Санкт-Петербургский государственный университет

Программная инженерия
Системное программирование

Клещин Антон Сергеевич

Исправление ошибок в зашумленных последовательностях при помощи графов сборки

Выпускная квалификационная работа магистра

Научный руководитель:
доц. каф. СП, к. т. н. Литвинов Ю. В.

Научные консультанты:
доц. каф. стат. мод., к. ф.-м. н. Коробейников А. И.
старший н. с., к. ф.-м. н. Пржибельский А. Д.

Рецензент:
приглашённый н. с., к. ф.-м. н. Нурк С. Ю.

Санкт-Петербург
2022

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Anton Kleshchin

Error correction in noisy sequences using assembly graphs

Master's Thesis

Scientific supervisor:
C.Sc., Associate Professor Yurii Litvinov

Scientific consultants:
Ph.D., Associate Professor Anton Korobeynikov
Ph.D., Senior Research Fellow Andrey Prjibelski

Reviewer:
Ph.D., Visiting Fellow Sergey Nurk

Saint-Petersburg
2022

Оглавление

Введение

Сборка геномов является одной из важнейших задач биоинформатики. За многие годы учёным уже удалось собрать, а затем и расшифровать большое количество геномов. Несмотря на это, автоматическая реконструкция генома всё ещё остаётся сложной вычислительной задачей [27, 6].

Обычно сборка генома происходит из ридов (англ. reads) — фрагментов ДНК, которые получаются в результате секвенирования, а результатом являются контиги (англ. contigs, восстановленные части ДНК из ридов) или скаффолды (англ. scaffolds, восстановленные части ДНК с пропусками известной длины).

Технологии секвенирования не стоят на месте: появилось третье поколение, позволяющие получать длинные риды дешевле, но количество ошибок в них на порядок выше [5]. Несмотря на то, что есть способы частично бороться с этими ошибками [19], для улучшения результатов сборки можно прибегнуть к другим методам. Например, для небольших геномов активно применяется метод гибридной сборки [34, 15, 29], в которой одновременно используются как длинные риды с большим количеством ошибок, так и короткие, уровень ошибок в которых минимален. Короткие риды здесь выступают основным материалом для построения графа сборки, а длинные риды позволяют находить более длинные пути в этом графе, которые в дальнейшем становятся контигами.

Кроме того, даже с небольшими геномами есть проблемы. Например, есть штаммы бактерий, получить большое количество экземпляров которых крайне сложно. В этом случае покрытие генома полученными ридами оказывается очень низким, как и качество длинных ридов. Более того, некоторые штаммы вообще невозможно получить без окружающих их бактерий, поэтому прибегают к метагеномике — направлению геномики, в котором рассматриваются ДНК не отдельного организма, а сразу множества. Покрытие ридами генома каждой бактерии в рамках одной сборки может сильно отличаться, что только ухудшает качество

контигов бактерий с низким покрытием.

Уже существуют инструменты, которые стараются исправлять контиги, используя риды. Их основная идея — найти выравнивания ридов на контиги и с помощью какого-нибудь алгоритма консенсуса решить, какие нуклеотиды исправлять. При этом подходе теряется информация, которую нам даёт граф сборки. Во-первых, рёбра графа в основном существенно длиннее ридов, а также уже проведена коррекция ошибок в самих ридах. Во-вторых, граф сборки даёт связи между рёбрами. Так как любой контиг это путь в графе, то можно попытаться исправить фрагмент контига, найдя подходящий путь между рёбрами, в местоположении которых в контиге мы уверены.

Итак, оказывается актуальной разработка нового инструмента, позволяющего исправлять ошибки в контигах при помощи графов сборки.

1. Постановка задачи

Целью данной работы является создание инструмента, позволяющего исправлять ошибки в контигах при помощи графов сборки. Для достижения этой цели были сформулированы следующие задачи.

- Формирование критериев фильтрации выравниваний рёбер графа на последовательности.
- Разработка алгоритма исправления ошибок за пределами выравненных рёбер с помощью путей в графе.
- Разработка алгоритма переноса полученных путей в графе обратно в последовательности.
- Реализация итогового алгоритма в виде отдельного инструмента.
- Апробация алгоритма на симулированных и реальных данных.

