny was constructed using FastTree v2.1.10 [100] to visualize the relationships between the Cry1 sequences.

We found 77 out of 232 reliable IPGs in the datasets with acc. no. SRR6238356, SRR6238345, and SRR4996216. These datasets were previously analyzed using BtToxin_scanner in the study [101], where authors investigated the transfer of Cry-rich plasmids among different species. Our analysis of these datasets confirms the IPGs found in single contigs reported in [101]. However, BtToxin_scanner failed

to identify 59 of the novel candidate IPGs detected by ORFograph (see Table S4 in [101]). Furthermore, two potential IPGs identified by BtToxin_scanner were only partial fragments of the IPGs detected by ORFograph.

The dataset with acc. no. SRR4996216 comprises Illumina reads from *B. thuringiensis* serovar aizawai strain HD-133 and contains 17 identified IPGs. The assembly was carried out using the A5-miseq pipeline [102] in the original study [103], and genome annotation was performed using Prokka [104] and the NCBI's Prokaryotic Genome Annotation Pipeline [105]. In the original study, Cry1Aa, Cry1Ab, Cry1Ca, Cry1Da, Cry1Ia, Cry2Ab, and Cry9Ea were identified, including many partial or apparently fragmented genes. In contrast, using ORFograph, we identified 16 full IPGs, including novel variants of Cry1Aa, Cry1b, Cry1Ca, and Cry1Da.

We demonstrated that ORFograph significantly increases the number of identified IPGs compared to previous studies. However, it should be noted that this is not a true benchmarking analysis since we identified sequences that were not previously assembled. Figure 2.4 illustrates the value of ORFograph in discovering putative novel IPGs by comparing them to existing annotations in the IPG database.

2.7.4 New York City subway metagenome

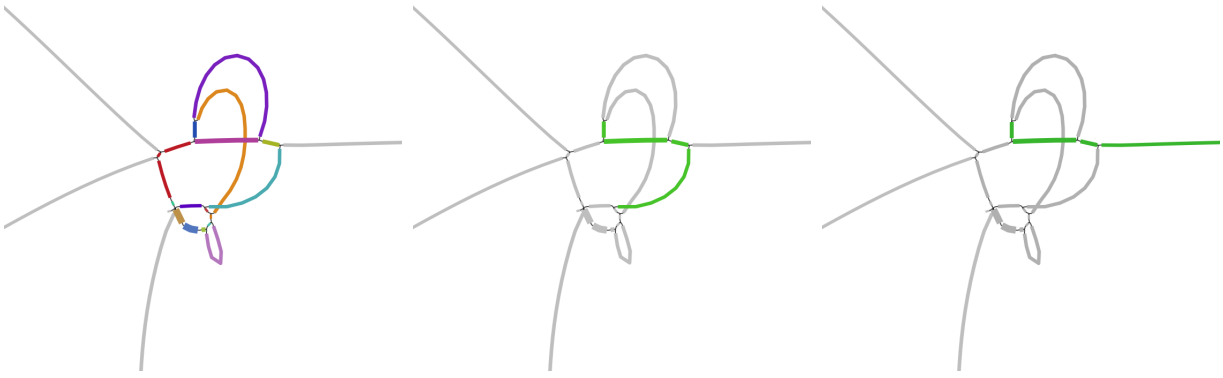

Figure 2.5 — A subgraph of the metagenomic assembly graph aligned to Cry4AA and Cry4Ba proteins. (Left) Each contig path is represented by edges of the same color. (Middle) The Cry4Aa gene-path. (Right) The Cry4Ba gene-path. Subgraphs were visualized using Bandage [68].

The authors of [106] investigated the microbial diversity present in the New York City subway system by analyzing read-sets from various metagenomic samples. Later genome of *B. thuringiensis* UBA3967 was assembled from one of its Illumina datasets with accession number SRR1748627 [107]. We run ORFograph on this dataset to search for Cry and Cyt genes that potentially can occure in *B. thuringiensis* UBA3967.

ORFograph found 48 clusters of putative IPGs in assembly graph generated by metaSPAdes v3.12 [92]. The majority of the representative clusters had a similarity of over 99% with thioredoxins, metallophosphoesterases, and disulfide reductases, which were identified from domain HMM alignments. The number of such false positives could potentially be decreased by producing more Cry and Cyt-specific HMMs. However, Cry60Aa, Cry60Ba, Cry11Aa, Cry10Aa, Cry4Aa, and two Cry4Ba genes, along with three Cyt toxins, were also discovered by ORFograph. Without the use of ORFograph, the Cry4 genes, which were located on different contigs (as shown in Figure 2.5), would have gone unnoticed.

2.8 Conclusions

In this chapter, we demonstrated that the analysis of assembly graphs reveals potential gene sequences that cannot be identified with traditional gene-finding methods. ORFograph combines HMM- and sequence-to-graph alignment methods and processes the results, producing a set of candidate genes that is consistent with the assembly graph structure and contig path information. To the best of our knowledge, ORFograph is the only pipeline that is able to extract gene sequences directly from an assembly graph based on domain HMMs or similar gene sequences. Thus, ORFograph can accelerate the identification of target genes in sequencing datasets.

Chapter 3. Automatic analysis of centromere sequence assemblies

3.1 Introduction

3.1.1 Human centromeres and their structure

The centromere is considered as a region of the chromosome, where the cell splits its DNA during the cell division process using special protein complex, *kinetochore*. The centromere plays the key role in chromosome segregation and inheritance in all vertebrate genomes, thus it can directly affect genetic disorders. It is eminently important to understand centromere structure, as it can help uncover evolutionary insights and fight against a lot of diseases, including cancer and infertility [108].

Futher in this work we consider only "live", or "active", human centromeres, i.e., subsequence of each centromere that associates with the kinetochore [17], mainly composed of so called alpha satellite arrays. Such centromeric sequences represent extra-long tandem repeats (ETRs) that cover around 3% of the human genome [109]. They are formed by sequences called *high-order repeats* (HORs) repeated thousands of times (Figure 3.1). HORs are formed by smaller blocks, called *alpha-satellites*, or *monomers*. The length of each monomer is around 171 bp. The similarity between two monomers within one HOR is around 50-90%. Within a single centromere, HORs usually consist of a single set of monomers, placed in a certain order. At the same time, the sets of monomers differ from centromere to centromere within the same organism. For example, the vast majority of HORs in the centromere of chromosome X [12; 110] consist of twelve monomers in a specific order, while the centromere of chromosome 8 represents a mix of two HORs, 7-monomer HOR and 11-monomer HOR (Figure 3.2).

In addition, some centromeric sequences contain substrings that are not formed by monomers (e.g., transposon insertions or LINE elements).

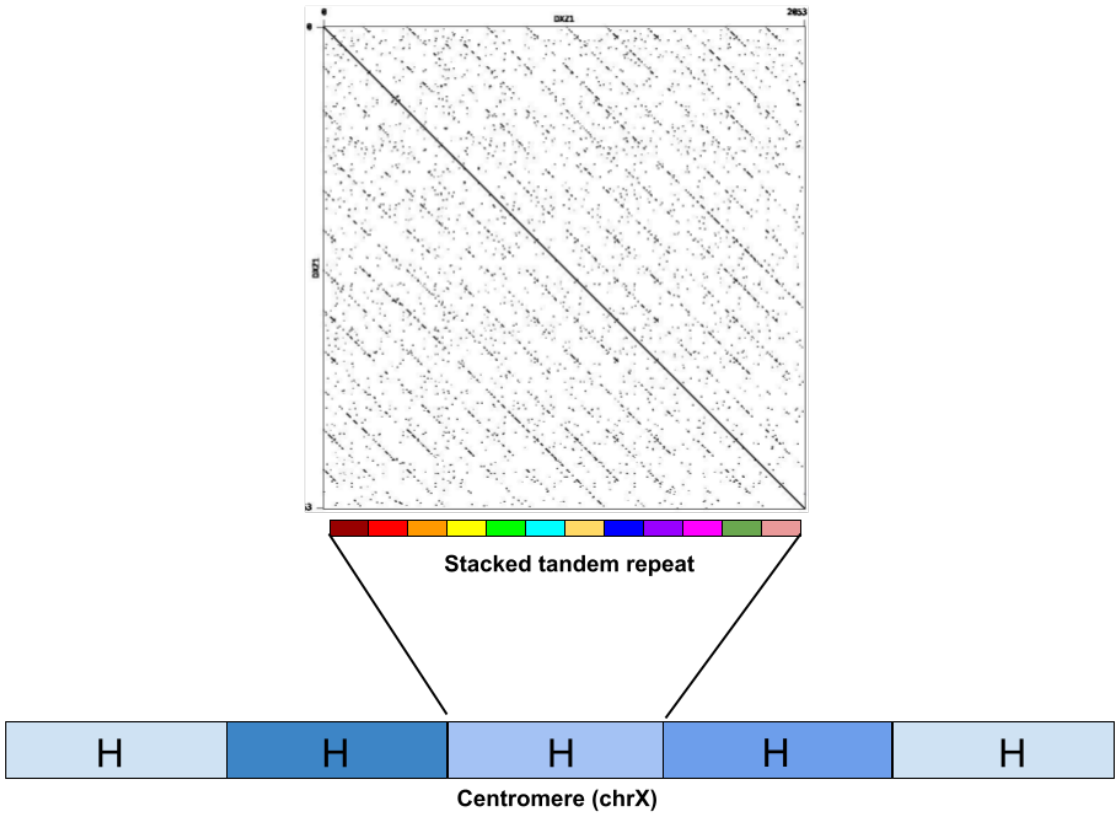

Figure 3.1 — The architecture of centromere on chromosome X. According to the assembly in [12], the centromere of chromosome X, referred to as cenX, comprises 18 100 monomers that are approximately 171 bp long. This information remains mostly unchanged in the newer T2T assembly [9]. These monomers are arranged into approximately 1500 units, five of which are highlighted in blue shades in the figure to demonstrate minor variations. Each unit corresponds to a high-order repeat (HOR), or stacked tandem repeat, which is composed of different monomers. Most of the units in cenX are canonical HORs consisting of twelve monomers, each represented by a different color. Although the canonical units are highly similar (95-100%), the monomers have a lower similarity of 65-88%. In addition to the canonical HORs, cenX has a few partial and auxiliary HORs that have different number of monomers. The figure above the text displays a dot plot of the nucleotide sequence of the canonical HOR, depicting the twelve monomers.

3.1.2 Centromere Evolution Postulate

A monomer is considered *frequent* if the number of its alignments per centromeric sequence exceeds a predetermined threshold, and *infrequent* otherwise.

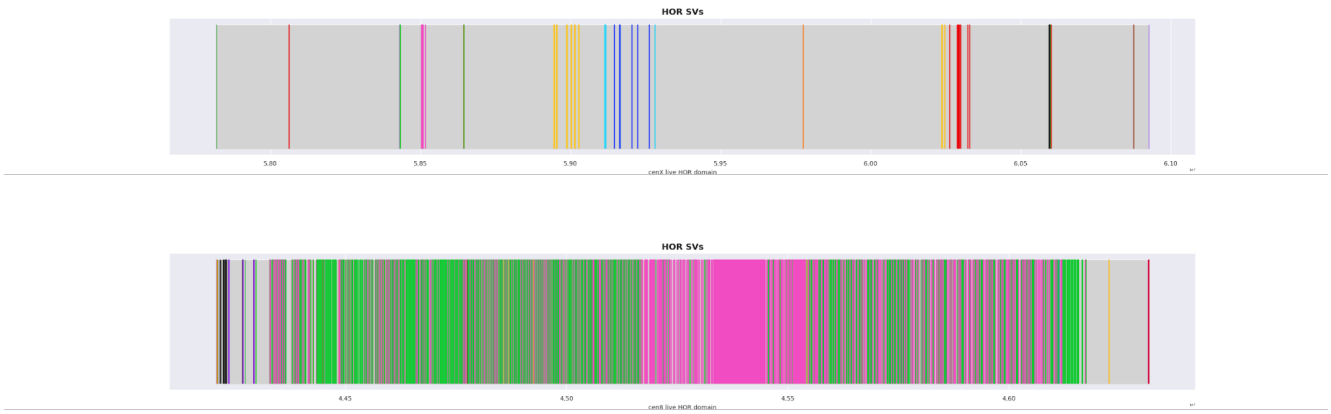

Figure 3.2 — Decomposition into HORs for centromere of chromosome X (cenX) and centromere of chromosome 8 (cen8). (Top) Decomposition into HORs for cenX. 12-monomer HORs represented as grey color vertical lines. LINE element is colored in black. Each other color represents HOR of different structure. (Bottom) Decomposition into HORs for cen8. Frequent 7-monomer HOR is colored in magenta, while frequent 11-monomer HOR is colored in green.

A monomer is called *hybrid* if it is a concatenation of two or more frequent monomers. The presence of hybrid monomers in centromeric sequences was first shown in the works [12; 13; 21], and it was also suggested that this is how new monomers emerged. *Canonical* HOR of a particular centromere is a cyclic sequence of frequent non-hybrid monomers that has evolved over time into the current centromere sequence. Unfortunately, at the moment, there is no more constructive definition of a *canonical* HOR.

One of the most difficult and at the same time important problems related to centromeres is the problem of understanding how centromere sequences arose and evolved [13; 111–113]. Despite the fact that there is no generally accepted theory of the centromere evolution, there is a set of principles that reflect the current view on this theory. In our work, we have combined principles important for the development of tools for centromeric sequence analysis into the Centromere Evolution Postulate.

The Centromere Evolution Postulate (CE Postulate), first described in [24], includes four points:

- Every existing human centromere originated from a single canonical HOR formed by k *different* monomers in a specific order.
- Each frequent non-hybrid monomer in the centromere corresponds to exactly one monomer in the canonical HOR.
- Each hybrid monomer is derived from the concatenation of two or more canonical HOR monomers.

– The centromere sequence consists of canonical HOR units, *partial* HOR units that are substrings of the canonical HOR, and *auxiliary* HOR units consisting of a single hybrid monomer.

3.1.3 Towards automatic centromere annotation

In 2001 Human Genome Project provided the first and most complete human genome assembly [8]. However, some complex parts, including centromeres, remained unassembled due to the lack of suitable sequencing technologies [9]. Though most attempts to analyze centromeric sequences were semi-manual, focused on single experimental cases, this non-systematic research has greatly expanded our understanding of centromere structure and evolution during these years, starting from the basic understanding that alpha-satellite monomers represent centromeric regions and are structured into more sophisticated high-order repeats [114] to building artificially constructed centromere sequences [13; 23] and revealing the importance of centromeric structure in homeostasis and genome stability [115].

Advancement in third generation sequencing technologies opened possibilities to assemble extra long tandem repeats (ETRs), including centromeres [9;12]. In April 2022, Telomere-to-Telomere (T2T) Consortium presented a new human genome reference where all centromeric sequences had been reconstructed [9]. This work has become an important milestone and led centromere exploration to a new level. Since more high quality centromeric data has become available, scientific community needs new instruments to analyze it automatically.

In this chapter, we present a set of instruments, StringDecomposer [21], CentromereArchitect [25], and HORmon [24], to perform automatic centromere sequence analysis, that were developed during the author's work in the T2T Consortium. In addition, this chapter discusses results obtained using these instruments that show new insights on centromere evolution.

The initial problem for centromere analysis was to efficiently and accurately represent long error-prone reads and assemblies of centromeres as a string of already known monomers, further referred as String Decomposition Problem. Representing reads as a string of monomers is essentially important for assembly tools [12; 116] and evolutionary studies if centromere is not assembled or only partially assembled

to perform comparative analysis [110]. CentroFlye [12], the centromere assembler for ultralong error-prone reads, uses reads in monomer alphabet representation to build an assembly graph and neglect read errors on a nucleotide level. It is very important for such representation to be accurate in order to avoid errors in assembly graph construction, and we developed StringDecomposer for this purpose.

Regular sequence aligners, such as Minimap2 [19], have recently been shown to produce poor results in highly repetitive regions and thus cannot be used to solve the String Decomposition Problem [117; 118].

