МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет Механико-математический
Кафедра Программирования

Направление подготовки Математика и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Паршуков Даниил Леонидович

Тема работы: Исследование и реализация алгоритма RecombinHunt для обнаружения рекомбинантных вирусных последовательностей

«К защите допущен»
И.О. заведующего кафедрой
к.ф.-м.н., с.н.с ИСИ СО РАН
Емельянов П. Г. / .....
(фамилия, И., О.) / (подпись, МП)
«.....».....20...г.

Научный руководитель
программист 1 кат. ИСИ СО РАН
Девятериков А. П. / .....
(фамилия, И., О.) / (подпись, МП)
«.....».....20...г.

Руководитель от НГУ
И.О. заведующего кафедрой
к.ф.-м.н., с.н.с ИСИ СО РАН
Емельянов П. Г. / .....
(фамилия, И., О.) / (подпись, МП)
«.....».....20...г.

Дата защиты: «.....».....20...г.

Новосибирск, 2026

Реферат

Название работы: Исследование и реализация алгоритма RecombinHunt для обнаружения рекомбинантных вирусных последовательностей.

Работа посвящена исследованию и реализации алгоритмов для обнаружения рекомбинантных вариантов вирусных геномов. Рекомбинация играет ключевую роль в эволюции вирусов, определяя их способность адаптироваться к иммунным барьерам и изменять фенотипические свойства. В условиях роста объёмов генетических данных и необходимости оперативного анализа требуется разработка высокопроизводительных решений для выявления рекомбинантов.

Целью данной работы является изучение современных алгоритмов поиска рекомбинантов и реализация одного из перспективных подходов – RecombinHunt, с последующей адаптацией и оптимизацией его с использованием C++. В рамках исследования выполняется анализ принципов работы существующих методов, сравнение их эффективности и определение областей возможных улучшений. Особое внимание уделяется применимости алгоритма к большим наборам данных (например, полной коллекциям геномов SARS-CoV-2).

Результаты исследования могут быть использованы при разработке программных инструментов для биоинформатического анализа, эпидемиологического мониторинга и изучения эволюции вирусов.

Количество страниц: 22

Количество использованных источников: 8

Количество иллюстраций: 2

Количество таблиц: 1

Ключевые слова: Рекомбинация, вирусные геномы, SARS-CoV-2, RecombinHunt, обнаружение рекомбинантов, C++, производительность

Содержание

ВВЕДЕНИЕ

Рекомбинация генетического материала представляет собой фундаментальный биологический процесс, играющий ключевую роль в эволюции вирусов [8]. Этот механизм, заключающийся в обмене участками геномов между разными штаммами, служит мощным двигателем генетического разнообразия и может приводить к возникновению вариантов с новыми свойствами - повышенной вирулентностью, устойчивостью к терапии или способностью ускользать от иммунного ответа. Особую актуальность задача обнаружения рекомбинантов приобрела в контексте пандемии SARS-CoV-2, где оперативное выявление новых генетических вариантов стало критически важным для эффективного эпидемиологического надзора.

Современные достижения в области секвенирования привели к экспоненциальному росту объёмов геномных данных. Базы, содержащие миллионы полногеномных последовательностей, стали обычным явлением, что выявило принципиальные ограничения классических методов анализа рекомбинации. Традиционные филогенетические подходы и алгоритмы, основанные на попарном сравнении последовательностей, как правило, обладают квадратичной или кубической вычислительной сложностью относительно числа анализируемых геномов. При переходе к масштабу пандемийного надзора такие методы становятся практически неприменимыми для задач оперативного скрининга. Возникает насущная потребность в вычислительных инструментах, сложность которых растёт линейно с размером входных данных и которые способны обрабатывать всю доступную коллекцию последовательностей без предварительной субдискретизации.

Значительным прорывом в решении этой проблемы стал подход, основанный на данных, реализованный в алгоритме RecombinHunt [1]. В отличие от классических методов, опирающихся на косвенные статистические признаки рекомбинации или ресурсоёмкое филогенетическое моделирование, RecombinHunt работает напрямую с мутационными профилями известных вирусных линий. Алгоритм выявляет рекомбинанты путём сопоставления наборов мутаций целевой последовательности с эталонными профилями линий, минуя стадию построения эволюционных моделей и полных множественных выравниваний. Вычислительная сложность такого подхода линейна относительно длины анализируемой последовательности

и числа рассматриваемых линий, что принципиально отличает его от традиционных методов с нелинейной сложностью и делает кандидатом для анализа данных пандемийного масштаба.

Опубликованное описание алгоритма сопровождается прототипной реализацией, выполненной на интерпретируемом языке и не ориентированной на высокопроизводительные вычисления. Вместе с тем вычислительное ядро алгоритма - интенсивные операции над матрицами логарифмических отношений правдоподобия, кумулятивные суммирования, многократные вычисления информационных критериев на тысячах кандидатов - имеет структуру, естественно отображаемую на эффективные конструкции компилируемого языка системного программирования. Это открывает возможность создания реализации, способной работать с данными экстремальных объёмов без характерных для интерпретируемых сред накладных расходов на динамическую типизацию и управление памятью.

Объектом исследования является алгоритм RecombinHunt как представитель подхода, основанного на данных, к обнаружению рекомбинантных последовательностей в вирусных геномах.