2. Обзор предметной области

2.1. Представление генома

Большинство природных ДНК состоит из двух скрученных спиралей, к которым крепятся молекулы, называемые нуклеотидами [30]. Всего в спиралях присутствует четыре вида нуклеотида: аденин (A), цитозин (C), гуанин (G), тимин (T). Двойная спираль ДНК может иметь либо линейную структуру, либо кольцевую. Одноцепочечную ДНК содержат лишь некоторые вирусы и бактериофаги. При этом известно, что одна спираль полностью задаёт другую: напротив каждого нуклеотида из одной цепочки стоит комплементарный ей из другой. Для аденина это тимин, а для гуанина — цитозин.

Таким образом можно считать, что ДНК состоит из двух комплементарных строк над алфавитом из четырёх букв. Длину подстрок принято измерять в спаренных основаниях (англ. base pair, **bp**), которые эквивалентны одному символу строки.

2.2. Секвенирование и сборка геномов

Современные технологии не могут считывать всю ДНК за раз, поэтому в результате секвенирования получается множество фрагментов ДНК, называемых ридами. В зависимости от используемой технологии, длины ридов можно разделить на два класса: короткие и длинные. Короткие риды получают, в основном, при использовании технологии Illumina [5], которая получает цепочки нуклеотидов порядка 100-350 bp. Длинные риды получают технологиями Pacbio и Oxford Nanopore [5], которые получают цепочки порядка 5-100 kbp и 10-1000 kpb соответственно. Для сравнения, например, длина генома у бактерий измеряется в миллионах bp, у простых многоклеточных в сотнях миллионов bp, а у человека геном содержит около трёх миллиардов нуклеотидов.

К сожалению, при секвенировании происходят ошибки, поэтому ДНК читается несколько раз, чтобы в покрытии каждого нуклеотида результирующим набором ридов было больше правильных значений, чем оши-

бочных. В коротких ридах уровень ошибок составляет около 0.1-1%, в то время как для длинных ридов это значение доходит до 10-15% [5].

Задачей ассемблера является восстановление одной из нитей спирали ДНК по ридам. Обычно удаётся восстановить лишь фрагменты ДНК, которые являются либо контигами, либо скаффолдами. Их отличие в том, что в случае скаффолдов допускаются подстроки, в которых нуклеотиды неизвестны, что даёт дополнительную информацию в виде расстояния между контигами и их порядка следования в геноме.

Так как в коротких ридах мало ошибок, то достаточно небольшого покрытия генома ридами, чтобы граф, а следовательно и контиги, практически не содержали ошибки. Для длинных же ридов для достижения такого же результата покрытие должно быть большим, что иногда бывает просто невозможно получить, и к тому же дополнительные эксперименты стоят очень дорого.

2.3. Граф сборки

Методы сборки ридов в контиги далеко шагнули вперёд. В некоторых современных ассемблерах, таких как SPAdes [3], Velvet [32], RAY [4], ABySS [1], IDBA-UD [11], SKESA [28], используется подход, основанный на графе де Брюйна (англ. de Bruijn graph) [20]. Этот граф строится по следующим принципам: берётся набор входных строк и выделяется из них множество всевозможных подстрок длины k+1 (k+1-меры). Тогда k+1-мер — это ребро, ведущее из своего префикса длины k в суффикс длины k. Пример графа можно видеть на рисунке 1.

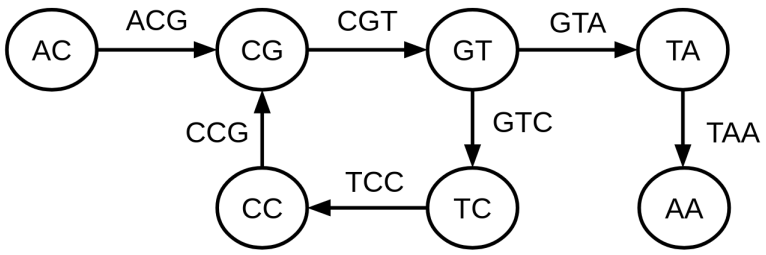

Рис. 1: Граф де Брюйна для строки ACGTCCGTAA с параметром $k = 2$.