More specialized tools such as Tandem Repeats Finder [119] and Noise-Cancelling Repeat Finder [117] provide only partial solutions to the problem. Tandem Repeats Finder (TRF) [119] processes a string containing a tandem repeat without any prior knowledge about the repetitive substring, i.e., monomer, or its length. TRF reports a consensus of all monomer occurrences and the location of each such occurrence in the input string. TRF does not provide an option to identify different monomers and needs further processing of its output to solve the String Decomposition Problem. Another complication of TRF output is that it reports different cyclic shifts of a monomer for each processed centromeric sequence. The Noise-Cancelling Repeat Finder (NCRF) [117] takes a string containing a tandem repeat and one monomer as input and partitions the string by occurrences of this one monomer. Thus, it was not designed to work with several monomers.

Alpha-CENTAURI [120] and PERCON [121] are tools that were specifically developed to work with satellite monomers. The String Decomposition Problem is addressed by PERCON through a comparison of the octanucleotide content of a proposed monomer sequence with the octanucleotide content of all input monomers. However, PERCON's benchmarking is challenging because it is only compatible with the Windows operating system. While Alpha-CENTAURI [120] solves a more general problem, on its first stage Alpha-CENTAURI aligns a pre-trained HMM of a consensus monomer to all reads using HMMer [89] providing accurate starting and ending positions for each monomer occurrence. Using a set of known monomers, one can classify each occurrence by the monomer with the highest identity to the corresponding substring and compare the Alpha-CENTAURI decomposition with any other.

The natural extension of StringDecomposer is an automatic solution for full centromere annotation, i. e., an instrument that takes as input a set of human centromeres or reads and a consensus monomer sequence and outputs a set of

corresponding monomers (solution to the Monomer Inference Problem), a set of corresponding HORs (solution to the HOR Inference Problem), and input sequences' decomposition into extracted monomers and HORs. Notably, the correct answer to all these questions depends on our definition of the initial HOR sequence and how centromeric sequence changed in the process of evolution.

Most previous approaches to tackling these problems were also semi-manual and not designed to answer evolutionary questions. Such previous approaches as ColorHOR [122] and HORdetect [123] tried to extract HORs from unassembled genome sequences, and due to limitations of sequencing technologies their main challenge was to assemble full HOR sequences rather than understand how different HORs in one centromeric sequence relate to each other. In addition, the exact objective function for HOR inference was not specified making the HOR concept highly dependent on parameters used for monomer-set generation [24].

Alpha-CENTAURI [120] aims to tackle a broader problem by identifying monomers and arranging them into HORs. However, its output often does not match manually obtained HORs [21]. Moreover, the monomer clustering in Alpha-CENTAURI is not ideal, and the accuracy of string decomposition decreases as the tool produces false abnormal HOR structures.

We proposed two main approaches to the centromere annotation problem. CentromereArchitect (CA) was the first fully automatic solution for centromere annotation [25]. It extracts monomer sequences by performing clustering of monomer blocks obtained by running StringDecomposer on a centromeric sequence and one consensus monomer. After that, CA solves the HOR Inference Problem by reducing it to a compression problem. In CA the main idea of solving HOR Inference Problem is to remove redundancy information from centromeric sequence by collapsing identical parts of it. While the HORmon tool uses monomers from CentromereArchitect as an initial step for Monomer and HOR Inference Problems, it tries to solve them together [24]. Although it is hard to provide data-grounded arguments as to which approach is more justified from evolutionary standpoint, current biological understanding of how centromeric sequences change over time, summarized in Centromere Evolution Postulate [24], is better reflected in the HORmon procedure.

3.2 String Decomposition Problem

The problem of String Decomposition can be stated as follows. Given a string R representing centromeric sequence and a set of strings called *Blocks*, each of which corresponds to one *monomer*. A concatenation of several *blocks* is defined as a *chain*. An *optimal chain* for R is defined as a chain of *Blocks* that results in the highest-scoring global alignment against R when aligned against all possible chains. The task is to find an optimal chain for R.

3.2.1 String Decomposition Graph

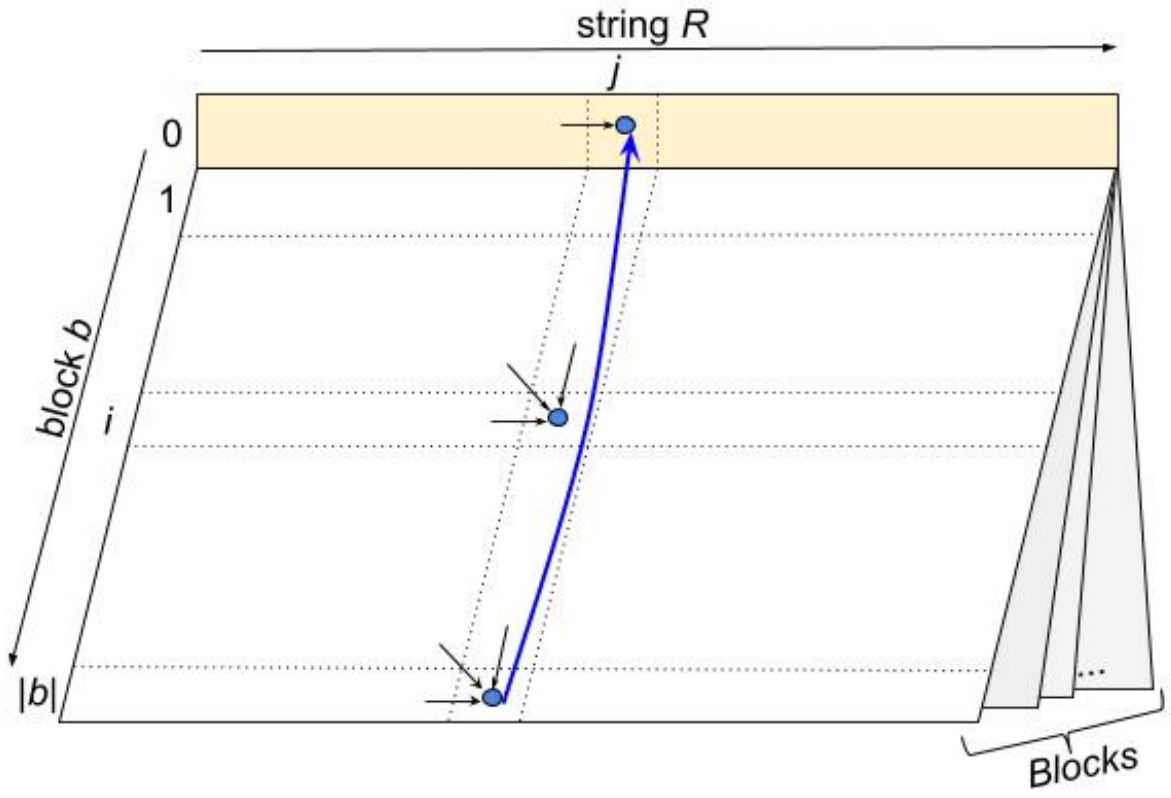

Figure 3.3 — The String Decomposition (SD) graph can be visualized as a "book", where each "page" corresponds to an alignment matrix for a single block, and the pages are connected by their 0-th row. The edges of the String Decomposition graph are represented by arrows. In particular, the edge that corresponds to switching between blocks is colored in blue.

Consider a standard *alignment graph* [124] of two strings, string R and block b from $Blocks$. It is usually represented as a matrix, where each cell with (i,j) coordinates represents a vertex of the alignment graph and stores an optimal alignment score between the prefix R of length i and the prefix b of length j. Directed edges between vertex (i,j) and vertices $(i+1,j)$, $(i,j+1)$, and $(i+1,j+1)$ correspond to insertions, deletions, and matches/mismatches, respectively.

String Decomposition graph (*SD graph*) for solving String Decomposition Problem can be represented as a set of alignment graphs for each block b and string R "glued together" by their 0-th row (Figure 3.3). Except for vertices in 0-th row, each vertex is represented as a triplet (b,i,j), where b referred to a block, i — position in a block b, and j — position in R. Vertices in the 0-th row are represented as a pair $(0,j)$, where j represents position in R. Furthemore we introduce new block-switching edges (blue edge in Figure 3.3) from each cell in the last row $(b,|b|,j)$ to the cell in the 0-th row $(0,j)$. These block-switching edges have zero weight and provide a transition from the end of one block alignment to the start of the next block alignment.

The formal definition of String Decomposition graph is the following. Each triplet (b,i,j), where b is a block, $1 \leq i \leq |b|$, and $0 \leq j \leq |R|$, introduces a single vertex. Each pair $(0,j)$, where $0 \leq j \leq |R|$ and 0 represents a starting position of all blocks, introduces a single vertex. Regular alignment graph edges between vertex (b,i,j) $((0,j))$ and $(b,i+1,j)$, $(b,i,j+1)$, $(b,i+1,j+1)$ represent insertions, deletions, and matches/mismatches, respectively. Insertion and deletion edges have a weigth of $-\delta$, mismatch edges have a weight of $-\sigma$, and match edges have a weight of 1. Block-switching edges with zero weight connect a vertex $(b, |b|, j)$ to a vertex $(0, j)$ for each block b. The vertex (0,0) is referred as a *source* and the vertex $(0,|R|)$ as a *sink* of the SD graph. Since all directed cycles in the SD graph have negative weights, the heaviest path from source to sink in this graph is well-defined.

3.2.2 The StringDecomposer algorithm

In the case of string R and one block b the solution for the String Decomposition Problem, called a *wrap-around dynamic programming algorithm*, was proposed by [22]. Matroud et al. [125] extended this algorithm to any number of

blocks and implemented the solution for two blocks in the NTRFinder software tool [126], which is not available anymore. In this section, we formally describe the StringDecomposer, a generalized wrap-around dynamic programming algorithm that solves the String Decomposition Problem for an arbitrary number of blocks.

Consider an SD graph for $|Blocks|$ blocks and string R and scoring scheme with $-\delta$ penalty for indels, $-\sigma$ penalty for mismatch, and $+1$ reward for match. Considering string S and index i of S (i.e., $0 \leq i \leq |S|$), $S[:i]$ is defined as a prefix of S of length i, and $S[i]$ is defined as the i-th symbol of string S. StringDecomposer uses dynamic programming to compute $score(b,i,j)$ for each vertex of SD graph, which is defined as a score of the optimal global alignment to $R[:j]$ among all possible block chains ending on $b[:i]$. Except vertices with incoming block-switching edges, the $score(b,i,j)$ for each vertex is defined as maximum among three values $\max(score(b,i-1,j) - \delta,\ score(b,i,j-1) - \delta,\ score(b,i-1,j-1) - \sigma)$, if $b[i]$ is not equal to $R[j]$), or $score(b,i-1,j-1) + 1$, if $b[i]$ is equal to $R[j]$, as in regular alignment problem [127]. The $score(0,j)$ is defined as a maximum among all $score(b,|b|,j)$. The initial variables are $score(b,0,j) = 0$ for all possible b and j, and $score(b,i,0) = -i \cdot \delta$ for all i.

After running dynamic programming algorithm the variable $score(0,|R|)$ stores the score of the heaviest path from source to sink. The heaviest path in this graph defines the optimal alignment and, thus, solves the SD Problem. Each path in the SD graph corresponds to only one possible alignment of chain of blocks. Conversely, each alignment of a chain of blocks corresponds to only one path in SD graph. In order to restore alignment string, consider the last block-switching edge from the vertex $(b,|b|,|R|)$ to the sink $(0,|R|)$, that gave the $score(0,|R|)$. The optimal chain's final block is represented by a block b that has the maximum score. To obtain the other blocks in this chain, we can backtrack from the sink $(0,|R|)$ to the source $(0,0)$.

The SD graph is not acyclic, but it can be transformed into an equivalent acyclic graph by introducing new vertices $(0,j+1/2)$ and substituting each block-switching edge ending in a vertex $(0,j)$ by two edges, a zero-length edge from its start to a new vertex $(0,j+1/2)$ and a zero-length edge from $(0,j+1/2)$ to $(0,j+1)$. The running time and the memory footprint of the heaviest path algorithm in a directed acyclic graph are proportional to the number of edges in this graph. Thus, the running time and memory footprint of the StringDecomposer algorithm are

proportional to $O(|R| \cdot length(Blocks))$, where $length(Blocks)$ is the total length of all blocks.

3.2.3 Transformation from the nucleotide alphabet to the block alphabet

Given a string R and a set of monomers $Monomers$, StringDecomposer generates the optimal chain of $Monomers$ for R. It returns the starting and ending positions $(s(m)), e(m))$ for monomer alignment m to R, as well as the identity $Identity(m)$ of this alignment, for each monomer alignment m in the chain. $R(m)$ will be denoted as a substring $R[s(m) : e(m)]$. In real-world applications, when StringDecomposer aligns a set of monomers to centromeric sequence (either assembly or reads), StringDecomposer may generate an unreliable alignment of monomer m with low $Identity(m)$. This is usually caused by read errors or an imperfect set of monomers provided to StringDecomposer that does not include all monomers in the given centromeric sequence. Another complication is that some centromeric sequences contain substrings that are not formed by monomers, e.g., transposon insertions, LINE elements, and decomposition of these substrings is meaningless. The problem of false monomer identification becomes paramount while processing error-prone reads (e.g., ONT reads).

SD uses the following postprocessing procedure to highlight possible false alignments. Consider the alignment of the monomer m to the centromeric string R in decomposition of R into monomers. Starting and ending positions of this alignment are ($s(m)$,$e(m)$). The alignment identity is denoted as $Identity(m)$. Consider the the alignment of the monomer m' to positions ($s(m)$,$e(m)$) that has second best alignment identity $Identity(m')$ to $R[s(m) : e(m)]$.

StringDecomposer classifies each monomer alignment m as *reliable*/*unreliable* using logistic regression model with two features: $Identity(m)$ and $IdentityDiff = Identity(m) - Identity(m')$. We compared this model with the one-predictor model, solely based on $Identity(m)$, and showed its superiority over the one-predictor model.

We trained logistic regressions on 1000 ONT reads aligned to cenX using TandemMapper [118](referred as ONT_CENX), and 1000 ONT reads aligned to

other regions of chromosome X assembly v0.7 with Minimap2 [19] (referred as ONT_NONCENX). Centromeric reads containing the LINE element were excluded from the ONT_CENX set. Both baseline and two-predictor logistic regression models were trained on train set constructed from 80% of ONT_CENX and ONT_NOTCENX reads and tested on test set constructed from the remaining 20%. Both models produced zero false-positive alignments. The two-predictor model produced 198 false negative alignments, and the one-predictor model produced 241 false-negative alignments out of 65,842 monomer alignments in test set (see Figure 3.4).