Предмет исследования - вычислительная эффективность реализации алгоритма на языке C++.

Цель работы - создание высокопроизводительной программной реализации алгоритма RecombinHunt на языке C++ и количественная оценка её вычислительных характеристик в сопоставлении с существующей реализацией.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести аналитический обзор алгоритмической структуры RecombinHunt, выделить компоненты, критические для общей производительности.

2. Спроектировать и реализовать на языке C++ программную систему, воспроизводящую полную функциональность алгоритма

3. Провести экспериментальное сравнение разработанной C++ реализации с существующей, количественно оценить достигаемое ускорение и характер масштабирования обеих версий.

4. Проанализировать полученные результаты и сформулировать рекомендации по дальнейшей оптимизации инструментов данного класса для задач обработки коллекций вирусных геномов.

Практическая значимость работы заключается в создании программного инструмента, пригодного для оперативного скрининга рекомбинантных форм вирусов в масштабных геномных базах данных, а также в получении количественных оценок, которые могут служить ориентиром при выборе технологической платформы для разработки аналогичных высоконагруженных средств биоинформатического анализа.

1. АНАЛИЗ МЕТОДОВ ОБНАРУЖЕНИЯ РЕКОМБИНАЦИИ И АЛГОРИТМИЧЕСКАЯ СТРУКТУРА RECOMBINHUNT

1.1. Классификация методов и проблемы вычислительной масштабируемости

Разработка вычислительных методов поиска рекомбинантных последовательностей ведётся с начала 2000-х годов. Систематический обзор и классификация существующих подходов представлены в работе [5], где также подробно обсуждаются последствия рекомбинации для филогенетического анализа и методы её выявления. Существующие подходы можно разделить на четыре основных класса, принципиально различающихся по типу используемой информации и характеру зависимости времени работы от объёма входных данных.

Методы, основанные на попарных расстояниях. Исторически первый класс алгоритмов, к которому относится, в частности, программа RDP (Recombination Detection Program) [6]. Идея состоит в сканировании множественного выравнивания скользящим окном и выявлении участков, на которых попарное сходство между последовательностями значимо меняется - это интерпретируется как признак смены эволюционной истории, вызванной рекомбинацией. Вычислительная сложность типичного представителя составляет $O(N^2 \cdot L)$, где N - число последовательностей в выравнивании, L - его длина. На выборках в несколько тысяч геномов время счёта становится неприемлемо большим. В пятой версии пакета RDP5 [7] авторы интегрировали десять различных методов, добавили поддержку параллельных вычислений и добились ускорения примерно на четверть по сравнению с предыдущей версией, однако фундаментальное ограничение - квадратичная зависимость от N - сохранилось.

Филогенетические методы. Этот класс опирается на неконгруэнтность (несогласованность) филогенетического сигнала в разных участках генома. Алгоритмы строят филогенетические деревья в скользящем окне и сравнивают их топологии: резкое изменение топологии указывает на возможную точку рекомбинации. Представителями являются SISCAN,

SimPlot, а также ряд модулей, включённых в состав RDP5 [7]. Построение дерева для одного окна имеет сложность не ниже $O(N^3)$, а при длине окна, сравнимой с длиной генома, итоговая сложность достигает $O(N^3 \cdot L)$ и выше. Для тысяч последовательностей такие методы практически неприменимы без радикальных эвристических упрощений или предварительного радикального прореживания выборки.

Методы на основе непараметрической статистики. Алгоритмы семейства 3SEQ [2] сводят задачу к комбинаторному тесту на тройках последовательностей. Исходная версия перебирает все $C(N, 3)$ троек, обрабатывая каждую за время, квадратичное по числу информативных позиций, что даёт итоговую сложность $O(N^3 \cdot m^2)$, где N - число последовательностей, m - число информативных сайтов. В [4] авторам удалось понизить сложность обработки одной тройки до линейной, сократив общую сложность до $O(N^3 \cdot m)$. Кубическая зависимость от числа последовательностей при этом сохранилась, что делает метод неприменимым для выборок пандемийного масштаба без предварительной субдискретизации, неизбежно влекущей риск пропуска редких рекомбинантных форм.

Важным дополнением к приведённой классификации является работа [3], в которой выполнено систематическое эмпирическое сравнение восьми методов обнаружения рекомбинации на симулированных вирусных данных. Авторы количественно оценили точность и вычислительную масштабируемость методов, подтвердив фундаментальный компромисс между чувствительностью к рекомбинации и производительностью при росте объёма данных.

Методы, основанные на данных. Наиболее поздний класс, к которому принадлежит и рассматриваемый в данной работе алгоритм RecombinHunt [1], принципиально отказывается от попарного сравнения последовательностей. Вместо полного множественного выравнивания анализируемой коллекции используется предварительно построенная база знаний о мутационных профилях известных вирусных линий. Каждая вновь поступающая последовательность сопоставляется с этими профилями независимо от остальных. Вычислительная сложность обработки одного генома составляет $O(L \cdot K)$, где K - число линий в базе.