Максимальное количество возможных вершин в графе де Брюйна

равно 4^k, а количество рёбер между ними — 4^{k+1}. При небольших значениях k в графе, скорее всего, будут присутствовать почти все варианты вершин, а сам граф будет близок к полносвязному, что даёт плохое представление о геноме. С другой стороны, при больших k граф может распасться на несвязные компоненты. Поэтому, например, ассемблер SPAdes проводит последовательно несколько сборок для разных k и часто останавливается для запутанных геномов и метагеномов на значении 55.

Так как пути в графе де Брюйна образуют контиги, вершины, в которые входит и выходит ровно по одному ребру, можно удалить, объединив соответствующие рёбра в одно, так как с точки зрения обхода графа такие вершины не информативны. Пример такого упрощения можно видеть на рисунке 2.

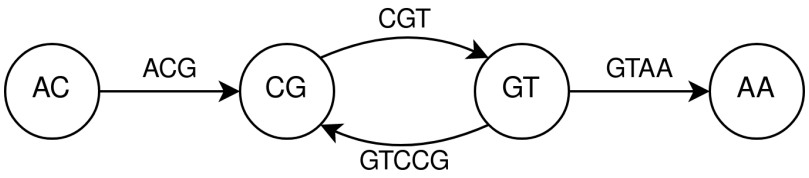

Рис. 2: Граф сборки, полученный в результате упрощения графа с рисунка 1.

Граф, полученный в результате упрощения графа де Брюйна, называется графом сборки. На самом деле, упрощение необходимо по нескольким причинам, которые описаны в работе [35]. В частности, это позволяет удалить рёбра, в которых содержатся ошибки.

2.4. Выравнивание последовательностей

Выравнивание последовательностей — это процесс сопоставления символов одной строки символам другой так, чтобы максимизировать функцию похожести этих последовательностей. Если мы ожидаем, что последовательности небольшие и должны хорошо совпадать по всей свой длине, то в качестве функции похожести можно взять редакционное расстояние. Если известны какие-то особенности последовательностей, то лучше воспользоваться аффинным выравниванием (напри-

мер, можно не штрафовать за большие несовпадающие участки). Находить точное значение функции требует количество действий, пропорциональное произведению длин, что для длинных строк неприемлемо, поэтому ограничиваются лишь приближёнными значениями [14].

Особенностью выравнивания ребер графа сборки на последовательность заключается в том, что ребро может соответствовать одновременно нескольким фрагментам последовательности, а также выравниваться лишь частично.

Помимо этого, существует задача выравнивания последовательности на целый граф сборки [25, 24]. В ней требуется найти *путь* в графе, строковое представление которого максимально похоже на выравниваемую последовательность.

2.5. Инструменты для выравнивания последовательностей

minimap2 [13] разрабатывался в ответ на появление технологий секвенирования, которые создают очень длинные риды (100 kbp в среднем). Его главное преимущество — выравнивание длинных последовательностей ДНК на большие референсные геномы более чем в 30 раз быстрее [13] по сравнению с существующими инструментами, при этом сохраняя достаточно высокий уровень точности.

winnowmap [31] основывается на minimap2 и рассматривает проблему неравномерного распределения k-меров в геноме, уменьшая количество ложно-положительных совпадений. Однако впоследствии его идеи были включены в последние версии minimap2.

GraphAligner [24], в отличие от предыдущих инструментов, предназначен для выравнивания последовательности на целый граф сборки. Он основывается на идее выравнивания последовательности на граф на основе битового параллелизма [23]. Основное преимущество — выравнивание в 12 раз быстрее и в 2 раза точнее по сравнению с существующими инструментами [24]. Помимо этого, полезной возможностью является нахождение не одного лучшего выравнивания, а сразу

нескольких.

2.6. Существующие решения

LoRDEC [26] стал первым инструментом, который использовал граф де Брюйна, построенный из коротких ридов, в качестве индекса для исправления длинных ридов. В LoRDEC длинные риды привязывались к графу, используя общие k-меры. Затем непривязанные подпоследовательности исправлялись с помощью путей, которые похожи на исправляемые подпоследовательности. Многие инструменты для гибридного исправления ошибок в длинных ридах основываются на этом подходе LoRDEC.