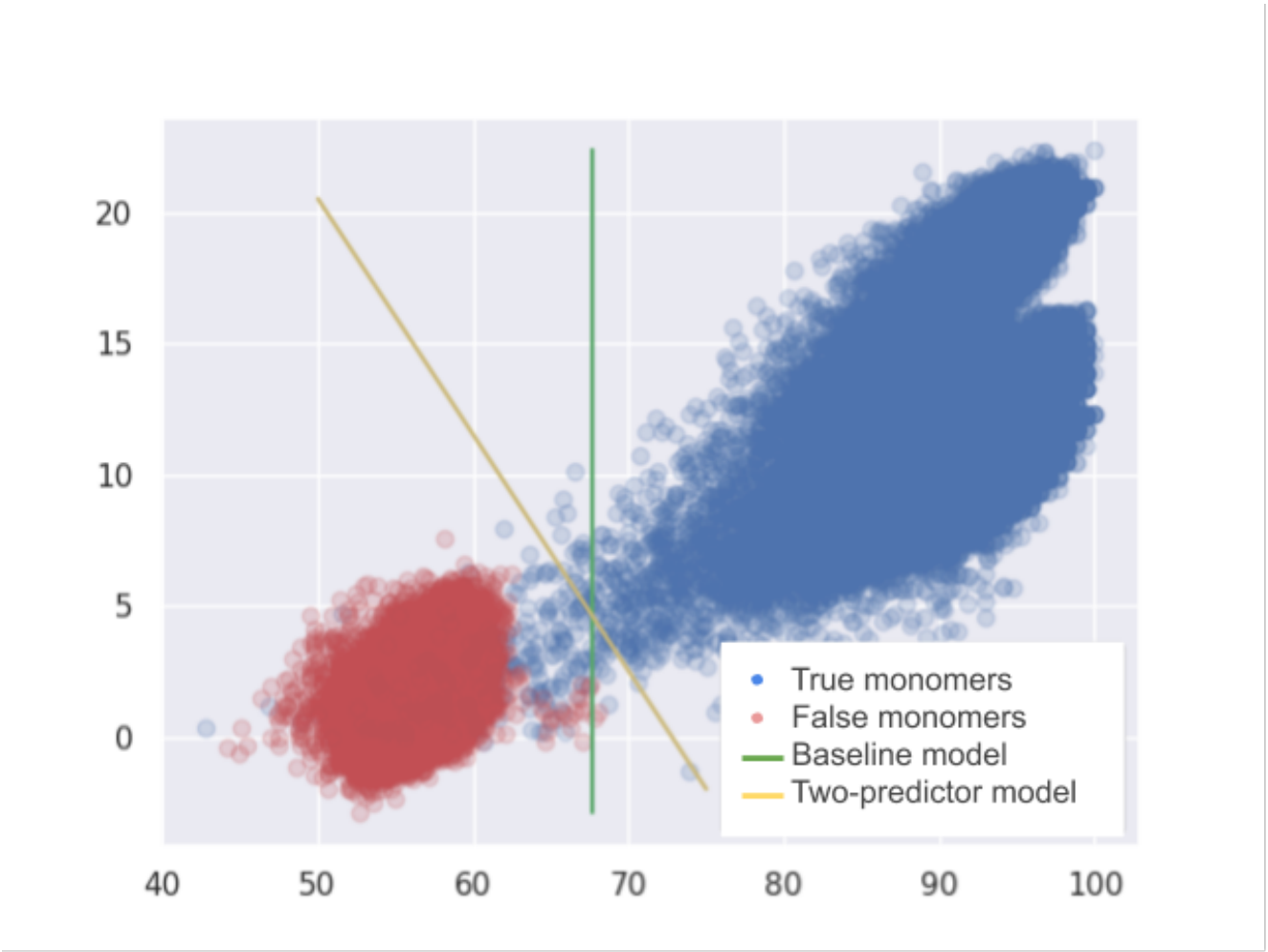

Figure 3.4 — The distribution of $Identity$ (x-axis) and $IdentityDiff$ (y-axis) for monomer alignments in centromeric regions. Each dot in the graph represents either a true monomer alignment (blue dots) or false monomer alignment (red dots). The green line separates true and false monomers according to baseline classifier based on $Identity$ value of each alignment, while the yellow line separates the true and false monomers alignments according to the logistic regression trained on both $Identity$ and $IdentityDiff$ values.

In final output, StringDecomposer marks unreliable monomers with *gap symbol* "?". This leads to a conversion of the string R into the *extended monomer alphabet*, which comprises all monomers and the gap symbol.

3.3 StringDecomposer Results

3.3.1 Datasets

In this section, we compare the StringDecomposer tool (further referred as SD) to the only current solution to String Decomposition Problem, Alpha-CENTAURI (further referred as AC), on ONT reads from centromere of chromosome X. For benchmarking, we used the chromosome X v0.7 assembly, the first full chromosome assembly provided by the T2T consortium [116]. The monomer set was derived from HOR consensus sequence DXZ1* [12]. DXZ1* was decomposed into twelve monomers, denoted by capital latin letters ABCDEFGHIJKL, using Alpha-CENTAURI [120] with manual curation.

Our study analyzed 11 069 717 Oxford Nanopore reads generated within the T2T Consortium initiative. The reads had a total length of 155 Gb and provided 50X coverage, with an N 50 read length of 70 kbp. Among these reads, 2680 (total read length 132.9 Mbp) were aligned to cenX as reported in [12]. Out of the 2680 cenX reads, 1442 were aligned to the reference by TandemMapper [118], providing a ground truth alignment to the reference sequence. These mapped reads comprised the set of reads with a total length of 121 Mbp and a total length of aligned fragments of 76 Mbp. We trimmed the unaligned prefixes and suffixes of each read and considered only the aligned substring of each read as a single *Read*. We referred to the substring of cenX where each *Read* aligned as its $origin(Read)$, and we considered the set of all such aligned substrings as *Reads*.

All scripts used to analyze this dataset are available on GitHub https://github.com/TanyaDvorkina/sdpaper.

3.3.2 Benchmarking StringDecomposer on error-prone reads

Consider a centromere sequence (a read sequence) S and a set of monomers $Monomers$. We will define a monocentromere (monoread) $mono(S)$ as a string in the monomeric alphabet that represents the decomposition of S. Consider $Read$ from $Reads$ and its decomposition $mono(Read)$ into monomers. In order to validate its decomposition, we can compare $mono(Read)$ with $mono(origin(Read))$, which represents the "ground truth" monomer decomposition of the corresponding reference substring. AC and SD decompositions were compared using this approach.

We applied both SD and AC tools to the cenX sequence, which has a length of 3.1 Mbp, to obtain the monocentromere sequence, denoted as $mono(cenX)$. Both SD and AC generated the same monocentromere sequence, with a median percent identity of monomer alignments across the cenX sequence of 98.8%. The resulting $mono(cenX)$ sequence consists of 18 103 monomers and 36 gap symbols, which indicate the presence of a LINE element in the region. Figure 3.5 displays the distribution of percent identities for the twelve cenX monomers and the gap alignments.

We used both SD and AC tools to convert each read in $Reads$ into a monoread: $mono_{SD}(Read)$ and $mono_{AC}(Read)$, respectively. We then aligned each of these monoreads to $mono(origin(Read))$ using ordinary edit distance scoring. Each alignment column consisted of a pair of symbols, where the first symbol corresponded to a position in $mono(origin(Read))$, and the second symbol corresponded to a position in $mono(Read)$. The symbols included twelve monomers, the "?" symbol to represent unreliable monomers, and the "-" symbol to represent an indel in the alignment. An example of $mono(Read)$ to $mono(origin(Read))$ alignment with all possible outcomes for a specific column is shown in Figure 3.6.

The overall alignment statistics on $Reads$ for both tools is presented in Table 7. The majority of errors for both tools are monomer-gap mismatches that usually occur due to errors in ONT reads. SD resulted in a three-fold reduction in errors as compared to AC (1183 versus 3478), showing better processing of corrupted regions in ONT reads.

In both cases, the overall percentage of errors is very low: 0.86% for AC and 0.32% for SD. While the difference between the two approaches may seem minor, for certain applications, such as monocentromere assembly, the length of

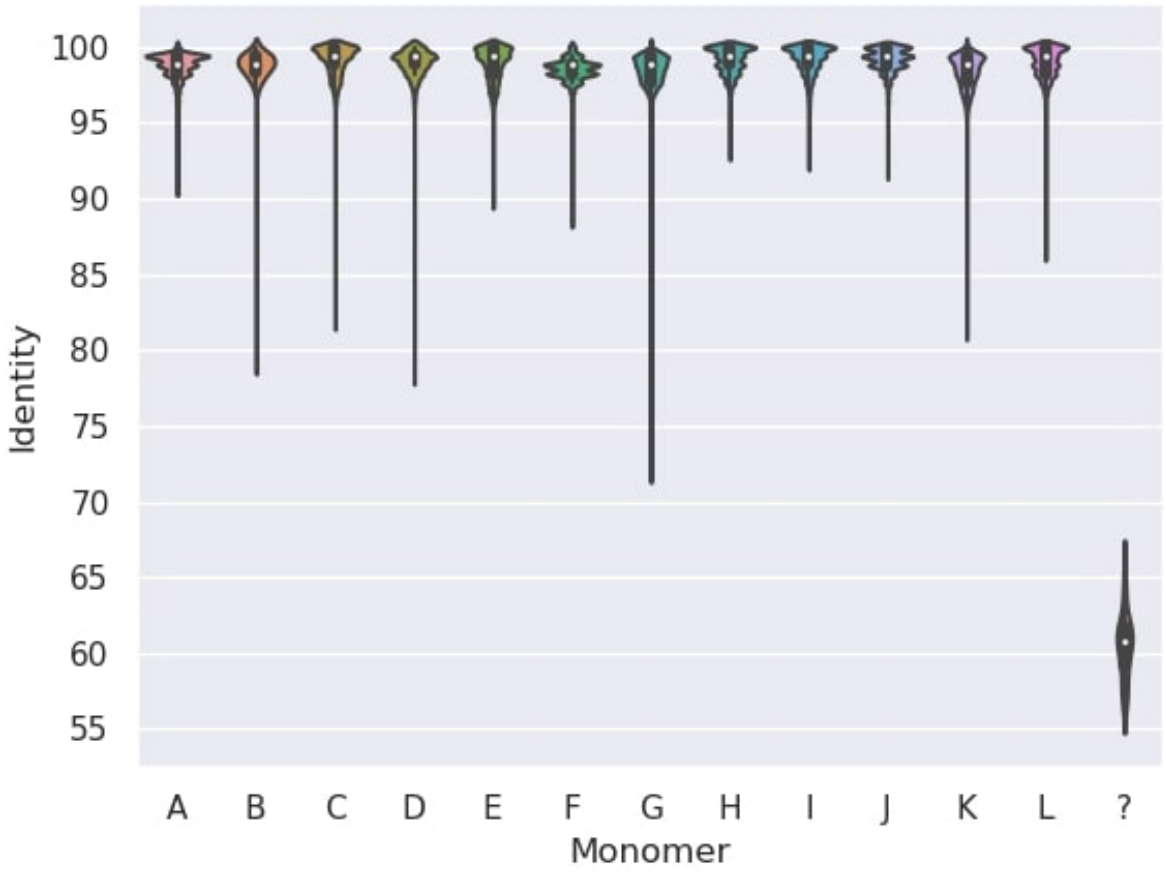

Figure 3.5 — Distribution of percent identities for each of the twelve cenX monomers and the gap monomer "?". Each violin plot represents a different monomer and shows the range and frequency of percent identity values across the cenX region. The monocentromere X contains a LINE element at positions $CHM13 : X : 2773652\text{-}2779726$, which is represented by 36 consecutive gaps ("?") in the sequence.

error-free k-monomers in monoreads can be crucial to the outcome. For instance, assuming errors are uniformly distributed (which is an unrealistic assumption), SD provides information on 312-monomers, while AC only provides information on 116-monomers in monoreads. This is because $100/0.32 = 312$ and $100/0.86 = 116$.

3.3.3 Identifing first hybrid monomers

In [13] authors analyzed the monomer-set generated by reference models [23] and explained how monomers can arise through the concatenation of two or more monomers, which they called *hybrid* monomers. By examining monomer-monomer

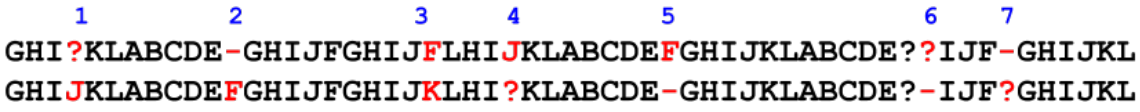

Figure 3.6 — Monoread-to-monocentromere alignments. The first row represents $mono(origin(Read))$, the second row represents $mono(Read)$. The matching positions are shown in black and positions with errors are shown in red. The following types of errors are shown: gap-monomer mismatch (1), monomer-insertion (2), monomer-monomer mismatch (3), monomer-gap mismatch (4), monomer-deletion (5), gap-deletion (6), gap-insertion (7).

	SD	AC	SD	AC	SD	AC	SD	AC
Read → cenX ↓	monomer		?		-		Total	
monomer	119	139	1183	3478	117	154	1419	3771
?	0	0	-	-	19	19	19	19
-	0	6	18	20	-	-	18	26
Total	119	145	1201	3498	136	173	1457 (0.32%)	3816 (0.86%)

Table 7 — Summary of errors in the alignments between monoreads and monocentromeres computed by the AC (black) and SD (blue) tools for 12 monomers. The table shows the errors for decomposition into 12 known cenX monomers. The symbols used in the table include "monomer" which represents one of the twelve cenX monomers, "?" which represents a gap symbol, and "-" which represents a space symbol that indicates an indel in the alignment of $mono(Read)$ against $mono(origin(Read))$. Each cell (i, j) in the table represents the number of times a symbol of type i in $mono(origin(Read))$ was aligned to a symbol of type j in $mono(Read)$. The number of matches for the 12 monomer decomposition is 445 169 for SD and 442 783 for AC.

mismatches identified by the SD and AC approaches, we discovered instances of *hybrid* monomers in actual cenX assembly.

Specifically, out of 119 (139) monomer-monomer mismatches identified by SD (AC), 104 (115) involved substitutions of the monomer K with the monomer F. These mismatches occurred in a non-standard 16-monomer HOR (ABCDEFGH**I**J**F**GHIJKL), where the second occurrence of F is frequently replaced by K. Similarly, in the non-standard 11-monomer HOR (ABCDEFGH**I**J**K**), the monomer K is typically substituted by L in a similar pattern. These findings are illustrated in the heatmap shown in Figure 3.7.

We considered all six occurrences of the non-standard HOR ABCDEFGHIJ**F**GHIJKL and all six occurrences of the non-standard HOR ABCDEFGHIJ**K**GHIJKL in the monocentromere X, extracted nucleotide string

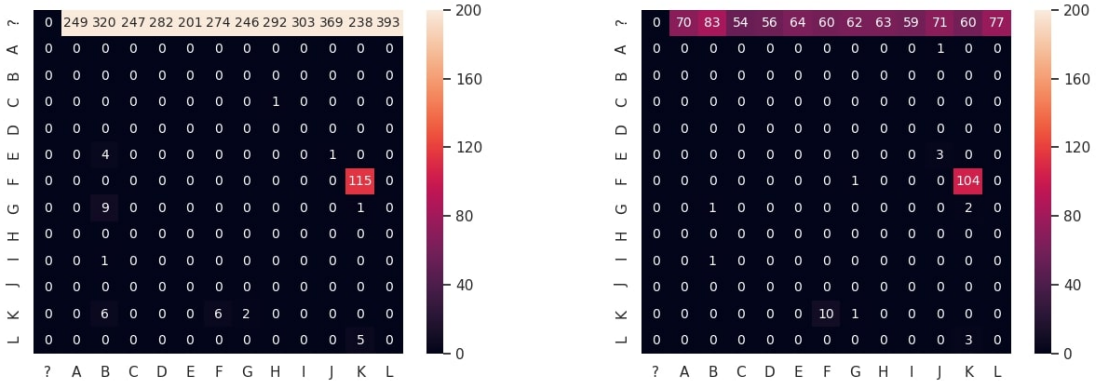

Figure 3.7 — Mismatch substitution matrices for the AC (Left) and SD (Right) approaches. Each cell (X,Y) indicates the number of occurrences where symbol X in the monocentromere was replaced by symbol Y in the monoread. Each symbol represents either a monomer or a gap symbol "?".

of **F** from each occurrence of ABCDEFGHIJ**F**GHIJKL and nucleotide string of **K** from each occurrence of ABCDEFGHIJ**K**GHIJKL (further referring to these monomers as **FK**), and built alignment matrix (Left panel of Figure 3.8), where each cell shows pairwise identity of two nucleotide **FK** strings. Monomer occurrences 1-4 and 11–12 were classified as F in monocentromere X, while monomer occurrences 5-10 were classified as K. Indeed, 1-4 are similar to monomer F (98-100% identity). While monomer occurrences 5-12 are very similar to each other, they have low identity scores for both F and K.

We analyzed pairwise alignment between occurrences 5–12 of **FK** and monomers F and K. Our analysis revealed that 5–12 represent occurrences of *hybrid* monomer formed by the first half of monomer K and the second half of monomer F. If we construct synthetic monomer, **K**+**F**, from the first part of monomer K (ends on 98 position of K) and the second part of monomer F (starts from 101 position of F), and align it to 5–12 occurrence of **FK**, the alignments will have identity of $\sim$ 97% (comparing to 91% identity to F and 92% identity to).

In order to show that such *hybrid* monomer is supported by the reads, we extracted all monoreads that contain non-standard HORs ABCDEFGHIJ**F**GHIJKL(272 monoreads) and ABCDEFGHIJ**K**GHIJKL (105 monoreads). 154 occurrences of monomer **FK** were assigned to the first cluster, i.e., sequences 1–4 on Left panel of Figure 3.8, and 223 were assigned to the second cluster, i.e., sequences 5–12 on Left panel of Figure 3.8. We aligned nucleotide

subsequences of **FK** to monomers F and K. Right panel of Figure 3.8 clearly demonstrates that sequences can be classified into two clusters based on their identity to F and K, and this classification coincides with the grouping on the Left panel of Figure 3.8.