Ключевой водораздел между первыми тремя классами и четвёртым пролегает именно по наличию или отсутствию зависимости времени рабо-

ты от N. Традиционные методы явно или неявно сравнивают последовательности друг с другом, что при переходе к масштабу миллионов образцов становится запретительно дорогим. Подход, основанный на данных, заменяет межпоследовательностное сравнение обращением к заранее вычисленным статистикам, устраняя этот фундаментальный ограничитель. Именно данное обстоятельство делает алгоритм RecombinHunt наиболее перспективным кандидатом для создания инструмента оперативного скрининга в масштабе глобальных геномных баз.

1.2. Алгоритм RecombinHunt

Алгоритм RecombinHunt реализует подход, основанный на данных, в следующей конкретной постановке. Для каждой вирусной линии L_i из принятой номенклатуры (например, классификации PANGO для SARS-CoV-2) предварительно, однократно, вычисляются три величины:

- $P(m \in L_i)$ - частота встречаемости мутации m среди всех последовательностей, отнесённых к линии L_i;
- $P(m)$ - фоновая частота мутации m во всей популяции вируса;
- бинарный профиль характерных мутаций линии - список позиций, мутации в которых присутствуют у всех или подавляющего большинства представителей линии.

Эти величины образуют статическую базу знаний, которая вычисляется один раз по референсной выборке и не зависит от состава анализируемой коллекции геномов. При поступлении новой целевой последовательности T для неё определяется множество нуклеотидных замен относительно референсного генома, после чего вычисляется центральная для алгоритма функция - логарифм отношения правдоподобия для линии L_i на интервале геномных позиций от $start$ до end:

$$\text{Likelihood}_{\text{ratio}}(L_i, T, R_{start:end}) = \sum_{\forall m \in R} \begin{cases} \ln\left(\dfrac{P(m \in L_i)}{P(m)}\right), & \text{если } m \in T \\ -\ln\left(\dfrac{P(m \in L_i)}{P(m)}\right), & \text{если } m \notin T \text{ и } m \in L \\ 0, & \text{иначе} \end{cases}$$

Смысл функции интуитивно понятен. Если мутация m присутствует в целевой последовательности и одновременно характерна для линии L_i, отношение $P(m \in L_i)/P(m)$ больше единицы, его логарифм положителен, и вклад в сумму повышает оценку линии. Если же мутация характерна для линии (члены L_i почти всегда её несут), но в T она отсутствует, логарифм берётся со знаком минус, штрафуя линию за несоответствие. Все мутации, не являющиеся характерными для L_i, дают нулевой вклад и не влияют на результат. Таким образом, функция измеряет, насколько хорошо мутационный профиль линии объясняет набор замен, наблюдаемый в целевом геноме на заданном интервале.

Общая схема алгоритма показана на рисунке 1.