Jabba [12], помимо использования подхода LoRDEC, перед построением графа применяет самокоррекцию коротких ридов. Кроме этого, длинные риды привязываются к графу, используя максимально точные совпадения, чтобы использовать k-меры разной длины во время исправления ошибок.

HG-CoLoR [17] также применяет самокоррекцию коротких ридов, а также выравнивает их на длинные риды, чтобы найти перекрытия между собой. Эти перекрытия привязывают риды к графу де Брюйна переменного порядка с учётом разных длин k-меров.

FMLRC [9] индексирует граф де Брюйна, используя многострочное преобразование Барроуза — Уилера (англ. multi-string Burrows-Wheeler Transform) коротких ридов. Это представление занимает мало места в памяти, допускает несколько длин k-меров и неявно сохраняет частоту появления k-меров. Исправление ошибок в FMLRC происходит в два прохода: во время первого используется небольшое значение k для k-меров, в то время как для второго используется k больше, чтобы упростить граф в особо сложных местах.

CoLoRMap [7], в отличии от вышеупомянутых инструментов, строит граф взвешенных выравниваний из сопоставления коротких ридов длинным. Это сопоставление позволяет получить пути в графе, которые максимально похожи на подпоследовательности исправляемых

последовательностей. CoLoRMap использует информацию о парных концах коротких ридов, чтобы перепрыгнуть регионы длинных ридов, которым не сопоставлен ни один короткий рид.

Ratatosk [22] в свою очередь, во первых, некоторым образом раскрашивает вершины графа де Брюйна с помощью коротких и длинных ридов, чтобы выделить существующие пути для исправления. Раскраска графа позволяет уменьшить пространство обхода графа, удаляя химерные пути[1]. Во вторых, длинные риды привязываются к графу, используя как точное совпадение k-меров, так и неточное, что позволяет привязать регионы с большим количеством ошибок. В третьих, граф аннотируется кандидатами на точечные ошибки в один нуклеотид, что позволяет выделить небольшие вариации между похожими участками генома, которые трудно уловить из ошибочных длинных ридов. В четвёртых, коррекция выполняется в два прохода, используя короткие и длинные риды по отдельности.

Для получения большей информации об исправлении ошибок в длинных ридах можно воспользоваться обзорами [18, 8, 33].

2.7. Оценка качества сборки

Для того чтобы понять, насколько результат сборки соответствует реальному геному, существует много метрик. Одни можно вычислить только по результату сборки, другие же требуют наличие референсного генома — некоторого эталонного генома для данного вида организма. Метрики, использующие референсный геном, позволяют намного детальнее оценить качество сборки. Стоит отметить, что собранный геном не должен полностью совпадать с референсным геномом, так как геном исследуемого организма может отличаться от эталонного на 1-2%.

Одной из программ, позволяющих вычислять оба вида метрик, является QUAST [21]. Помимо одиночных геномов он также поддерживает использование метагеномов для оценки качества сборки как отдель-

[1] пути, образованные последовательностями разных организмов

ных геномов в него входящих, так и их объединения [16].

3. Алгоритм

Представленный в данной работе алгоритм является эвристическим. Все приведённые точные значения являются параметрами алгоритма по умолчанию и были получены как близкие к оптимальным при тестировании на различных данных.