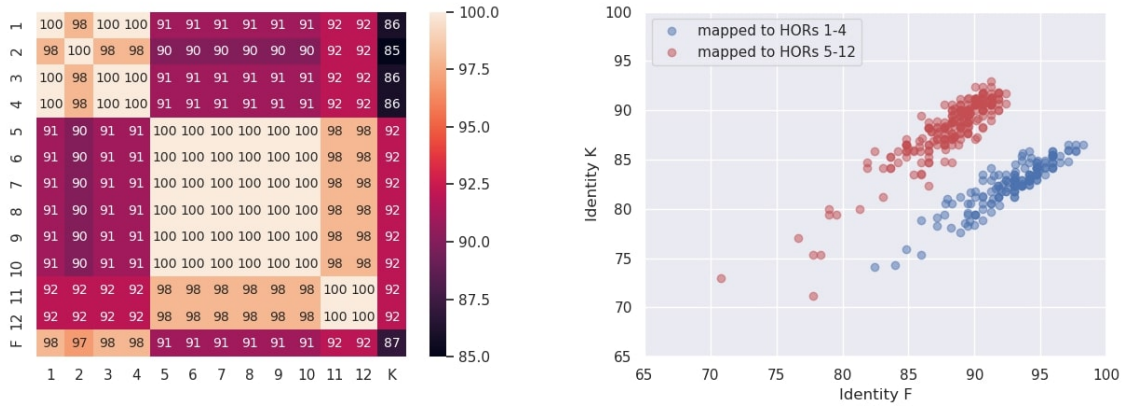

Figure 3.8 — Analysis of the non-standard HOR ABCDEFGHIJFGHIJKL. (Left) The left panel shows a heatmap that displays the identities between occurrences of **FK** in twelve non-standard HORs ABCDEFGHIJ**F**GHIJKL and ABCDEFGHIJ**K**GHIJKL in cenX, as well as monomers F and K. (Right) The right panel shows the identity of 377 alignments of monomer F (x-axis) and monomer K (y-axis) against the corresponding substring of the read for all 377 occurrences of ABCDEFGHIJ**F**GHIJKL or ABCDEFGHIJ**K**GHIJKL in *Reads*. Blue (red) circles represent occurrences of monomer F or K aligned to **FK** from HORs 1–4 (HORs 5–12).

Table 8 shows the results of SD and AC recomputed on the set of 12 original monomers enriched with monomer **K**+**F**. Monocentromere was recomputed with 13 monomers, and occurrences 5–12 in it were classified as a monomer **K**+**F** rather than as monomers K or F. The number of monomer-to-monomer mismatches decreased from 119 (139) to 11 (47) for SD (AC), while the number of other errors did not change.

These results showed the potential mechanism of new monomer generation. It seems that during DNA replication, the second half of monomer **K** and some monomers located further were somehow skipped until the second half of monomer **F** was glued, starting regular replication. It is worth to note that monomers **K** and **F** were glued by conservative part identical in both sequences, and the length of resulting monomer is 171 bp.

	SD	AC	SD	AC	SD	AC	SD	AC
Read → cenX ↓	monomer		?		-		Total	
monomer	11	38	1256	3481	119	154	1386	3673
?	0	0	-	-	19	19	19	19
-	1	6	17	21	-	-	18	27
Total	12	44	1273	3502	138	173	1423 (0.31%)	3719 (0.83%)

Table 8 — Summary of errors in the alignments between monoreads and monocentromeres computed by the AC (black) and SD (blue) tools for 13 monomers. The table shows the errors for decomposition into 12 known cenX monomers or K+F-monomer. The symbols used in the table include "monomer" which represents one of the twelve cenX monomers or K+F-monomer, "?" which represents a gap symbol, and "-" which represents a space symbol that indicates an indel in the alignment of $mono(Read)$ against $mono(origin(Read))$. Each cell (i, j) in the table represents the number of times a symbol of type i in $mono(origin(Read))$ was aligned to a symbol of type j in $mono(Read)$. The number of matches for the 13 monomer decomposition is 445 206 for SD and 442 887 for AC.

3.4 Centromere annotation

This section provides overview of two instruments for complete centromeric sequence annotation, CentromereArchitect and HORmon. HORmon can be considered as the successor to CentromereArchitect.

3.4.1 CentromereArchitect pipeline overview

CentromereArchitect (CA) is the first automated tool for human centromere sequence annotation. CA takes as input a set of centromeric sequences *Centromeres* and an *InitialMonomer*, e.g. consensus monomer for all alpha-satellites sequences in human genome. Its pipeline consists of two main phases, Monomer Inference and HOR Inference. During Monomer Inference phase, it constructs a set of reliable frequent monomers that cover sequences from *Centromeres*. During HOR Inference phase, CA attempts to identify HORs by processing mono(Centromeres).

In addition, CA tries to identify larger repeated structures, named *superHORs*. Finally, CA outputs the set of resulting monomers, the set of HORs and the set of superHORs, and decompositions into monomers and HORs. Figure 3.9 shows a full pipeline of CA and here is a brief description of each step:

- **Monomer inference.** On this step, CA iteratively extends the set of monomers, starting from a set consisting of the *InitialMonomer*. CA uses StringDecomposer to obtain mono(Centromeres) with current Monomers set, clusters alignments of monomers in mono(Centromeres), that have low identity scores using single-linkage clustering, and adds a representative of the largest cluster to the Monomers set. This process ends after 95% of *Centromeres*' length is covered by alignments of *Monomers* with high identity.
- **Hybrid monomer identification.** For each Monomer M with low number of occurrences, CA tries to identify a pair of frequent monomers X,Y that can serve as a source for construction of M as a concatenation of a prefix of X and a suffix of Y.
- **HOR decomposition.** CA considers HOR Inference problem as a data compression problem. The main idea is that if the centromere is a repetitive sequence, canonical HOR (or a set of canonical HORs) is a short subsequence, that reflects a variety of the centromere sequence. In addition, CA tries to build even higher level repeats, called superHORs.

Analysis of CentromereArchitect results on human centromere data demonstrated the importance of monomer order in HOR for Monomer Inference Problem and formulate more rigorous definition of a canonical HOR. Performing monomer clustering procedure, CA considers each monomer alignment separately without neigbouring alignments. In addition, CentromereArchitect defines canonical HOR as the most frequent repetitive substring of the monocentromere without any biological reason. In order to address these issues and to incorporate the CE Postulate we developed the HORmon algorithm.

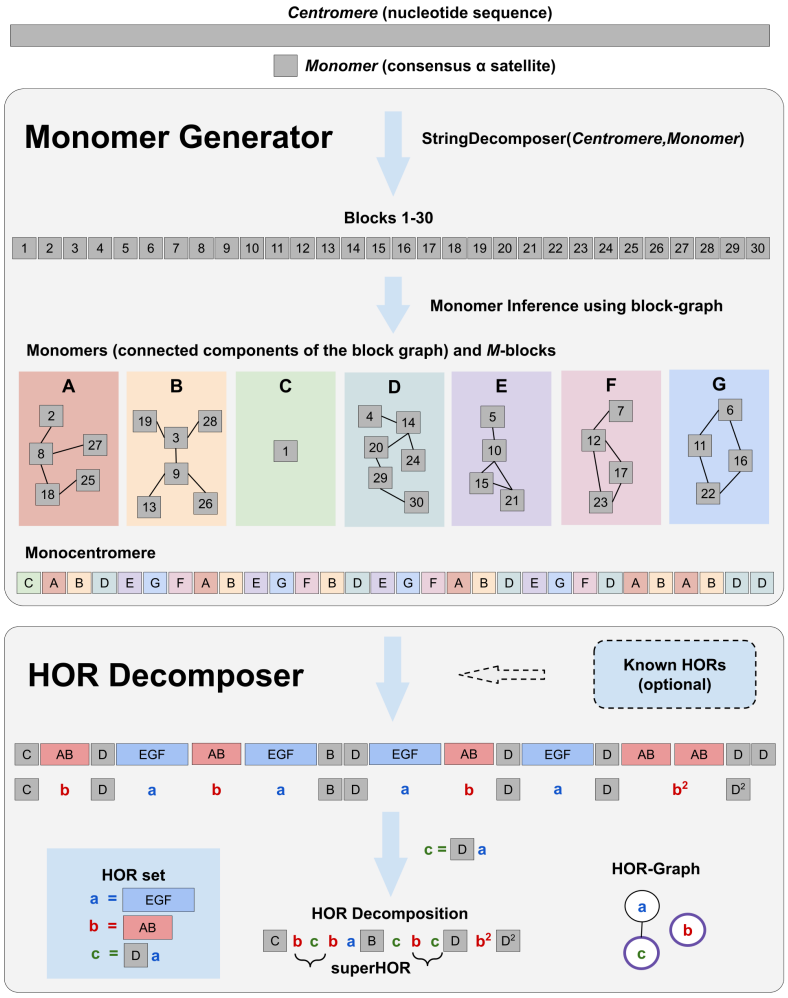

Figure 3.9 — CentromereArchitect pipeline. As an Input, CentromereArchitect takes the nucleotide centromeric sequence *Centromere* and a consensus alpha satellite sequence *InitialMonomer*. CentromereArchitect consists of two modules MonomerGenerator and HORDecomposer. MonomerGenerator constructs the block-graph and extracts each monomer from a connected component of this graph. It further uses the constructed monomer-set to partition the centromere into M-blocks using StringDecomposer. This partitioning transforms centromere into monocentromere. HORDecomposer infers the set of HORs and partitions the monocentromere into these HOR. As an optional input, HORDecomposer can incorporate known HORs in its Output. Finally, HORDecomposer infers superHORs in the resulting HOR decomposition.

3.4.2 HORmon pipeline overview

Similar to CentromereArchitect, HORmon input parameters are a centromeric sequences *Centromeres* and an *InitialMonomer*. HORmon pipeline starts by running the first step of CentromereArchtect and considers monomers generated by CA *CAMonomers* as initial monomers for further processing. HORmon replaces next steps of CentromereArchitect annotation procedure as follows (3.10):

- **Construction of a monomer-graph.** HORmon constructs a directed graph, where each vertex corresponds to one monomer from *CAMonomers*, and weight of an edge from monomer A vertex to monomer B vertex shows how many times B follows A in monocentromere generated on the set of *CAMonomers*. Importantly, that the monomer graph represents a simple De Bruijn graph DBG(Monomers, 2).
- **Monomer set improvement based on positional information.** HORmon performs iterative merge/split operations of monomers based on their mutial position and information from monomer-graph. For example, if monomers X and X' frequently occur between the same two monomers A and B (AXB and AX'B in decomposition) and have high similarity, then probably they represent the same monomer, and HORmon merges monomers X and X' into one. On the other hand, if monomer X usually has two different contexts, i.e., sometimes it occurs between monomers A and B and sometimes it occurs between monomers A' and B', then probably X should be splitted into two different monomers. We will refer to the monomer set after this step as *Monomers*.
- **Hybrid monomer identification.** HORmon considers all monomers from the final set of monomers after split/merge operations as potential hybrid monomers. For each monomer A, HORmon finds a pair of distinct monomers B and C, a concatenation of a prefix and a suffix of which is the most similar to A among all possible concatenations of prefixes and suffixes of other monomers, and considers A as a *hybrid candidate*, if it is similar to concatenation of B and C, but distinct from both B and C. Further HORmon considers as *hybrid monomers* only hybrid candidates, that can be represented as a concatenation of non-hybrid candidates.

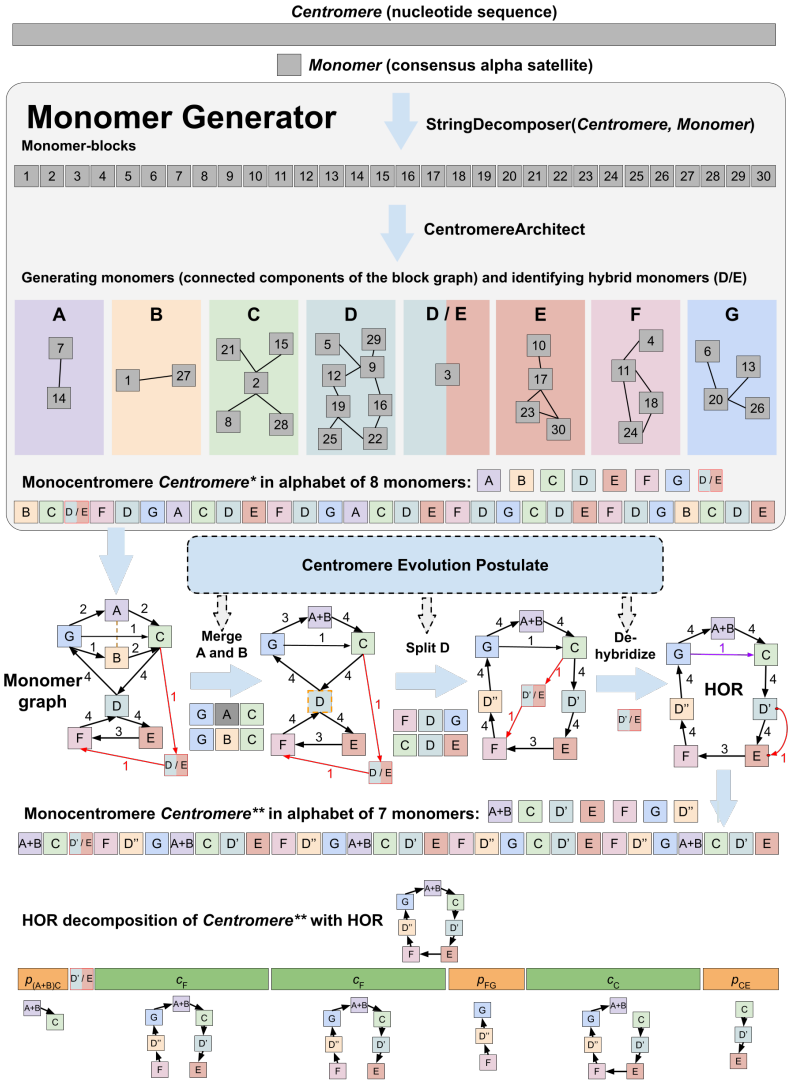

Figure 3.10 — HORmon pipeline. The HORmon algorithm partitions a nucleotide sequence called Centromere into smaller blocks called monomer-blocks using the StringDecomposer [21]. It then uses CentromereArchitect [25] to identify similar blocks, cluster them into monomers, and transform Centromere into a monocentromere referred as Centromere*. The resulting monomer-graph is used to perform split/merge transformations and dehybridizations to comply with the CE Postulate. The decomposition of the centromere sequence in a new monomer-set is referred as Centromere**. The black cycle in the monomer-graph represents the HOR and is used to generate the HOR decomposition of Centromere** into canonical, partial, and auxiliary HORs.

– **Dehybridization.** All detected hybrid monomers are replaced in monomer graph with appropriate edges (see Figure 3.10 dehybridization of monomer D/E) and removed from *Monomers*.
– **HOR inference.** HORmon builds a simplified monomer graph, that consists only of edges that connect monomers of canonical HOR in specific order. Using simplified monomer graph representation, HORmon extracts a set of canonical HORs.
– **HOR decomposition.** After that HORmon decomposes monocentromere into canonical, partial and auxiliary HORs.

3.5 Annotation of human centromeres

This section provides analysis of HORmon v1.0 results on first centromere assemblies from [9]. The exact coordinates of each centromeric sequence are provided in [24] in Supplementary Table S3.1. Identified monomers, HORs, and decompositions of centromeric sequences into monomers and HORs can be found in Supplementary materials of [24].

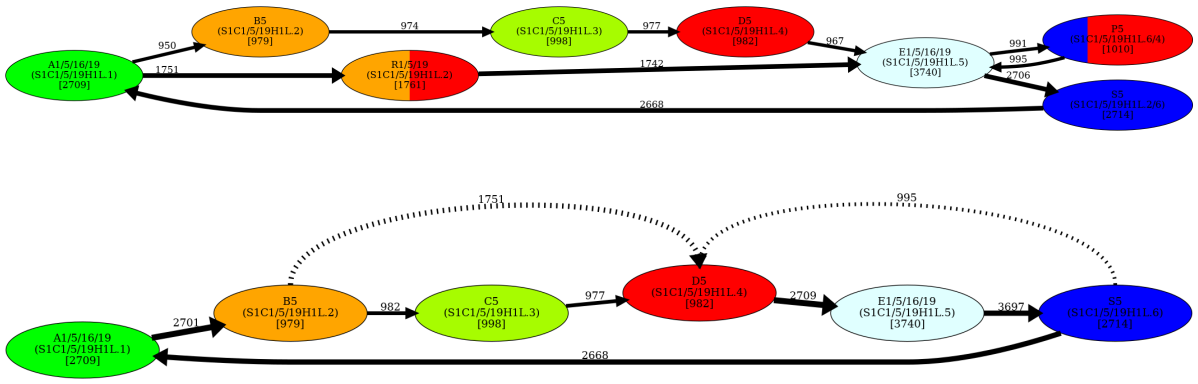

Figure 3.11 — Dehybridization substitutes hybrid vertices (monomers) by hybrid edges in the monomer graph of cen5. Top figure shows initial monomer graph for cen5 and bottom figure shows monomer graph for cen5 after dehybridization of P5 and R1/5/19.