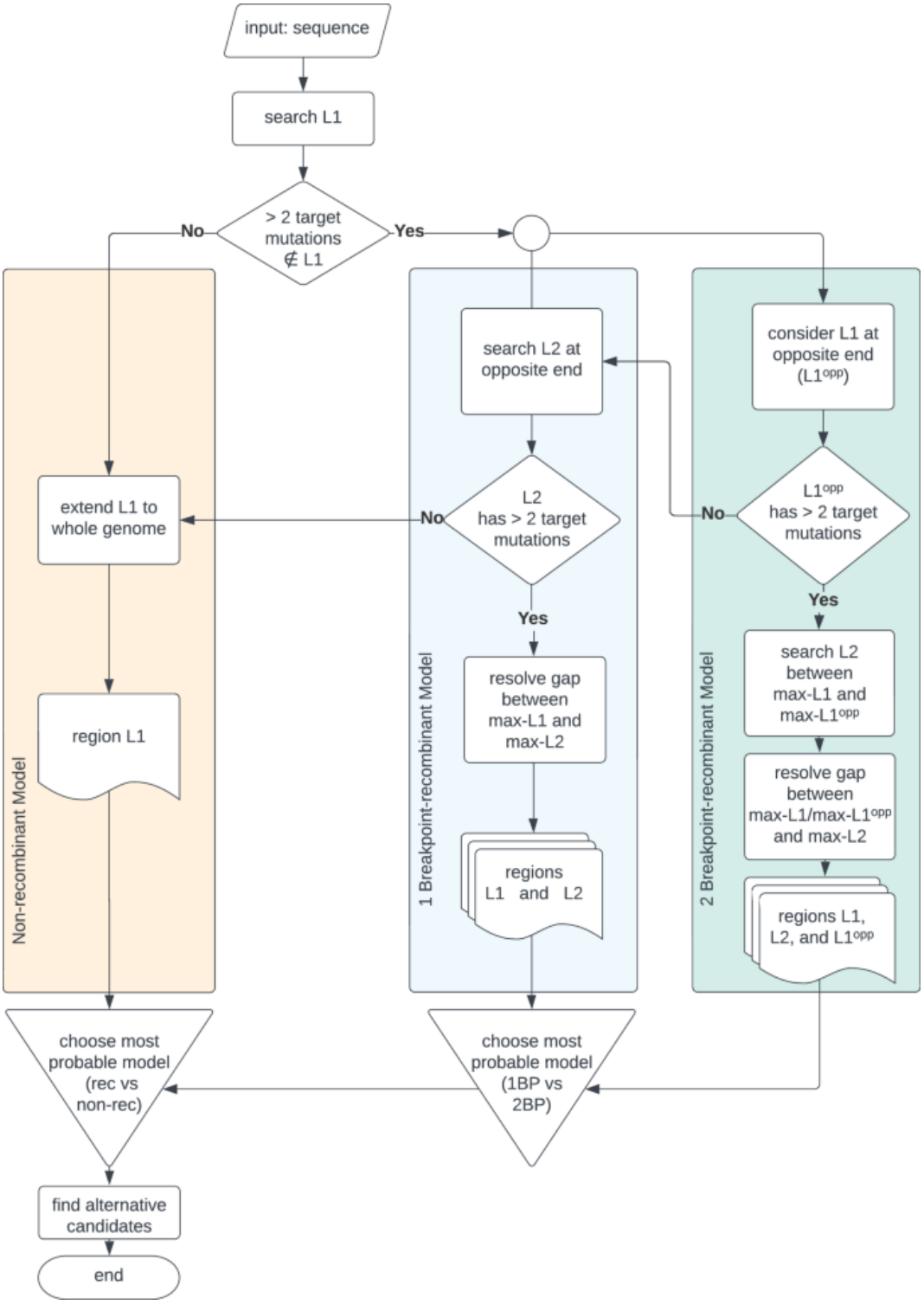

Рис. 1 – Схема алгоритма RecombinHunt. Источник: (Alfonsi et al., 2024), Figure 2 (лицензия CC BY 4.0).

Далее последовательно рассматриваются основные этапы работы алгоритма.

Этап 1. Поиск первичного кандидата. Для целевой последовательности вычисляются две кумулятивные суммы $\text{Likelihood}_{\text{ratio}}$ для всех линий из базы данных: в прямом направлении (от 5'-конца к 3'-концу) и в обрат-

ном (от 3'-конца к 5'-концу). Линия, достигающая максимального значения кумулятивной суммы, назначается первым кандидатом и обозначается $L1$. Позиция, в которой достигается этот максимум (обозначим её $maxL1$), задаёт предполагаемую границу рекомбинации. Если расхождение между мутационным профилем $L1$ и наблюдаемыми заменами в T минимально (менее трёх позиций), последовательность признаётся нерекомбинантной и целиком относится к линии $L1$; дальнейшие шаги не выполняются.

Этап 2. Построение и сравнение моделей рекомбинации. Если целевая последовательность не может быть объяснена одной линией, алгоритм параллельно строит две конкурирующие гипотезы:

- **Модель с одной точкой разрыва (1BP):** предполагает, что геном состоит из двух участков, происходящих от разных линий: $L1$ и $L2$. Второй кандидат $L2$ ищется на противоположном конце генома, в области, не покрытой $L1$. Для $L2$ также вычисляется кумулятивная сумма, и её максимальное значение определяет вторую предполагаемую границу ($maxL2$). Интервал между $maxL1$ и $maxL2$ задаёт область разрыва.

- **Модель с двумя точками разрыва (2BP):** предполагает трёхсегментную структуру $L1 - L2 - L1$. Она рассматривается в тех случаях, когда линия $L1$ имеет характерные мутации на обоих концах генома. Область между двумя сегментами $L1$ заполняется кандидатом $L2$, который отыскивается с помощью двунаправленного поиска.

Для обеих моделей реализована процедура уточнения границ (разрешение разрыва). Если границы, определённые по максимумам кумулятивных сумм, не являются смежными в пространстве мутаций, позиция разрыва p выбирается так, чтобы минимизировать суммарную потерю правдоподобия для обоих соседних регионов: участок слева от p относится к левому региону, участок справа - к правому.

Этап 3. Статистическая оценка и выбор модели. Качество построенных моделей сравнивается с помощью информационного критерия Акаике (AIC). Для этого вычисляется глобальная функция правдоподобия на всём интервале:

$$\text{Global}_{\text{likelihood}}(L_i, T, R_{start:end}) = \sum_{\forall m \in R} \begin{cases} \ln(P(m \in L_i)), & \text{если } m \in T \text{ и } m \in L_i \\ -\ln(P(m \in L_i)), & \text{если } m \notin T \text{ и } m \in L_i \end{cases}$$

Значение AIC позволяет корректно сравнивать модели с разным числом параметров (одна линия, две линии, три линии). Модель с наименьшим значением AIC и, следовательно, наибольшей статистической значимостью при попарном сравнении выбирается в качестве итоговой. Дополнительно выполняется сравнение с нерекомбинантной моделью (одиночная линия), чтобы убедиться, что факт рекомбинации статистически значим.

Этап 4. Выявление альтернативных кандидатов. Алгоритм не ограничивается одной парой линий-участников. Для каждого из регионов ($L1$ или $L2$) формируется множество филогенетически близких альтернативных кандидатов. Линия включается в это множество при одновременном выполнении трёх условий: (1) разница в значении AIC с наилучшим кандидатом не превышает заданного порога ($p \geq 10^{-5}$); (2) позиция максимума правдоподобия для этой линии находится в пределах одной мутации от позиции наилучшего кандидата; (3) линия принадлежит к той же филогенетической ветви, что и наилучший кандидат. Такая стратегия позволяет учесть естественную неопределённость в точном определении родительских линий, связанную с неполнотой данных и наличием близкородственных вариантов.

1.3. Вычислительное ядро: критически нагруженные компоненты

Анализ описанной схемы с точки зрения вычислительной нагрузки позволяет выделить несколько компонентов, в которых сосредоточена основная доля времени выполнения при обработке одного генома.

1. Многократное вычисление логарифмических отношений правдоподобия. Функция $\text{Likelihood}_{\text{ratio}}$ вычисляется для каждой линии и каждой геномной позиции, в которой зафиксирована мутация. При характерных значениях $K \approx 3{\cdot}10^3$ линий и $m \approx 10^3$ мутаций на геном получается

порядка трёх миллионов элементарных операций на одну последовательность. Вычислительная структура этих операций - позлементные действия над матрицей «линия–мутация» - естественным образом векторизуется, что важно для итоговой производительности.

2. Кумулятивные суммирования. Для поиска максимумов правдоподобия вдоль генома выполняются прямые и обратные кумулятивные суммы по всем линиям. Сложность этой операции составляет $O(K \cdot L)$. При наивной реализации она может порождать избыточное копирование данных. Использование вычислений на месте и размещение данных в непрерывных блоках памяти позволяет избежать фрагментации и сократить число аллокаций.

3. Вычисление информационного критерия Акаике. Расчёт AIC и соответствующих p-значений требует вычисления глобального логарифма правдоподобия на нескольких интервалах и для нескольких комбинаций линий. При большом потоке анализируемых последовательностей необходимо исключать повторные логарифмирования одних и тех же частот, кешируя промежуточные результаты.

4. Операции с филогенетической иерархией. Проверка отношений «родитель–потомок» и формирование групп альтернативных кандидатов опираются на структуру данных, представляющую номенклатуру PANGO: разворачивание псевдонимов, определение иерархических связей между линиями. При большом числе запросов неоптимальная реализация этих операций может стать скрытым узким местом.

Перечисленные компоненты определяют требования к целевой вычислительной платформе: компилируемый язык для устранения накладных расходов времени выполнения, эффективные библиотеки линейной алгебры для матричных операций, компактное представление табличных данных в памяти и минимальные накладные расходы на динамическое управление объектами. Язык C++ в сочетании с библиотеками Eigen и Apache Arrow представляет собой естественный выбор, позволяющий адресовать каждое из перечисленных требований.

2. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ АЛГОРИТМА RECOMBINHUNT НА ЯЗЫКЕ C++

2.1. Архитектура программной системы

Программная система реализована в виде разделяемой библиотеки, предоставляющей программный интерфейс на языке C++. Такое решение позволяет встраивать функциональность обнаружения рекомбинации в произвольные вычислительные конвейеры без необходимости запуска внешних процессов или межъязыкового взаимодействия, а также открывает возможность параллельного вызова процедуры анализа из нескольких потоков.

Библиотека состоит из шести основных модулей, взаимодействие которых организовано вокруг единого объекта окружения, загружаемого однократно и разделяемого между всеми экземплярами Experiment. Схема архитектуры представлена на рисунке 2.

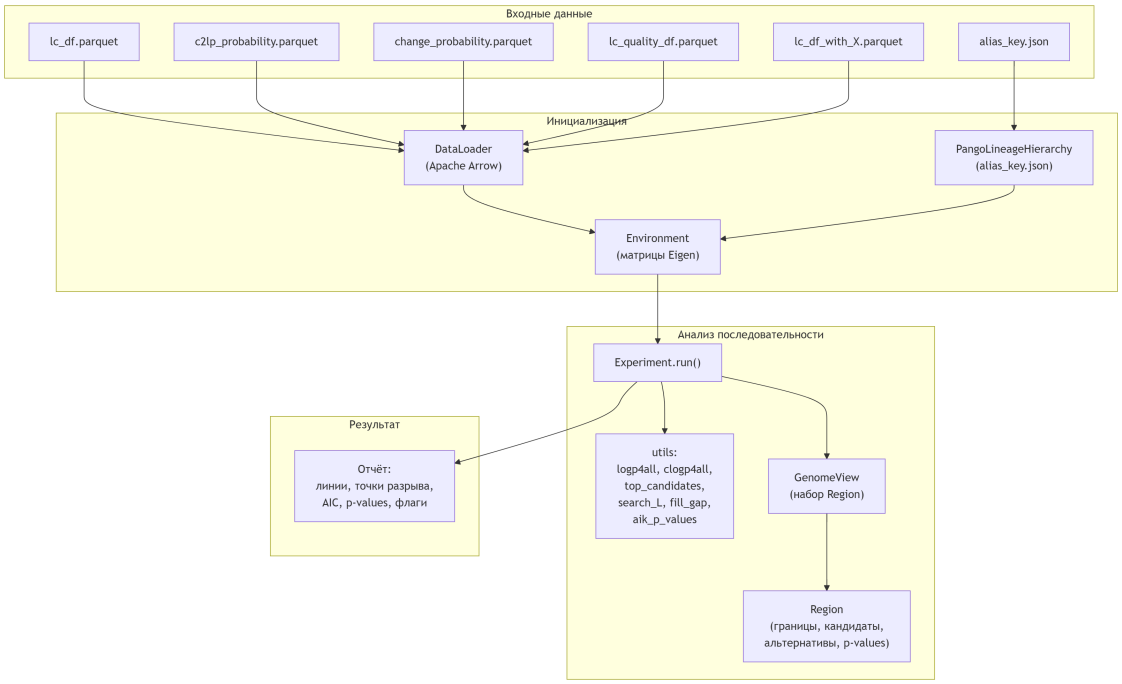

Рис. 2 – Архитектура программной системы.