Monomer and canonical HOR identification. At the initial step, HORmon runs CentromereArchitect Monomer Inference module on *all* centromeric sequences to identify monomers presented across different centromeres. As it was previously shown in [13; 25; 115], most centromeres are represented by a set of monomers distinct from a set of monomers of any other centromere, but pairs

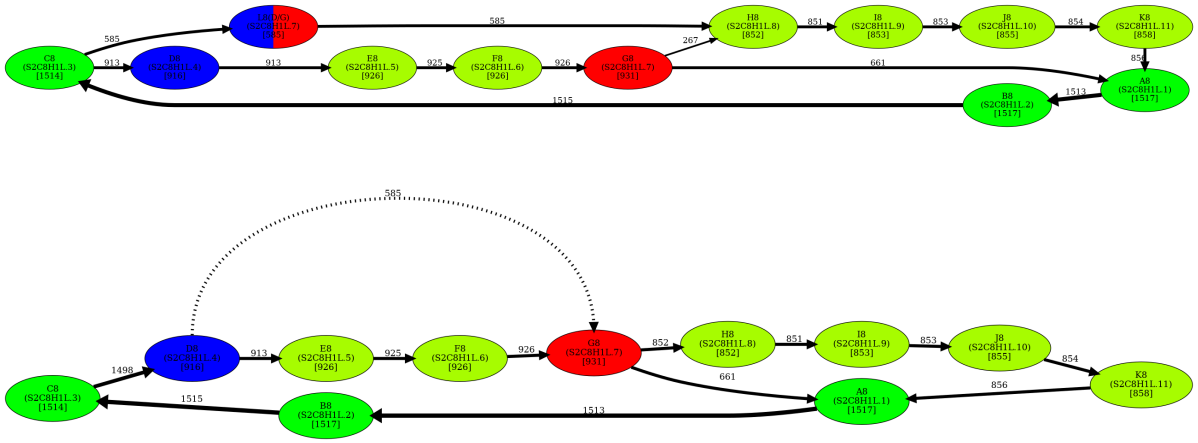

Figure 3.12 — Dehybridization substitutes hybrid vertices (monomers) by hybrid edges in the monomer graph of cen8. Top figure shows initial monomer graph for cen8 and bottom figure shows monomer graph for cen8 after dehybridization of L8 monomer.

of centromeres (chromosome 13 and chromosome 21) and (chromosome 14 and chromosome 22), as well as triple of centromeres (chromosome 1, chromosome 5, and chromosome 19), have been reported to share the same monomers. The results of CentromereArchitect Monomer Inference module are aligned with this monomer distribution across centromeres.

The set of non-hybrid monomers for each centromere, obtained after a full monomer inference step of HORmon pipeline, coincides with a set of monomers, obtained within manual effort in [17] (Supplementary Table S4, [24]). In addition, HORmon found three *frequent* hybrid monomers in centromeres 5 and 8: P5, R1/5/19, and L8. Figures 3.11 and 3.12 shows monomer graphs for centromeres 5 and 8 before and after dehybridization procedure.

Ten monomer-graph structures (related to centromeres 3, 11, 14, 16, 17, 19, 20, 21, 22, X, and Y), generated by HORmon, represent cycles that correspond to canonical HORs (Figure 3.13, Left). Monomer-graph for centromere 17 contains D17Z1 HOR cycle, which represents the main HOR, and D17Z1-B HOR cycle for so-called sister HOR [128]. For all other centromeres the procedure of HOR inference with simplified graph construction helps to reveal canonical HORs.

Revealing canonical HOR in centromere 9 represents a difficult case for HORmon and CE Postulate, as a repetitive sequence of cen9 seems to mainly consist of two different HORs (represented by blue and yellow cycles on Right panel of Figure 3.13). Each of the two HORs contains at least one monomer that is not in the other HOR and can not be represented as hybrid of any other two monomers

according to our current parameter settings. The one monomer found only in the yellow HOR cycle (Right panel of 3.13) differs from the closest Z4/9 monomer by 12 nucleotides, which is quite a lot. Due to the limited data available (so far only one human genome has been fully assembled), it is difficult to reconstruct the real evolutionary scenario and identify one canonical HOR in this case.

Generating the centromere decomposition into HORs. After extracting monomers and HORs, HORmon performed decomposition of each centromere into monomers, and decomposed each resulting monocentromere into canonical, partial, and auxiliary HORs. While canonical HOR is usually represented as a string with a fixed "first" monomer, HORmon considers it as a cycle in monomer-graph with no particular start. Consider canonical HOR H that is represented as a sequence of monomers $M_1, \ldots, M_n$, we collapse each cyclic shift of this sequence $M_i, \ldots, M_n, M_1, \ldots, M_{i-1}$ in monocentromere into symbol c_i. Each partial HOR $M_i, \ldots, M_j$ that represents substring of $M_1, \ldots, M_n$ is denoted as p_{i-j}. Each auxiliary HOR is represented by a monomer capital letter. In order to denote m consecutive occurrences of the same HOR (or *HOR-run* of size m), we use notation ${c_i}^m$ (${p_{i-j}}^m$).

HOR decomposition of centromere X produced by HORmon is shown on Figure 3.14. Again monomer-set is represented by twelve original monomers $AB \ldots KL$. HOR decompositions for all other centromeres can be found in HOR Decomposition File. The process used to generate nucleotide consensus sequences for all monomer and HORs can be found in Supplementary Note 8 of [24]. However, it is not clear whether these consensus sequences are representative for other individuals.

Although some pairs and triplets of centromeres share similar monomers, the analysis revealed that their HORs are different in the CHM13 cell line. For instance, the edit distance between the consensus of HORs in cen13 and cen21 is 20, indicating a 1% divergence, while the edit distance between the consensus of HORs in cen14 and cen22 is much lower at only 3 differences, indicating a 0.2% divergence. In the case of centromeres 1, 5, and 19, the canonical HOR consisted of two non-hybrid monomers according to [115], while our analysis revealed a HOR consisting of six non-hybrid monomers. This discrepancy is likely due to the absence of a complete genome assembly during previous studies. Notably, the edit distance between HORs in cen1 and cen5 is large, with 34 differences or 3.3% divergence.

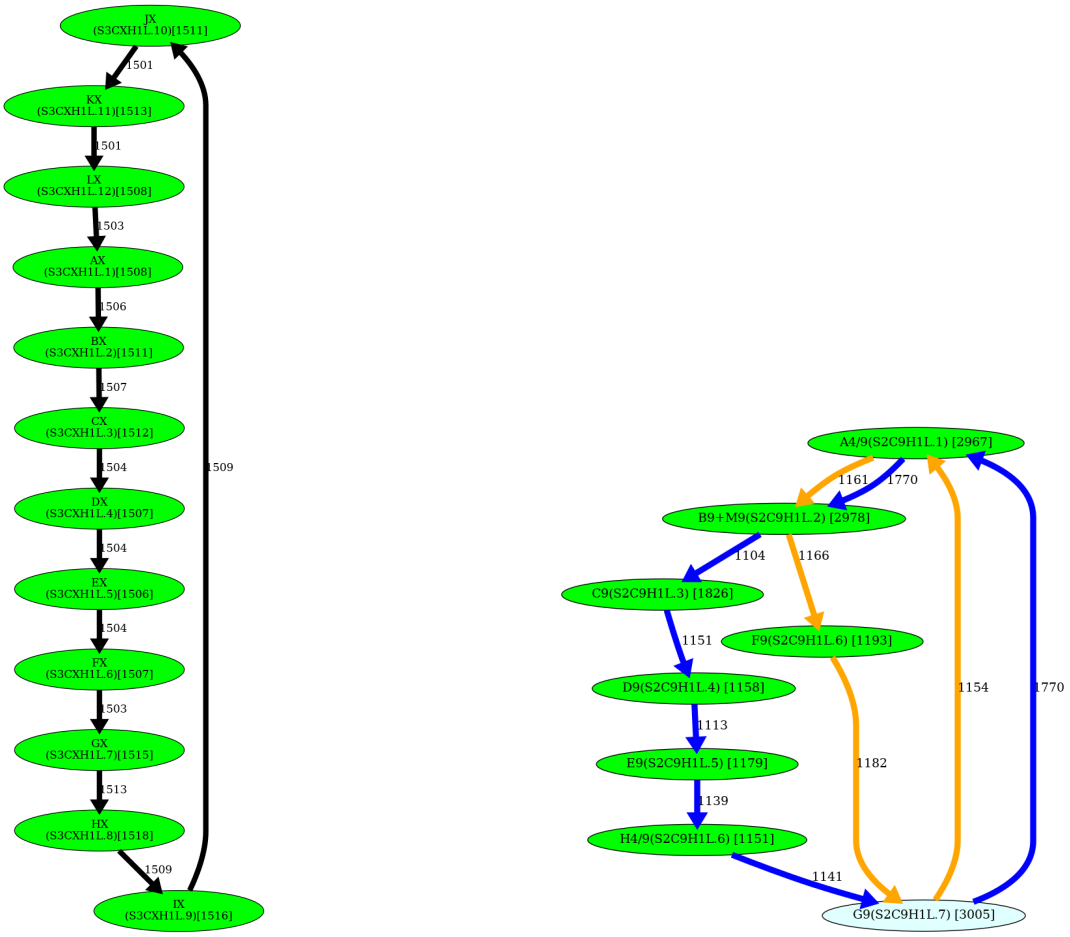

Figure 3.13 — The monomer graphs for cenX (Left) and cen9 (Right). (Left) The simplified monomer graph represents the cycle that corresponds to the canonical high-order repeat (HOR) in cenX. (Right) The manually inferred HOR of cen9 [115] is shown as the blue cycle, which is in conflict with the CE Postulate, because the frequently traversed yellow cycle contains a monomer that does not belong to the blue cycle.

$$p_{10\text{-}12}p_{5\text{-}6}ILp_{2\text{-}3}Ep_{9\text{-}12}p_{2\text{-}5}c_6^{22}p_{7\text{-}12}c_1^{3}p_{2\text{-}6}c_7^{3}p_{8\text{-}6}c_7p_{8\text{-}5}c_6^{95}p_{6\text{-}9}c_{10}^{26}Kc_{12}^{3}\underline{\text{LINE}}p_{1\text{-}7}c_8^{128}p_{11\text{-}7}c_8^{2}p_{11\text{-}7}c_8^{11}p_{11\text{-}7}c_8p_{11\text{-}7}c_8p_{1\text{-}7}c_8p_{11\text{-}7}c_8^{11}Kc_{12}^{8}p_{7\text{-}5}c_6^{2}p_{6\text{-}11}c_{12}p_{7\text{-}10}c_{11}^{50}p_{12\text{-}6}c_7^{174}p_{1\text{-}9}c_{10}^{240}p_{1\text{-}9}c_{10}^{8}p_{12\text{-}9}c_{10}^{18}p_{12\text{-}9}c_{10}^{8}p_{12\text{-}9}c_{10}^{19}p_{12\text{-}9}c_{10}^{8}p_{12\text{-}9}c_{10}^{13}p_{1\text{-}9}c_{10}p_{1\text{-}11}c_{12}^{41}p_{7\text{-}11}c_{12}^{6}p_{7\text{-}11}c_{12}^{5}p_{7\text{-}11}c_{12}^{7}p_{7\text{-}11}c_{12}^{16}p_{7\text{-}11}c_{12}^{4}p_{7\text{-}3}c_4^{13}p_{5\text{-}3}c_4^{21}Ec_6^{14}p_{7\text{-}3}c_4^{52}p_{5\text{-}2}c_3^{42}p_{11\text{-}5}c_6^{57}p_{6\text{-}12}c_1^{4}p_{8\text{-}11}c_{12}^{3}p_{8\text{-}11}c_{12}p_{8\text{-}12}c_1p_{8\text{-}4}c_5^{32}p_{6\text{-}2}c_3p_{5\text{-}1}c_2^{24}p_{3\text{-}4}c_5^{2}p_{6\text{-}4}^{4}p_{6\text{-}12}c_1^{94}p_{2\text{-}12}c_1^{11}p_{2\text{-}7}c_8^{14}p_{9\text{-}7}c_8^{5}p_{9\text{-}7}c_8^{21}p_{11\text{-}4}c_5^{28}p_{6\text{-}2}c_3^{87}Dp_{9\text{-}10}$$

Figure 3.14 — Decomposition of cenX into HORs. The 12-monomer HOR for cenX is represented as $M_1 \dots M_{12} = AB \dots KL$. Each occurrence of the HOR that starts from a particular monomer M_i is denoted by c_i (shown in red), while each occurrence of a partial HOR that includes monomers from i to j is labeled as p_{i-j}. The notation c_i^m (p_{i-j}^m) is used to represent m consecutive occurrences of a canonical (partial) HOR. The most frequent partial monomers p_{3-7}, p_{7-3}, and p_{5-2} in cenX are colored in blue, green, and brown, respectively. The length of the HOR decomposition of cenX is 72, and it includes 1486 complete HORs that form 34 HOR-runs. Only 257 out of 18 089 (1.4%) monomer-blocks in cenX are not covered by complete HORs. The position of the LINE element is shown under the "LINE" entry.

3.6 Conclusions

Recent improvements in long-read sequencing technologies revolutionized human genome assembly and made it possible to perform structural and evolutionary studies of human alpha satellite arrays based on complete centromere assemblies rather than separate reads or Reference Models [23].

StringDecomposer is the first tool designed specifically for decomposing centromere assemblies and reads into monomers. As it was shown, StringDecomposer can accurately convert even error-prone centromeric reads into monoreads and identify non-monomeric regions.

CentromereArchitect and HORmon aim to build full centromere annotations automatically with little prior knowledge of the centromeric monomer set. While CentromereArchitect introduced computationally rigorous definitions of monomers and HORs and provided initial step for monomer inference procedure, HORmon pipeline was developed to output centromere annotation that is consistent with the CE Postulate and previous manual centromere annotations efforts. HORmon not only extracts monomers and HORs in "live" alpha satellite arrays, but also builds their consensus sequences. The HORmon centromere annotations have already contributed significant information to the analysis of CENP-A ChIP-seq enrichment and DNA methylation in satellite arrays [17; 24].

At the same time, HORmon was benchmarked only on "live" centromeres, thus, automatic procedures for annotating other alpha satellite domains are currently not established, and it is hard to predict HORmon results on more variable arrays. The HORmon parameters were chosen using the only fully assembled human genome available at the time, which may have resulted in overfitting. Again due to the lack of complete centromeric assemblies, the CE Postulate was not properly checked versus alternative models of centromere evolution [113; 129; 130].

In addition, as shown in [23; 110], centromere sequences are very diverse across the human population, and we believe that the concept of monomer-graph introduced in [24] will assist in the problem of comparing centromeres across multiple individuals.

Instruments described in this chapter promise to contribute to ongoing centromere studies and accelerate research of human centromeres.

Conclusions

Outcome and impact on the scientific community. Although general methods of genome assembly and analysis of genome assembly provide a lot of valuable information, there are still regions of the genome sequence that are difficult to process. The first two parts of the dissertation describe algorithms for aligning biological sequences to the assembly graph. Incorporating information from the assembly graph and additional data, such algorithms can improve the analysis of regions fragmented in the final assembly. The third part introduced algorithms and tools for the analysis of human centromere sequences. Until recently, centromeres were not assembled into complete sequences, so there were no special tools for their analysis.

SPAligner effectively and accurately aligns biological sequences to assembly graphs produced by SPAdes and other popular short-read assemblers. It helps to incorporate additional information from error-prone long reads to improve assembly quality and from protein databases to identify unassembled protein sequences. In addition, SPAligner was implemented as a part of hybridSPAdes assembler and SPAligner's long reads alignment module improved productivity of hybridSPAdes. ORFograph paper presented an important application of SPAlinger protein search module — search for insecticide toxins directly in assembly graphs rather than in fragmented (usually metagenomic) assemblies. ORFograph pipeline was successfully used to identify potential new IPGs from assembly graphs of all available *B. thuringiensis* Illumina datasets from NCBI.