Далее приводится назначение каждого модуля.

Модуль загрузки данных (DataLoader). Обеспечивает чтение предварительно вычисленных статистических данных из файлов формата

Parquet. Реализован с использованием библиотеки Apache Arrow, что позволяет эффективно обрабатывать столбцовые табличные данные. Поддерживается чтение колонок целочисленных, вещественных и булевых типов с автоматическим приведением к формату с плавающей точкой двойной точности, используемому во всех вычислительных модулях.

Модуль окружения (Environment). Центральное хранилище всех данных, необходимых для анализа. Содержит матрицы характерных мутаций, вероятностей принадлежности мутаций линиям и глобальных вероятностей, а также сведения об иерархии линий. Предоставляет методы для построения объединённой матрицы «мутации–линии» для целевой последовательности и матрицы соответствующих вероятностей. Поддерживает исключение заданных линий из анализа и копирование с фильтрацией.

Модуль иерархии линий (PangoLineageHierarchy). Реализует логику работы с классификацией PANGO: разворачивание псевдонимов, определение отношений «родитель–потомок», вычисление иерархического расстояния между линиями. Используется на этапе фильтрации альтернативных кандидатов для каждого найденного региона.

Модуль представления генома (Region и GenomeView). Реализует объектную модель участка генома, ассоциированного с конкретной линией-кандидатом. Region хранит границы участка (в координатах объединённой матрицы и в координатах целевой последовательности), список кандидатов, значения логарифмов правдоподобия и статистику сравнения альтернативных кандидатов. GenomeView управляет набором регионов, покрывающих анализируемую последовательность, и предоставляет методы для вычисления свободных участков, точек разрыва и списка линий, участвующих в рекомбинации.

Модуль вычислений (utils). Содержит реализации ключевых статистических функций: вычисление матрицы логарифмических отношений правдоподобия для всех линий, кумулятивные суммы, поиск наилучших кандидатов, процедуру уточнения границ, вычисление информационного критерия Акаике и p-значений для сравнения моделей. Функции оперируют матрицами библиотеки Eigen и используют кеширование для устранения повторных вычислений.

Модуль эксперимента (Experiment). Центральный модуль, управляющий полным циклом анализа одной последовательности.

Сборка выполнена средствами CMake. Зависимости включают библиотеку Eigen для матричных операций и Apache Arrow для работы с форматом Parquet.

2.2. Данные окружения

Окружение алгоритма образуют предварительно вычисленные статистические данные о мутационных профилях вирусных линий, загружаемые из файлов формата Parquet. Данные включают характерные мутации каждой линии, вероятности принадлежности мутаций линиям и глобальные вероятности нуклеотидных замен. Исходный набор построен на основе выборки Nextstrain и включает более тысячи линий классификации PANGO.

Загрузка реализована с использованием библиотеки Apache Arrow и выполняется однократно при инициализации. Данные размещаются в матрицах Eigen и хранятся в объекте Environment, который разделяется между всеми экземплярами Experiment через разделяемые указатели. Это исключает избыточное копирование при обработке нескольких последовательностей.

Иерархия линий PANGO загружается из файла alias_key.json. На основе содержащихся в нём записей о псевдонимах строится двусторонний словарь, используемый на этапе фильтрации альтернативных кандидатов для проверки филогенетической близости линий.

2.3. Вычислительные модули: алгоритмические решения и особенности реализации

Вычислительное ядро образуют функции, реализующие основные этапы алгоритма. Ниже рассматриваются ключевые решения, принятые при их реализации.

Вычисление логарифмических отношений правдоподобия

Функция logp4all вычисляет для каждой пары «мутация–линия» логарифм отношения $P(m \in L_j)/P(m)$ с пороговой фильтрацией: значения, не достигающие заданного порога, заменяются на ограниченное отрица-

тельное значение, что эквивалентно отбрасыванию статистически незначимых отношений и численно устойчивее использования настоящей минус бесконечности. Результат кешируется; при повторных вызовах с теми же параметрами возвращается из кеша без перевычисления.

Функция clogp4all строит кумулятивную сумму матрицы отношений правдоподобия с учётом знака: логарифм прибавляется, если мутация принадлежит целевой последовательности, и вычитается, если мутация характерна для линии, но отсутствует в цели. Кумулятивная сумма последовательна по своей природе и не поддаётся векторизации.

Формирование матрицы вкладов, определяющей знак каждого слагаемого, объединяет три условных случая (мутация в цели, мутация только в линии, отсутствие мутации) в одном цикле. Для обратного направления реализован прямой проход с конца без создания промежуточных копий массива. Результат также кешируется; ключ кеша включает флаг направления, что позволяет различать прямой и обратный проходы.