Instruments for centromere annotation, StringDecomposer, HORmon, and CentromereArchitect, opened new possibilities for analysis of centromere assembly and reads, which was successfully demonstrated during T2T Consortium work [17]. Work on StringDecomposer showed how hybrid monomers can be formed. The CentromereArchitect and HORmon tools have made it possible to create and validate annotations for the first time collected centromere sequences from the human genome. CentromereArchitect and HORmon are fully automated approaches, that allow to perform centromere sequence analysis on a large number of human centromere assemblies.

Further development of this work. Recent improvements of seeding methods and sequence-to-graph alignment algorithms opened a huge potential

to improve both SPAligner and ORFograph implementation [131–133]. At the same time, another important direction of future work is the adaptation of SPAligner methods to align ultralong error-prone ONT reads to assembly graphs constructed from accurate PacBio HiFi reads [134–136]. Recently released Verkko hybrid assembler [137] is able to reconstruct 20 out of 46 diploid chromosomes without gaps by combining accurate HiFi reads and ultralong ONT reads. It uses GraphAligner [49] to perform alignment of ONT reads to assembly graph built from HiFi reads.

Validation of HORmon results on larger number of human centromere assemblies is essential for human centromere analysis. It will help to understand boundaries and potential improvements of CE Postulate and proposed HOR inference procedure. In addition, we foresee application of StringDecomposer, CentromereArchitect, and HORmon to represent first steps to automatic annotation of centromeres not only in humans, but also in other species.

Acknowledgements. The author is grateful to her supervisor, Pavel Pevzner, all the colleagues who participated in joint projects and research — Ivan Alexandrov, Dmitry Antipov, Anton Bankevich, Andrey Bzikadze, Anton Korobeynikov, Olga Kunyavskaya, and Sergey Nurk. A special acknowledgment goes to the colleagues and friends who read the draft of the dissertation and provided useful comments that helped to improve the quality of the text — Dmitry Antipov, Andrey Bzikadze, Tatiana Likhomanenko, Alex Rogozhnikov, and Yana Safonova. The work would not have been possible without the strongest support from my family and friends, especially from my husband Mikhail Dvorkin.

Additional acknowledgments are devoted to the Russian Science Foundation and St. Petersburg State University that provided funding for this work.

List of abbverations and acronyms

AAI	amino acid identity
AMR	antimicrobial resistance
BFS	breadth-first search; an algorithm for traversing a graph
bp	base pair
Bt	Bacillus thuringiensis
CA	CentromereArchitect
CE Postulate	Centromere Evolution Postulate
CDS	coding sequence
Cry	crystal (proteins)
Cyt	cytolic (proteins)
DAG	directed acyclic graph
DFS	depth-first search; an algorithm for traversing a graph
DNA	deoxyribonucleic acid
ETR	extra-long tandem repeat
HMM	hidden Markov model
HOR	high-order repeat (sequence)
IPG	insecticide protein gene
kbp	kilo base pairs
LINE	long interspersed nuclear element
Mbp	mega base pairs
NCBI	National Center for Biotechnology Information
NCRF	Noise-Cancelling Repeat Finder
NGS	next-generation sequencing
ORF	open reading frame
OLC	Overlap-Layout-Consensus
ONT	Oxford Nanopore Technology
PCR	polymerase chain reaction
PB, PacBio	Pacific Biosciences
RAM	random access memory
RNA	ribonucleic acid
SD	StringDecomposer

SD graph	string decomposition graph
SIMD	single instruction, multiple data; type of parallelization
SRA	Short Read Archive
T2T (Consortium)	Telomere-to-Telomere Consortium
TGS	third-generation sequencing
TRF	TandemRepeatsFinder
Vip	vegetative insecticidal protein

Glossary

Alignment — the placement of one sequence under the other in order to find similar areas.

Anchor — regions of high similarity between query sequence and edges of assembly graph in sequence-to-graph alignment problem.

Assembler — a software capable of joining reads into longer accurate sequences called contigs.

Assembly graph — a compacted de Bruijn graph (in this work); can also denote any sequence graph used during the assembly process.

Biopesticides — types of pesticides derived from living organisms or natural products.

Canonical HOR — a cyclic sequence of frequent non-hybrid monomers that has evolved over time into the current centromere sequence.

Centromere — region of the chromosome, where the cell splits its DNA during the cell division process using a special protein complex, *kinetochore*, that is assembled within the same centromere region.

Chromatid — one of the two identical parts of a chromosome, joined at a constricted region of the chromosome called the centromere.

Chromosomome — long DNA molecule that stores a part of a genetic material of an organism.

Codon — a nucleotide triplet that encodes an amino acid.

Compacted de Bruijn graph — a graph constructed from the ordinary de Bruijn graph by replacing all non-branching paths with single edges.

Contig — continuous fragment of DNA or RNA, produced by the assembler.

De Bruijn graph — a graph constructed for one or more strings using all their substrings of length k. A de Bruijn graph for string S and a parameter k is defined as an oriented graph with set of vertices $V = \{S_i^k | i = 1 \dots |S| - k + 1\}$ and set of edges $E = \{(u,v) | u \in V, v \in V, \exists i : u + v_k^1 = S_i^{k+1}\}$.

***De novo* assembly** — a process of joining reads into the initial sequence without any prior information, e.g. from scratch.

Dijkstra algorithm — an algorithm for finding the shortest paths between nodes in a graph.

Edit distance — a scoring scheme that assigns a penalty of 1 for insertion, deletion or replacement of a symbol.

FASTA format — a text format for representing nucleotide or amino acid sequences.

Global sequence alignment – an alignment of two sequences over their entire lengths, so that each symbol of one sequence corresponds to either a symbol of another sequence or a symbol of the gap.

Haplotype — a unique combination of genomic variants inherited together from one parent.

Hidden Markov Model (HMM) — in this work, a statistical model that describes sequences belonging to a specific domain located in genes. HMM can be represented as a directed acyclic graph.

High-order repeat (HOR) — a nucleotide sequence that form a centromere of a specific human chromosome by repeating in it thousands of times.

Hybrid assembly — a process of genome assembly by using several sources of data of different nature (e.g., short and long reads).

Hybrid monomer — a monomer that represents a concatenation of two or more non-hybrid monomers.

Identity — a measure of similarity between two sequences, usually calculated as the difference between the query length (alignment length) and the edit distance divided by the query length (alignment length).

Index — data structure that allows for fast lookup of specific information within a larger set of data.

Insert size — length of the DNA / RNA fragment to be sequenced from both ends and that is "inserted" between the adapters.

Kinetochore — special protein complex responsible for cell division.

K-mer — a sequence of length K.

K-mer coverage — a number of times which a k-mer of interest occurs in all reads. For a longer sequence, it is computed as the average number for all k-mers in this sequence.

Label — a nucleotide sequence corresponding to an edge or a vertex of an assembly graph. The label of the path in an assembly graph is a concatenation of labels of its edges in case, when vertex labels have zero length.

Long reads — reads generated by TGS technologies.

Metagenomics — the study of genetic material recovered directly from environmental samples.

Monomer (alpha-satellites) — a sequence of length 171 that represents a unit of a HOR in a human centromere.

Optimal global (semi-global) alignment — global (semi-global) alignment of a query sequence to a target sequence that reflects the lowest total penalty of converting the query to the target (to a substring of the target) with respect to the given scoring scheme.

Overlap-Layout-Consensus — computational method designed for assembly of long reads (initially Sanger reads).

Paired reads — two reads sequenced from the different ends of the same DNA fragment.

Read — a fragment of DNA, sequence of which is being determined during sequencing process.

Reference-based assembly — a process of determining the exact sequence using known genome or transcriptome sequence of the same species.

Repeat — a fragment that occurs multiple times throughout the genome or other nucleic sequence.

Reverse-complement sequence — a sequence obtained from the initial one using the following operations: each nucleotide is replaced by its complement and the entire sequence is reversed.

Repeat resolution — a process of solving ambiguities during sequence assembly caused by genomic repeats.

Scoring scheme — a set of penalties for each operation to change the query sequence within the alignment problem, for example, such as insertion, deletion or replacement of a symbol. Frequently, scoring scheme also sets a reward for matching symbols in the alignment of two sequences.

Seed — exact k-mers shared between a query and a target sequences during alignment.

Seed-and-extend approach — heuristic approach that perform alignment in two phases. First, it searches for seeds between a query and a target sequences. Second, it extends seeds to connect them into full alignment.

Sequence graph — a graph constructed from one or more strings that represent connections between these strings.

Sequencing — the process of determining the order of nucleotides within DNA molecules.

Semi-global alignment — a global alignment of a query sequence to an arbitrary substring of a target sequence.

Path of optimal semi-global sequence-to-graph alignment — a path with label, that represents the best target for semi-global alignment of a query sequence across all possible paths of the assembly graph.

Short reads — reads generated by NGS technologies.

Start codon — the first codon of a messenger RNA transcript translated by a ribosome.

Stop codon — the codon that signals the termination of the translation process of the current protein.

String graph — sequence graph, where a vertex represents a read sequence and an edge represents an overlap between two reads.

Variation graph — sequence graph that represents genetic variation across several individuals.

References

1. *Peltola Hannu, Söderlund Hans, Ukkonen Esko.* SEQAID: A DNA sequence assembling program based on a mathematical model // *Nucleic Acids Research.* — 1984. — Vol. 12, no. 1Part1. — Pp. 307–321. — [Online; accessed 2022-12-06].

2. *Huang Xiaoqiu.* A contig assembly program based on sensitive detection of fragment overlaps // *Genomics.* — 1992. — 9. — Vol. 14, no. 1. — Pp. 18–25. — [Online; accessed 2022-12-06].

3. *Myers Jr Eugene W.* A history of DNA sequence assembly // *it - Information Technology.* — 2016. — jun 25. — Vol. 58, no. 3. — Pp. 126–132. — [Online; accessed 2022-09-03].

4. hybridSPAdes: An algorithm for hybrid assembly of short and long reads / Dmitry Antipov, Anton Korobeynikov, Jeffrey S. McLean, Pavel A. Pevzner // *Bioinformatics.* — 2015. — nov 20. — Vol. 32, no. 7. — Pp. 1009–1015. — [Online; accessed 2022-09-03].

5. *Salmela Leena, Rivals Eric.* LoRDEC: Accurate and efficient long read error correction // *Bioinformatics.* — 2014. — aug 26. — Vol. 30, no. 24. — Pp. 3506–3514. — [Online; accessed 2022-09-03].

6. Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads / Ryan R. Wick, Louise M. Judd, Claire L. Gorrie, Kathryn E. Holt // *PLOS Computational Biology.* — 2017. — jun 8. — Vol. 13, no. 6. — P. e1005595. — [Online; accessed 2022-09-03].

7. A graph-based approach to diploid genome assembly / Shilpa Garg, Mikko Rautiainen, Adam M Novak et al. // *Bioinformatics.* — 2018. — jun 27. — Vol. 34, no. 13. — Pp. i105–i114. — [Online; accessed 2022-09-03].

8. Initial sequencing and analysis of the human genome / Eric S. Lander, Lauren M. Linton, Bruce Birren et al. // *Nature.* — 2001. — feb 15. — Vol. 409, no. 6822. — Pp. 860–921. — [Online; accessed 2022-09-03].

9. The complete sequence of a human genome / Sergey Nurk, Sergey Koren, Arang Rhie et al. // *Science*. — 2022. — 4. — Vol. 376, no. 6588. — Pp. 44–53. — [Online; accessed 2022-09-20].

10. *HiFi Reads - Highly accurate long-read sequencing*. — https://www.pacb.com/technology/hifi-sequencing/. — 2019. — jul 19. — [Online; accessed 2023-01-14].

11. Nanopore sequencing and assembly of a human genome with ultra-long reads / Miten Jain, Sergey Koren, Karen H Miga et al. // *Nature Biotechnology*. — 2018. — jan 29. — Vol. 36, no. 4. — Pp. 338–345. — [Online; accessed 2023-01-14].

12. *Bzikadze Andrey V., Pevzner Pavel A.* Automated assembly of centromeres from ultra-long error-prone reads // *Nature Biotechnology*. — 2020. — jul 14. — Vol. 38, no. 11. — Pp. 1309–1316. — [Online; accessed 2022-09-20].

13. Classification and monomer-by-monomer annotation dataset of suprachromosomal family 1 alpha satellite higher-order repeats in hg38 human genome assembly / L.I. Uralsky, V.A. Shepelev, A.A. Alexandrov et al. // *Data in Brief*. — 2019. — 6. — Vol. 24. — P. 103708. — [Online; accessed 2022-12-03].

14. Assembling single-cell genomes and mini-metagenomes from chimeric MDA products / Sergey Nurk, Anton Bankevich, Dmitry Antipov et al. // *Journal of Computational Biology*. — 2013. — 10. — Vol. 20, no. 10. — Pp. 714–737. — [Online; accessed 2022-09-03].

15. SPAligner: Alignment of long diverged molecular sequences to assembly graphs / Tatiana Dvorkina, Dmitry Antipov, Anton Korobeynikov, Sergey Nurk // *BMC Bioinformatics*. — 2020. — 7. — Vol. 21, no. S12. — [Online; accessed 2022-08-27].

16. ORFograph: Search for novel insecticidal protein genes in genomic and metagenomic assembly graphs / Tatiana Dvorkina, Anton Bankevich, Alexei Sorokin et al. // *Microbiome*. — 2021. — jun 28. — Vol. 9, no. 1. — [Online; accessed 2022-08-30].

17. Complete genomic and epigenetic maps of human centromeres / Nicolas Altemose, Glennis A. Logsdon, Andrey V. Bzikadze et al. // *Science*. — 2022. — 4. — Vol. 376, no. 6588. — [Online; accessed 2022-09-20].

18. *Li Heng*. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. — https://arxiv.org/abs/1303.3997. — 2013. — mar 16.

19. *Li Heng*. Minimap2: Pairwise alignment for nucleotide sequences // *Bioinformatics*. — 2018. — may 10. — Vol. 34, no. 18. — Pp. 3094–3100. — [Online; accessed 2022-09-03].

20. *Shlemov Alexander, Korobeynikov Anton*. Tech. Rep.: : Cold Spring Harbor Laboratory, 2019. — feb 27. — [Online; accessed 2022-08-27].

21. *Dvorkina Tatiana, Bzikadze Andrey V, Pevzner Pavel A*. The string decomposition problem and its applications to centromere analysis and assembly // *Bioinformatics*. — 2020. — jul 1. — Vol. 36, no. Supplement_1. — Pp. i93–i101. — [Online; accessed 2022-09-20].

22. Identifying periodic occurrences of a template with applications to protein structure / Vincent A. Fischetti, Gad M. Landau, Jeanette P. Schmidt, Peter H. Sellers // Combinatorial Pattern Matching. — Berlin, Heidelberg: Springer Berlin Heidelberg, 1992. — Pp. 111–120. — [Online; accessed 2022-12-03].

23. Centromere reference models for human chromosomes X and Y satellite arrays / Karen H. Miga, Yulia Newton, Miten Jain et al. // *Genome Research*. — 2014. — apr 1. — Vol. 24, no. 4. — Pp. 697–707.

24. Automated annotation of human centromeres with HORmon / Olga Kunyavskaya, Tatiana Dvorkina, Andrey V. Bzikadze et al. // *Genome Research*. — 2022. — may 11. — Vol. 32, no. 6. — Pp. i1137–i1151.

25. CentromereArchitect: Inference and analysis of the architecture of centromeres / Tatiana Dvorkina, Olga Kunyavskaya, Andrey V Bzikadze et al. // *Bioinformatics*. — 2021. — jul 1. — Vol. 37, no. Supplement_1. — Pp. i196–i204. — [Online; accessed 2022-09-20].

26. *Sanger F., Nicklen S., Coulson A. R.* DNA sequencing with chain-terminating inhibitors // *Proceedings of the National Academy of Sciences*. — 1977. — 12. — Vol. 74, no. 12. — Pp. 5463–5467. — [Online; accessed 2022-09-03].

27. Comparison of next-generation sequencing systems / Lin Liu, Yinhu Li, Siliang Li et al. // *Journal of Biomedicine and Biotechnology*. — 2012. — Vol. 2012. — Pp. 1–11. — [Online; accessed 2022-09-03].

28. Sanger confirmation is required to achieve optimal sensitivity and specificity in next-generation sequencing panel testing / Wenbo Mu, Hsiao-Mei Lu, Jefferey Chen et al. // *The Journal of Molecular Diagnostics*. — 2016. — 11. — Vol. 18, no. 6. — Pp. 923–932. — [Online; accessed 2022-09-03].

29. Long walk to genomics: History and current approaches to genome sequencing and assembly / Alice Maria Giani, Guido Roberto Gallo, Luca Gianfranceschi, Giulio Formenti // *Computational and Structural Biotechnology Journal*. — 2020. — Vol. 18. — Pp. 9–19. — [Online; accessed 2022-09-03].

30. Real-Time DNA sequencing from single polymerase molecules / John Eid, Adrian Fehr, Jeremy Gray et al. // *Science*. — 2009. — jan 2. — Vol. 323, no. 5910. — Pp. 133–138. — [Online; accessed 2023-01-14].

31. The Oxford Nanopore MinION: Delivery of nanopore sequencing to the genomics community / Miten Jain, Hugh E. Olsen, Benedict Paten, Mark Akeson // *Genome Biology*. — 2016. — nov 25. — Vol. 17, no. 1. — [Online; accessed 2023-01-14].

32. *Pop M.* Genome assembly reborn: Recent computational challenges // *Briefings in Bioinformatics*. — 2009. — may 29. — Vol. 10, no. 4. — Pp. 354–366. — [Online; accessed 2022-09-03].

33. *Myers Eugene W.* Toward simplifying and accurately formulating fragment assembly // *Journal of Computational Biology*. — 1995. — 1. — Vol. 2, no. 2. — Pp. 275–290. — [Online; accessed 2022-09-03].

34. *Pevzner Pavel A., Tang Haixu, Waterman Michael S.* An Eulerian path approach to DNA fragment assembly // *Proceedings of the National Academy of Sciences*. — 2001. — aug 14. — Vol. 98, no. 17. — Pp. 9748–9753. — [Online; accessed 2022-09-03].

35. *Zerbino Daniel R., Birney Ewan.* Velvet: Algorithms for de novo short read assembly using de Bruijn graphs // *Genome Research.* — 2008. — mar 18. — Vol. 18, no. 5. — Pp. 821–829. — [Online; accessed 2022-08-30].

36. ABySS: A parallel assembler for short read sequence data / Jared T. Simpson, Kim Wong, Shaun D. Jackman et al. // *Genome Research.* — 2009. — feb 27. — Vol. 19, no. 6. — Pp. 1117–1123. — [Online; accessed 2023-01-21].

37. SOAPdenovo2: An empirically improved memory-efficient short-read de novo assembler / Ruibang Luo, Binghang Liu, Yinlong Xie et al. // *GigaScience.* — 2012. — 12. — Vol. 1, no. 1. — [Online; accessed 2023-01-21].

38. SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing / Anton Bankevich, Sergey Nurk, Dmitry Antipov et al. // *Journal of Computational Biology.* — 2012. — 5. — Vol. 19, no. 5. — Pp. 455–477. — [Online; accessed 2022-08-27].

39. *Needleman Saul B., Wunsch Christian D.* A general method applicable to the search for similarities in the amino acid sequence of two proteins // *Journal of Molecular Biology.* — 1970. — 3. — Vol. 48, no. 3. — Pp. 443–453. — [Online; accessed 2022-09-03].

40. *Langmead Ben, Salzberg Steven L.* Fast gapped-read alignment with Bowtie 2 // *Nature Methods.* — 2012. — mar 4. — Vol. 9, no. 4. — Pp. 357–359. — [Online; accessed 2022-09-03].

41. *Chikhi Rayan, Rizk Guillaume.* Space-efficient and exact de Bruijn graph representation based on a Bloom filter // *Algorithms for Molecular Biology.* — 2013. — 1. — Vol. 8, no. 1. — [Online; accessed 2022-08-27].

42. MEGAHIT: An ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph / Dinghua Li, Chi-Man Liu, Ruibang Luo et al. // *Bioinformatics.* — 2015. — jan 20. — Vol. 31, no. 10. — Pp. 1674–1676. — [Online; accessed 2022-08-27].

43. Variation graph toolkit improves read mapping by representing genetic variation in the reference / Erik Garrison, Jouni Sirén, Adam M Novak et al. // *Nature Biotechnology.* — 2018. — 10. — Vol. 36, no. 9. — Pp. 875–879. — [Online; accessed 2022-09-03].

44. BrownieAligner: Accurate alignment of Illumina sequencing data to de Bruijn graphs / Mahdi Heydari, Giles Miclotte, Yves Van de Peer, Jan Fostier // *BMC Bioinformatics*. — 2018. — sep 4. — Vol. 19, no. 1. — [Online; accessed 2022-09-03].

45. On the complexity of sequence to graph alignment / Chirag Jain, Haowen Zhang, Yu Gao, Srinivas Aluru // Lecture Notes in Computer Science. — Cham: Springer International Publishing, 2019. — Pp. 85–100. — [Online; accessed 2022-09-03].

46. Sequence alignment on directed graphs / Vaddadi Naga Sai Kavya, Kshitij Tayal, Rajgopal Srinivasan, Naveen Sivadasan // *Journal of Computational Biology*. — 2019. — 1. — Vol. 26, no. 1. — Pp. 53–67. — [Online; accessed 2022-09-03].

47. Read mapping on de Bruijn graphs / Antoine Limasset, Bastien Cazaux, Eric Rivals, Pierre Peterlongo // *BMC Bioinformatics*. — 2016. — jun 16. — Vol. 17, no. 1. — [Online; accessed 2022-09-03].

48. *Sirén Jouni*. Indexing variation graphs // 2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX) / Society for Industrial and Applied Mathematics. — Philadelphia, PA: 2017. — 1. — [Online; accessed 2022-09-03].

49. *Rautiainen Mikko, Marschall Tobias*. GraphAligner: Rapid and versatile sequence-to-graph alignment // *Genome Biology*. — 2020. — sep 24. — Vol. 21, no. 1. — [Online; accessed 2022-09-03].

50. MUMmer4: A fast and versatile genome alignment system / Guillaume Marçais, Arthur L. Delcher, Adam M. Phillippy et al. // *PLOS Computational Biology*. — 2018. — jan 26. — Vol. 14, no. 1. — P. e1005944. — [Online; accessed 2022-09-03].

51. *Rautiainen Mikko, Mäkinen Veli, Marschall Tobias*. Bit-parallel sequence-to-graph alignment // *Bioinformatics*. — 2019. — mar 9. — Vol. 35, no. 19. — Pp. 3599–3607. — [Online; accessed 2022-09-03].

52. *Li, Feng, Chu.* The design and construction of reference pangenome graphs with minigraph // *Genome Biology.* — 2020. — oct 16. — Vol. 21, no. 1. — Pp. 1–19.

53. MegaGTA: A sensitive and accurate metagenomic gene-targeted assembler using iterative de Bruijn graphs / Dinghua Li, Yukun Huang, Chi-Ming Leung et al. // *BMC Bioinformatics.* — 2017. — 10. — Vol. 18, no. S12. — [Online; accessed 2022-09-03].

54. *Myers Eugene W.* AnO(ND) difference algorithm and its variations // *Algorithmica.* — 1986. — 11. — Vol. 1, no. 1-4. — Pp. 251–266. — [Online; accessed 2022-09-04].

55. *Amir Amihood, Lewenstein Moshe, Lewenstein Noa.* Pattern matching in hypertext // *Journal of Algorithms.* — 2000. — 4. — Vol. 35, no. 1. — Pp. 82–99. — [Online; accessed 2022-09-04].

56. *Gotoh Osamu.* An improved algorithm for matching biological sequences // *Journal of Molecular Biology.* — 1982. — 12. — Vol. 162, no. 3. — Pp. 705–708. — [Online; accessed 2022-09-04].

57. *Navarro Gonzalo.* A guided tour to approximate string matching // *ACM Computing Surveys.* — 2001. — 3. — Vol. 33, no. 1. — Pp. 31–88. — [Online; accessed 2022-09-04].

58. *Šošić Martin, Šikić Mile.* Edlib: A C/C ++ library for fast, exact sequence alignment using edit distance // *Bioinformatics.* — 2017. — jan 31. — Vol. 33, no. 9. — Pp. 1394–1395. — [Online; accessed 2022-09-04].

59. *Daily Jeff.* Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments // *BMC Bioinformatics.* — 2016. — feb 10. — Vol. 17, no. 1. — [Online; accessed 2022-09-05].

60. *Pearson William R.* Selecting the right similarityscoring matrix // *Current Protocols in Bioinformatics.* — 2013. — 10. — Vol. 43, no. 1. — [Online; accessed 2022-09-05].

61. MinION-based long-read sequencing and assembly extends the *Caenorhabditis elegans* reference genome / John R Tyson, Nigel J O'Neil, Miten Jain et al. // *Genome research.* — 2018. — 2. — Vol. 28, no. 2. — Pp. 266–274.

62. *Nagarajan Niranjan, Pop Mihai.* Sequence assembly demystified // *Nature Reviews Genetics*. — 2013. — jan 29. — Vol. 14, no. 3. — Pp. 157–167. — [Online; accessed 2022-09-11].

63. Genome-resolved metagenomics identifies genetic mobility, metabolic interactions, and unexpected diversity in perchlorate-reducing communities / Tyler P. Barnum, Israel A. Figueroa, Charlotte I. Carlström et al. // *The ISME Journal*. — 2018. — feb 23. — Vol. 12, no. 6. — Pp. 1568–1581. — [Online; accessed 2022-09-11].

64. Accurate, multi-kb reads resolve complex populations and detect rare microorganisms / Itai Sharon, Michael Kertesz, Laura A. Hug et al. // *Genome Research*. — 2015. — feb 9. — Vol. 25, no. 4. — Pp. 534–543. — [Online; accessed 2022-09-11].

65. Characterization of metagenomes in urban aquatic compartments reveals high prevalence of clinically relevant antibiotic resistance genes in wastewaters / Charmaine Ng, Martin Tay, Boonfei Tan et al. // *Frontiers in Microbiology*. — 2017. — nov 16. — Vol. 8. — [Online; accessed 2022-09-11].

66. Tech. Rep.: / Michael Feldgarden, Vyacheslav Brover, Daniel H. Haft et al.: Cold Spring Harbor Laboratory, 2019. — feb 15. — [Online; accessed 2022-09-11].

67. Basic local alignment search tool / Stephen F. Altschul, Warren Gish, Webb Miller et al. // *Journal of Molecular Biology*. — 1990. — 10. — Vol. 215, no. 3. — Pp. 403–410. — [Online; accessed 2022-09-11].

68. Bandage: Interactive visualization ofde novogenome assemblies: Fig. 1. / Ryan R. Wick, Mark B. Schultz, Justin Zobel, Kathryn E. Holt // *Bioinformatics*. — 2015. — jun 22. — Vol. 31, no. 20. — Pp. 3350–3352. — [Online; accessed 2022-08-30].

69. What are Biopesticides? — https://www.epa.gov/ingredients-used-pesticide-products/what-are-biopesticides. — 2015. — aug 31. — [Online; accessed 2022-08-23].

70. Molecular and Insecticidal Characterization of a Novel Cry-Related Protein from Bacillus Thuringiensis Toxic against Myzus persicae / Leopoldo Palma, Delia Muñoz, Colin Berry et al. // *Toxins*. — 2014. — nov 1. — Vol. 6, no. 11.

71. Genetically engineered crops help support conservation biological control / Jörg Romeis, Steven E. Naranjo, Michael Meissle, Anthony M. Shelton // *Biological Control*. — 2019. — 3. — Vol. 130. — Pp. 136–154. — [Online; accessed 2022-08-23].

72. *Ohba Michio, Mizuki Eiichi, Uemori Akiko*. Parasporin, a New Anticancer Protein Group from Bacillus thuringiensis // *Anticancer Research*. — 2009. — jan 1. — Vol. 29, no. 1. — Pp. 427–433.

73. Field-evolved resistance by western corn rootworm to multiple Bacillus thuringiensis toxins in transgenic maize / Aaron J. Gassmann, Jennifer L. Petzold-Maxwell, Eric H. Clifton et al. // *Proceedings of the National Academy of Sciences*. — 2014. — mar 17. — Vol. 111, no. 14. — Pp. 5141–5146. — [Online; accessed 2022-08-23].

74. *Höfte H, Whiteley H R*. Insecticidal crystal proteins of Bacillus thuringiensis // *Microbiological Reviews*. — 1989. — 6. — Vol. 53, no. 2. — Pp. 242–255. — [Online; accessed 2022-08-23].

75. Prediction of insecticidal activity of Bacillus thuringiensis strains by polymerase chain reaction product profiles / N B Carozzi, V C Kramer, G W Warren et al. // *Applied and Environmental Microbiology*. — 1991. — 11. — Vol. 57, no. 11. — Pp. 3057–3061. — [Online; accessed 2022-08-24].

76. Screening and identification ofvipgenes inBacillus thuringiensisstrains / C.S. Hernández-Rodríguez, A. Boets, J. Van Rie, J. Ferré // *Journal of Applied Microbiology*. — 2009. — 7. — Vol. 107, no. 1. — Pp. 219–225. — [Online; accessed 2022-08-24].

77. *Juárez-Pérez V M, Ferrandis M D, Frutos R*. PCR-based approach for detection of novel Bacillus thuringiensis cry genes // *Applied and Environmental Microbiology*. — 1997. — 8. — Vol. 63, no. 8. — Pp. 2997–3002. — [Online; accessed 2022-08-24].

78. *Lin Yi, Fang Guangwei, Peng Kun.* Characterization of the highly variable cry gene regions of Bacillus thuringiensis strain ly4a3 by PCR-SSCP profiling and sequencing // *Biotechnology Letters.* — 2006. — dec 7. — Vol. 29, no. 2. — Pp. 247–251. — [Online; accessed 2022-08-24].

79. *Noguera Pedro A., Ibarra Jorge E.* Detection of New cry Genes of Bacillus thuringiensis by Use of a Novel PCR Primer System // *Applied and Environmental Microbiology.* — 2010. — sep 15. — Vol. 76, no. 18. — Pp. 6150–6155. — [Online; accessed 2022-08-24].

80. *Besemer J., Borodovsky M.* GeneMark: Web software for gene finding in prokaryotes, eukaryotes and viruses // *Nucleic Acids Research.* — 2005. — jul 1. — Vol. 33. — Pp. W451–W454. — [Online; accessed 2022-08-25].

81. Prodigal: Prokaryotic gene recognition and translation initiation site identification / Doug Hyatt, Gwo-Liang Chen, Philip F LoCascio et al. // *BMC Bioinformatics.* — 2010. — mar 8. — Vol. 11, no. 1. — [Online; accessed 2022-08-25].

82. Identifying bacterial genes and endosymbiont DNA with Glimmer / Arthur L. Delcher, Kirsten A. Bratke, Edwin C. Powers, Steven L. Salzberg // *Bioinformatics.* — 2007. — jan 19. — Vol. 23, no. 6. — Pp. 673–679. — [Online; accessed 2022-08-25].

83. *Zhu Wenhan, Lomsadze Alexandre, Borodovsky Mark.* Ab initio gene identification in metagenomic sequences // *Nucleic Acids Research.* — 2010. — apr 19. — Vol. 38, no. 12. — Pp. e132–e132. — [Online; accessed 2022-08-25].

84. Gene and translation initiation site prediction in metagenomic sequences / Doug Hyatt, Philip F. LoCascio, Loren J. Hauser, Edward C. Uberbacher // *Bioinformatics.* — 2012. — jul 12. — Vol. 28, no. 17. — Pp. 2223–2230. — [Online; accessed 2022-08-25].

85. Gene prediction with Glimmer for metagenomic sequences augmented by classification and clustering / David R. Kelley, Bo Liu, Arthur L. Delcher et al. // *Nucleic Acids Research.* — 2011. — nov 17. — Vol. 40, no. 1. — Pp. e9–e9. — [Online; accessed 2022-08-25].

86. BtToxin_Digger: A comprehensive and high-throughput pipeline for mining toxin protein genes from Bacillus thuringiensis / Hualin Liu, Jinshui Zheng, Dexin Bo et al. // *Bioinformatics*. — 2021. — jul 9. — Vol. 38, no. 1. — Pp. 250–251. — [Online; accessed 2022-08-24].