Поиск наилучших кандидатов и уточнение границ

Функция top_candidates находит до десяти линий с наибольшим значением кумулятивной суммы в заданном интервале генома. Поиск выполняется в два этапа: вычисление максимума по каждой колонке матрицы через Eigen::colwise().maxCoeff() и последующая фильтрация по минимальному числу мутаций целевой последовательности. Если ни одна линия не удовлетворяет порогу, порог понижается до нуля, и поиск повторяется. Сортировка отобранных линий по убыванию максимума реализована стандартной std::sort по вектору индексов. Нормировка значений в окне поиска выполнена как одна матричная операция вычитания вектора из строк - Eigen::rowwise() - vector.

Функция fill_gap определяет оптимальную позицию разрыва между двумя соседними регионами путём перебора позиций в интервале неопределённости и суммирования значений правдоподобия левого и правого кандидатов. Сложность процедуры линейна относительно длины интервала; она не является узким местом.

Вычисление информационного критерия Акаике

Функции aik_p_values и update_region_p_values вычисляют информационный критерий Акаике для построенных моделей и сравнивают их статистическую значимость.

Для каждого региона строится маска мутаций как логическая сумма характерных мутаций кандидатов и мутаций цели через поэлементное OR в Eigen. Маска применяется через создание урезанной копии матрицы, содержащей только отмеченные строки: это воспроизводит семантику фильтрации по условию, но реализовано явно через копирование данных. Такой подход гарантирует, что каждый кандидат оценивается на одном и том же множестве мутаций.

Для каждого кандидата вычисляется кумулятивная сумма логарифмического правдоподобия по отфильтрованным строкам; её последнее значение принимается за итоговое правдоподобие. Сравнение двух моделей выполняется по формуле $p = \exp((\mathrm{AIC}_1 - \mathrm{AIC}_2)/2)$.

Для модели с двумя точками разрыва, где первый и третий регионы относятся к одной линии, маски L1-регионов объединяются, что обеспечивает корректное сравнение альтернативных кандидатов на общем множестве мутаций.

Управление потоком выполнения

Метод Experiment::run() реализует последовательный перебор гипотез о структуре целевой последовательности. После нахождения первичного кандидата L1 алгоритм пытается построить модель с одной точкой разрыва; если модель не может быть построена (не найден кандидат L2, удовлетворяющий минимальной длине региона), управление передаётся на построение модели с двумя точками разрыва. Если и она не может быть построена (не найден противоположный регион L1 или центральный кандидат L2), последовательность признаётся нерекомбинантной: первичный кандидат L1 расширяется на весь геном, и анализ завершается.

Переход между ветками алгоритма реализован через исключения. Ключевые вычислительные функции (search_L, top_candidates) выбрасывают исключение при отсутствии кандидатов; model1BP и model2BP - при невозможности построить соответствующую модель. Метод run() содержит

вложенные блоки try/catch, которые перехватывают эти исключения и направляют выполнение по альтернативному пути.

Для каждого исхода устанавливается флаг, реализованный как элемент перечисления (enum): SingleCandidateGenome (последовательность нерекомбинантна), Model_1BP_Best (выбрана модель с одной точкой разрыва), Model_2BP_Best (выбрана модель с двумя точками разрыва), а также флаги, фиксирующие конкретную причину неудачи - Model_1BP_NoL2, Model_2BP_Bad_L1_opp, Model_2BP_NoL2 и другие. Флаги доступны через программный интерфейс и позволяют для каждой проанализированной последовательности восстановить, какой путь был пройден алгоритмом.

2.4. Управление памятью и кеширование

Управление памятью опирается на два механизма: разделяемые указатели с подсчётом ссылок для матриц окружения и явное кеширование для повторно используемых результатов вычислений.

Матрицы, загруженные при инициализации окружения, используются несколькими компонентами одновременно. Для исключения избыточного копирования все объекты, нуждающиеся в доступе к матрице, хранят разделяемый указатель `std::shared_ptr<NamedMatrix>`. Это гарантирует освобождение памяти только после того, как последний пользователь прекратит работу с данными.

Класс Cache хранит результаты промежуточных вычислений в словаре с разделяемыми указателями на матрицы. Ключ кеша объединяет имя функции, размеры матрицы и флаг направления, что позволяет различать варианты вычисления (например, прямой и обратный проходы для кумулятивной суммы). Время жизни закешированных данных определяется временем жизни объекта Experiment: при вызове метода resetInternalVariables() кеш очищается явным образом, без опоры на сборщик мусора. Это позволяет повторно использовать один и тот же экземпляр Experiment для анализа нескольких последовательностей без накопления данных между запусками.

3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ

Для оценки производительности использовались 1393 линий классификации PANGO. Для каждой линии была сформирована консенсусная последовательность нуклеотидных замен, подаваемая на вход процедуре анализа. Тестовый набор покрывает все основные сценарии работы алгоритма: нерекомбинантный случай, модель с одной точкой разрыва и модель с двумя точками разрыва.

Сравнение проводилось с эталонной реализацией алгоритма RecombinHunt, выполненной на языке Python. Обе версии использовали одни и те же файлы окружения.

В таблице 1 приведены результаты сравнения времени выполнения.

Таблица 1: Сравнение времени выполнения (мс) для 1393 линий

Тип / Показатель	**Текущая**	**Эталонная**	**Ускорение**
Нерекомбинант	452–550	735–900	1.6
Одна точка разрыва (1BP)	500-600	800–1200	1.8
Две точки разрыва (2BP)	600–784	900–1593	2.0
Среднее на линию	477	784	1.6
Суммарно 1393 линий	664.4 сек	1093.4 сек	1.6

Среднее время анализа одной линии составило 477 мс для разработанной реализации против 784 мс для эталонной - ускорение в 1.6 раза. Минимальное время (452 мс против 735 мс, ускорение в 1.6 раза) достигается на нерекомбинантных последовательностях; максимальное (784 мс против 1593 мс, ускорение в 2.0 раза) - на наиболее сложных случаях с двумя точками разрыва. Рост времени при переходе от нерекомбинантного случая к модели с одной точкой разрыва и далее к модели с двумя точками разрыва объясняется увеличением объёма вычислений: каждая дополнительная точка разрыва требует нового поиска кандидата и дополнительной процедуры уточнения границ.

Основной вклад в достигнутое ускорение вносит переход на компилируемый язык: время выполнения сократилось в среднем в 1.6 раза при сохранении полной функциональности алгоритма. Дополнительный эффект дали замена части поэлементных циклов матричными операциями Eigen с

автоматической векторизацией и устранение промежуточных копирований при вычислении обратной кумулятивной суммы.

Абсолютное время порядка 477 мс на линию и линейный характер масштабирования алгоритма делают разработанную реализацию пригодной для оперативного скрининга больших объёмов данных. При пакетной обработке миллионов последовательностей дополнительный резерв ускорения даёт параллельный вызов метода анализа из нескольких потоков - архитектура разделяемой библиотеки с однократно загружаемым окружением такую возможность предусматривает.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была создана программная реализация алгоритма RecombinHunt на языке C++ и проведено экспериментальное сравнение её характеристик с эталонной реализацией.

1. Проведён анализ алгоритмической структуры RecombinHunt. Показано, что алгоритм обладает линейной вычислительной сложностью относительно длины последовательности и числа линий, не зависящей от размера коллекции геномов, что принципиально отличает его от традиционных методов обнаружения рекомбинации с квадратичной или кубической сложностью.

2. Спроектирована и реализована на языке C++ программная система в виде разделяемой библиотеки, воспроизводящая полную функциональность алгоритма. В реализации использованы библиотека Eigen для матричных операций и Apache Arrow для загрузки статистических данных из формата Parquet. Архитектура системы предусматривает возможность параллельного вызова метода анализа из нескольких потоков.

3. Проведено экспериментальное сравнение разработанной реализации с эталонной на наборе из 1393 линий классификации PANGO. Среднее время анализа одной линии составило 477 мс против 784 мс - ускорение в 1.6 раза. Суммарное время обработки 1393 линий сократилось с 1093.4 до 664.4 секунд.

Полученные результаты показывают, что реализация алгоритма RecombinHunt на компилируемом языке обеспечивает существенное сокращение времени обработки и может рассматриваться как основа для дальнейшей разработки высокопроизводительных инструментов анализа вирусных геномов.

СПИСОК ЛИТЕРАТУРЫ

1. Alfonsi T., Bernasconi A., Chiara M., Ceri S. Data-driven recombination detection in viral genomes // Nature Communications. 2024. Vol. 15, Iss. 1. Art. 3313. https://doi.org/10.1038/s41467-024-47464-5
2. Boni M.F., Posada D., Feldman M.W. An exact nonparametric method for inferring mosaic structure in sequence triplets // Genetics. 2007. Vol. 176, No. 2. P. 1035-1047. https://doi.org/10.1534/genetics.106.068874
3. Jaya F.R., Brito B.P., Darling A.E. Evaluation of recombination detection methods for viral sequencing // Virus Evolution. 2023. Vol. 9, Iss. 2. Art. vead066. https://doi.org/10.1093/ve/vead066
4. Lam H.M., Ratmann O., Boni M.F. Improved algorithmic complexity for the 3SEQ recombination detection algorithm // Molecular Biology and Evolution. 2018. Vol. 35, No. 1. P. 247-251. https://doi.org/10.1093/molbev/msx283
5. Martin D.P., Lemey P., Posada D. Analysing recombination in nucleotide sequences // Molecular Ecology Resources. 2011. Vol. 11, Iss. 6. P. 943-955. https://doi.org/10.1111/j.1755-0998.2011.03026.x
6. Martin D.P., Rybicki E. RDP: detection of recombination amongst aligned sequences // Bioinformatics. 2000. Vol. 16, Iss. 6. P. 562–563. https://doi.org/10.1093/bioinformatics/16.6.562
7. Martin D.P., Varsani A., Rouard P., et al. RDP5: a computer program for analyzing recombination in, and removing signals of recombination from, nucleotide sequence datasets // Virus Evolution. 2021. Vol. 7, No. 1. Art. veaa087. https://doi.org/10.1093/ve/veaa087
8. Pérez-Losada M., Arenas M., Galán J.C., et al. Recombination in viruses: mechanisms, methods of study, and evolutionary consequences // Infection, Genetics and Evolution. 2015. Vol. 30. P. 296–307. https://doi.org/10.1016/j.meegid.2014.12.022