87. UniProt: The universal protein knowledgebase in 2021 / Alex Bateman, Maria-Jesus Martin, Sandra Orchard et al. // *Nucleic Acids Research*. — 2020. — nov 25. — Vol. 49, no. D1. — Pp. D480–D489. — [Online; accessed 2022-08-27].

88. Introduction to algorithms / Thomas H.. Cormen, Thomas H Cormen, Charles E Leiserson et al. — MIT Press, 2001. — [Online; accessed 2022-08-27].

89. *Eddy S. R.* Profile hidden Markov models // *Bioinformatics*. — 1998. — oct 1. — Vol. 14, no. 9. — Pp. 755–763. — [Online; accessed 2022-08-27].

90. A structure-based nomenclature for Bacillus thuringiensis and other bacteria-derived pesticidal proteins / Neil Crickmore, Colin Berry, Suresh Panneerselvam et al. // *Journal of Invertebrate Pathology*. — 2021. — 11. — Vol. 186. — P. 107438. — [Online; accessed 2022-08-29].

91. The Pfam protein families database in 2019 / Sara El-Gebali, Jaina Mistry, Alex Bateman et al. // *Nucleic Acids Research*. — 2018. — oct 24. — Vol. 47, no. D1. — Pp. D427–D432. — [Online; accessed 2022-08-29].

92. metaSPAdes: A new versatile metagenomic assembler / Sergey Nurk, Dmitry Meleshko, Anton Korobeynikov, Pavel A. Pevzner // *Genome Research*. — 2017. — mar 15. — Vol. 27, no. 5. — Pp. 824–834. — [Online; accessed 2022-08-30].

93. Plasmid detection and assembly in genomic and metagenomic data sets / Dmitry Antipov, Mikhail Raiko, Alla Lapidus, Pavel A. Pevzner // *Genome Research*. — 2019. — may 2. — Vol. 29, no. 6. — Pp. 961–968. — [Online; accessed 2022-08-30].

94. No more tears: Mining sequencing data for novel bt cry toxins with cryprocessor / Anton E. Shikov, Yury V. Malovichko, Rostislav K. Skitchenko et al. // *Toxins*. — 2020. — mar 23. — Vol. 12, no. 3. — P. 204. — [Online; accessed 2022-08-30].

95. Critical Assessment of Metagenome Interpretation—a benchmark of metagenomics software / Alexander Sczyrba, Peter Hofmann, Peter Belmann et al. // *Nature Methods*. — 2017. — oct 2. — Vol. 14, no. 11. — Pp. 1063–1071. — [Online; accessed 2022-08-29].

96. ART: A next-generation sequencing read simulator / Weichun Huang, Leping Li, Jason R. Myers, Gabor T. Marth // *Bioinformatics*. — 2011. — dec 23. — Vol. 28, no. 4. — Pp. 593–594. — [Online; accessed 2022-08-29].

97. *Buchfink Benjamin, Xie Chao, Huson Daniel H.* Fast and sensitive protein alignment using DIAMOND // *Nature Methods*. — 2014. — nov 17. — Vol. 12, no. 1. — Pp. 59–60. — [Online; accessed 2022-08-29].

98. Revision of the Nomenclature for the Bacillus thuringiensis Pesticidal Crystal Proteins / N. Crickmore, D. R. Zeigler, J. Feitelson et al. // *Microbiology and Molecular Biology Reviews*. — 1998. — 9. — Vol. 62, no. 3. — Pp. 807–813. — [Online; accessed 2022-08-29].

99. *Edgar*. MUSCLE: A multiple sequence alignment method with reduced time and space complexity // *BMC Bioinformatics*. — 2004. — aug 19. — Vol. 5, no. 1. — Pp. 1–19.

100. *Price Morgan N., Dehal Paramvir S., Arkin Adam P.* FastTree 2 – approximately maximum-likelihood trees for large alignments // *PLoS ONE*. — 2010. — mar 10. — Vol. 5, no. 3. — P. e9490. — [Online; accessed 2022-08-30].

101. Lineage-specific plasmid acquisition and the evolution of specialized pathogens inBacillus thuringiensisand theBacillus cereusgroup / Guillaume Méric, Leonardos Mageiros, Ben Pascoe et al. // *Molecular Ecology*. — 2018. — 4. — Vol. 27, no. 7. — Pp. 1524–1540. — [Online; accessed 2022-08-29].

102. *Coil David, Jospin Guillaume, Darling Aaron E.* A5-miseq: An updated pipeline to assemble microbial genomes from Illumina MiSeq data // *Bioinformatics*. — 2014. — oct 22. — Vol. 31, no. 4. — Pp. 587–589. — [Online; accessed 2022-08-29].

103. *Jeong Haeyoung, Choi Soo-Keun, Park Seung-Hwan.* Genome Sequences of Bacillus thuringiensis Serovar kurstaki Strain BP865 and B. thuringiensis

Serovar aizawai Strain HD-133 // *Genome Announcements*. — 2017. — feb 2. — Vol. 5, no. 5. — [Online; accessed 2022-08-29].

104. *Seemann T.* Prokka: Rapid prokaryotic genome annotation // *Bioinformatics*. — 2014. — mar 18. — Vol. 30, no. 14. — Pp. 2068–2069. — [Online; accessed 2022-08-29].

105. NCBI prokaryotic genome annotation pipeline / Tatiana Tatusova, Michael DiCuccio, Azat Badretdin et al. // *Nucleic Acids Research*. — 2016. — jun 24. — Vol. 44, no. 14. — Pp. 6614–6624. — [Online; accessed 2023-02-20].

106. Geospatial resolution of human and bacterial diversity with city-scale metagenomics / Ebrahim Afshinnekoo, Cem Meydan, Shanin Chowdhury et al. // *Cell Systems*. — 2015. — 7. — Vol. 1, no. 1. — Pp. 72–87. — [Online; accessed 2022-08-30].

107. Recovery of nearly 8,000 metagenome-assembled genomes substantially expands the tree of life / Donovan H. Parks, Christian Rinke, Maria Chuvochina et al. // *Nature Microbiology*. — 2017. — sep 11. — Vol. 2, no. 11. — Pp. 1533–1542. — [Online; accessed 2022-08-30].

108. *McKinley Kara L., Cheeseman Iain M.* The molecular basis for centromere identity and function // *Nature Reviews Molecular Cell Biology*. — 2015. — nov 25. — Vol. 17, no. 1. — Pp. 16–29. — [Online; accessed 2022-09-26].

109. Sequences associated with centromere competency in the human genome / Karen E. Hayden, Erin D. Strome, Stephanie L. Merrett et al. // *Molecular and Cellular Biology*. — 2013. — feb 15. — Vol. 33, no. 4. — Pp. 763–772. — [Online; accessed 2022-09-20].

110. *Suzuki Yuta, Myers Eugene W., Morishita Shinichi.* Rapid and ongoing evolution of repetitive sequence structures in human centromeres // *Science Advances*. — 2020. — dec 11. — Vol. 6, no. 50. — [Online; accessed 2022-09-20].

111. *Smith George P.* Evolution of repeated DNA sequences by unequal crossover // *Science*. — 1976. — feb 13. — Vol. 191, no. 4227. — Pp. 528–535. — [Online; accessed 2023-03-11].

112. *Malik Harmit S., Henikoff Steven.* Major evolutionary transitions in centromere complexity // *Cell.* — 2009. — 9. — Vol. 138, no. 6. — Pp. 1067–1082. — [Online; accessed 2023-03-11].

113. *Rice William R.* Tech. Rep.: : Cold Spring Harbor Laboratory, 2019. — aug 10. — [Online; accessed 2022-12-03].

114. *Waye J S, Willard H F.* Chromosome-specific alpha satellite DNA: Nucleotide sequence analysis of the 2.0 kilobasepair repeat from the human X chromosome. // *Nucleic Acids Research.* — 1985. — apr 25. — Vol. 13, no. 8.

115. *McNulty Shannon M, Sullivan Beth A.* Alpha satellite DNA biology: Finding function in the recesses of the genome // *Chromosome research : an international journal on the molecular, supramolecular and evolutionary aspects of chromosome biology.* — 2018. — 9. — Vol. 26, no. 3. — Pp. 115–138.

116. Telomere-to-telomere assembly of a complete human X chromosome / Miga, Koren, Rhie et al. // *Nature.* — 2020. — jul 14. — Vol. 585, no. 7823. — Pp. 79–84.

117. *Harris Robert S, Cechova Monika, Makova Kateryna D.* Noise-cancelling repeat finder: Uncovering tandem repeats in error-prone long-read sequencing data // *Bioinformatics.* — 2019. — jul 10. — Vol. 35, no. 22. — Pp. 4809–4811. — [Online; accessed 2022-12-03].

118. TandemTools: Mapping long reads and assessing/improving assembly quality in extra-long tandem repeats / Mikheenko, Bzikadze, Gurevich et al. // *Bioinformatics.* — 2020. — jul 13. — Vol. 36, no. Supplement_1. — Pp. i75–i83.

119. *Benson G.* Tandem repeats finder: A program to analyze DNA sequences // *Nucleic Acids Research.* — 1999. — jan 1. — Vol. 27, no. 2. — Pp. 573–580. — [Online; accessed 2022-12-03].

120. Alpha-CENTAURI: Assessing novel centromeric repeat sequence variation with long read sequencing / Volkan Sevim, Ali Bashir, Chen-Shan Chin, Karen H. Miga // *Bioinformatics.* — 2016. — feb 24. — Vol. 32, no. 13. — Pp. 1921–1924. — [Online; accessed 2022-12-03].

121. Interspersed repeats are found predominantly in the “old“ alpha-satellite families / Alexei E Kazakov, Valery A Shepelev, Irina G Tumeneva et al. // *Genomics*. — 2003. — Vol. 82, no. 6. — Pp. 619–627. — URL: `https://www.sciencedirect.com/science/article/pii/S0888754303001824`.

122. ColorHOR–novel graphical algorithm for fast scan of alpha satellite higher-order repeats and HOR annotation for GenBank sequence of human genome / V. Paar, N. Pavin, M. Rosandic et al. // *Bioinformatics*. — 2004. — oct 27. — Vol. 21, no. 7. — Pp. 846–852. — [Online; accessed 2022-12-03].

123. Organization and evolution of primate centromeric DNA from whole-genome shotgun sequence data / Can Alkan, Mario Ventura, Nicoletta Archidiacono et al. // *PLoS Computational Biology*. — 2007. — sep 28. — Vol. 3, no. 9. — P. e181. — [Online; accessed 2022-12-03].

124. *Compeau Phillip, Pevzner Pavel*. Bioinformatics algorithms: An active learning approach. — 1986. — 6. — [Online; accessed 2022-12-03].

125. *Matroud Atheer A., Tuffley Christopher P., Hendy Michael D.* An algorithm to solve the motif alignment problem for approximate nested tandem repeats in biological sequences // *Journal of Computational Biology*. — 2011. — 9. — Vol. 18, no. 9. — Pp. 1211–1218. — [Online; accessed 2022-12-03].

126. *Matroud A. A., Hendy M. D., Tuffley C. P.* NTRFinder: A software tool to find nested tandem repeats // *Nucleic Acids Research*. — 2011. — nov 25. — Vol. 40, no. 3. — Pp. e17–e17. — [Online; accessed 2022-12-03].

127. *Gusfield Dan*. Algorithms on strings, trees, and sequences: Computer science and computational biology. — Cambridge University Press, 1997. — may 28. — [Online; accessed 2022-12-03].

128. *Miga Karen H., Alexandrov Ivan A.* Variation and evolution of human centromeres: A field guide and perspective // *Annual Review of Genetics*. — 2021. — nov 23. — Vol. 55, no. 1. — Pp. 583–602. — [Online; accessed 2022-12-03].

129. Evolution of satellite DNAs from the genus Palorus–experimental evidence for the "library"hypothesis / N. Mestrovic, M. Plohl, B. Mravinac, D. Ugarkovic // *Molecular Biology and Evolution*. — 1998. — aug 1. — Vol. 15, no. 8. — Pp. 1062–1068. — [Online; accessed 2022-12-03].

130. *Henikoff Steven, Ahmad Kami, Malik Harmit S.* The centromere paradox: Stable inheritance with rapidly evolving DNA // *Science*. — 2001. — aug 10. — Vol. 293, no. 5532. — Pp. 1098–1102. — [Online; accessed 2022-12-03].

131. *Ivanov Pesho, Bichsel Benjamin, Vechev Martin.* Tech. Rep.: : Cold Spring Harbor Laboratory, 2021. — nov 8. — [Online; accessed 2022-12-10].

132. Tech. Rep.: / Jun Ma, Manuel Cáceres, Leena Salmela et al.: Cold Spring Harbor Laboratory, 2022. — jan 7. — [Online; accessed 2022-12-10].

133. *Chandra Ghanshyam, Jain Chirag.* Tech. Rep.: : Cold Spring Harbor Laboratory, 2022. — sep 1. — [Online; accessed 2022-12-10].

134. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm / Cheng, Concepcion, Feng et al. // *Nature Methods*. — 2021. — feb 1. — Vol. 18, no. 2. — Pp. 170–175.

135. HiCanu: Accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads / Sergey Nurk, Brian P. Walenz, Arang Rhie et al. // *Genome Research*. — 2020. — aug 14. — Vol. 30, no. 9. — Pp. 1291–1305. — [Online; accessed 2022-12-10].

136. Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads / Anton Bankevich, Andrey V. Bzikadze, Mikhail Kolmogorov et al. // *Nature Biotechnology*. — 2022. — feb 28. — Vol. 40, no. 7. — Pp. 1075–1081. — [Online; accessed 2023-02-18].

137. Telomere-to-telomere assembly of diploid chromosomes with Verkko / Mikko Rautiainen, Sergey Nurk, Brian P. Walenz et al. // *Nature Biotechnology*. — 2023. — feb 16. — [Online; accessed 2023-02-18].

Table of figures

1.1 SPAligner pipeline for nucleotide sequence alignment 125
1.2 SPAligner pipeline for amino acid sequence alignment 129
1.3 Metagenomic assembly subgraph with a reconstructed fragmented beta-lactamase protein sequence 135

2.1 Distribution of the lengths for known Cry and Vip insecticide proteins and bacterial genes from the Uniprot database 138
2.2 The ORFograph pipeline 139
2.3 Conflicting and non-conflicting paths in the assembly graph 142
2.4 Diversity of the reliable Cry1 sequences identified from isolated bacteria datasets 148
2.5 A subgraph of the metagenomic assembly graph aligned to Cry4Aa and Cry4Ba proteins 149

3.1 The architecture of centromere on chromosome X 152
3.2 Decomposition into HORs for centromere of chromosome X and centromere of chromosome 8 153
3.3 The String Decomposition (SD) graph visualized as a "book" 157
3.4 The distribution of $Identity$ (x-axis) and $IdentityDiff$ (y-axis) for monomer alignments in centromeric regions 161
3.5 Distribution of percent identities for each of the twelve cenX monomers and the gap monomer "?" 164
3.6 Monoread-to-monocentromere alignments 165
3.7 Mismatch substitution matrices for the AC and SD approaches 166
3.8 Analysis of the non-standard HOR ABCDEFGHIJFGHIJKL 167
3.9 CentromereArchitect pipeline 170
3.10 HORmon pipeline 172
3.11 Dehybridization substitutes hybrid vertices (monomers) by hybrid edges in the monomer graph of cen5. 173
3.12 Dehybridization substitutes hybrid vertices (monomers) by hybrid edges in the monomer graph of cen8. 174
3.13 The monomer graphs for cenX and cen9. 176
3.14 Decomposition of cenX into HORs 176

Table of tables

1 Datasets for benchmarking GraphAligner and SPAligner 131
2 The performance of SPAligner and GraphAligner on aligning PacBio reads to short-read assembly graphs 132
3 The performance of SPAligner and GraphAligner on aligning ONT reads to short-read assembly graphs 133
4 Information about seven simulated datasets enriched with Cry genes . . 145
5 Information about the identity between six analyzed Cry proteins 146
6 ORFograph benchmarking on five simulated datasets of varying complexity 147
7 Summary of errors in the alignments between monoreads and monocentromeres computed by the AC and SD tools for 12 monomers . 165
8 Summary of errors in the alignments between monoreads and monocentromeres computed by the AC and SD tools for 13 monomers . 168