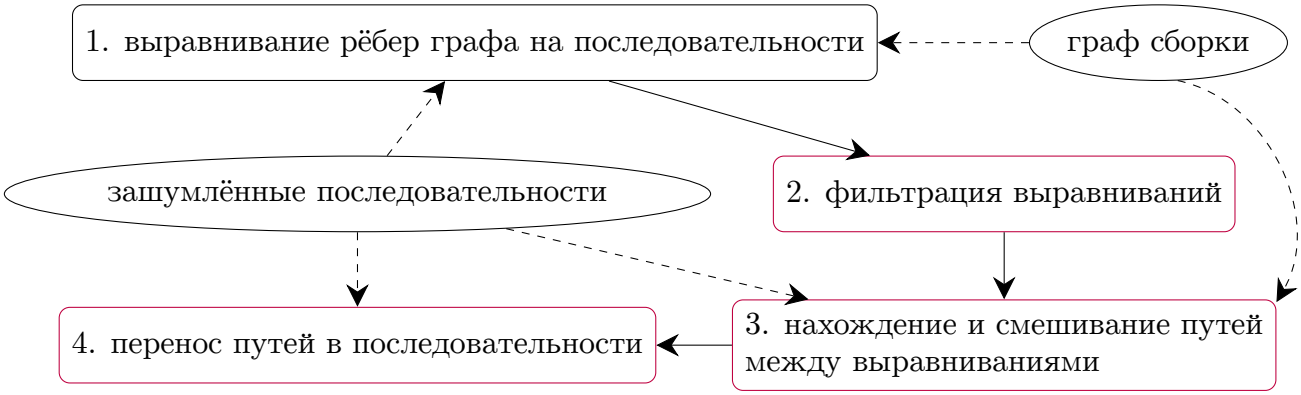

Рис. 3: Алгоритм коррекции. Фиолетовым цветом обозначены стадии, описанные в данной работе. Пунктирными стрелками — использование данных.

Перед началом работы алгоритма коррекции считается, что уже имеются какие-то зашумлённые последовательности, а также уже построен граф сборки с помощью какого-либо ассемблера. Эти последовательности и граф являются входными данными. В конце работы алгоритм выдаёт те же самые последовательности, но с исправленными ошибками.

Алгоритм коррекции состоит из следующей последовательности шагов, изображённых на рисунке 3.

1. Для выравнивания рёбер графа сборки на исправляемые последовательности может быть использован любой из существующих методов. Так как в графе уже присутствуют как прямые рёбра, так и обратно-комплементарные, необходимо получить выравнивания рёбер только в их прямом направлении. Кроме того, так как в графе есть неуникальные рёбра (например, отражают повтор в геноме или принадлежат нескольким организмам в метагеноме), необходимо получить все выравнивания ребра, которые

нашёл выравниватель, а не только одно наилучшее.

2. Найденных выравниваний довольно много, они перекрывают друг друга, и далеко не все из них действительно должны находиться на найденных местах (например, повторы). Поэтому их нужно отфильтровать. Те выравнивания рёбер, что останутся, будем называть *якорями*.

3. Помимо исправления последовательностей непосредственно якорями, нужно исправлять и другие места. Для этого между якорями находятся пути, которые затем смешиваются для получения строки, которая заменит фрагмент последовательности между этими якорями. Будем называть такие строки *строками-заменителями*. Якоря со строками-заменителями между ними образуют *смешанный путь* в графе.

4. Полученные смешанные пути переносятся в исправляемые последовательности.

3.1. Фильтрация выравниваний

Из множества всех найденных выравниваний на рёбра графа нас интересуют те, которые «скорее всего» находятся на правильном месте. Так, например, с увеличением длины ребра, вероятность, что оно будет выравнено на множество мест, уменьшается.

Для выбора якорей используются два фильтра: грубый (англ. rough) и точный (англ. fine). Грубый фильтр работает на этапе чтения выравниваний и оценивает их правильность в отрыве от остальных выравниваний посредством следующих критериев:

- длина выравнивания (а значит и длина ребра) больше минимальной разрешённой длины (minAlignmentLength, по умолчанию 300 символов);

- выравненные фрагмент последовательности и фрагмент ребра совпадают как минимум на minAlignmentIdentity (по умолчанию 50%),

то есть их идентичность[2] (англ. identity) >minAlignmentIdentity%;

- разница длин выравненных фрагмента последовательности и фрагмента ребра не превышает ожидаемого уровня ошибочности сборки (expectedErrorness, по умолчанию 5% для контигов);
- выравнивание является одним из случаев:
  - ребро полностью содержится в последовательности;
  - последовательность полностью содержится в ребре;
  - конец ребра является началом последовательности;
  - начало ребра является концом последовательности[3];
  - выравнялась лишь часть ребра, но она достаточно длинная ($\geq$ longAlignmentLength, по умолчанию 2000 символов), чтобы можно было считать это структурным отличием графа от последовательности;

Точный фильтр оценивает правильность выравнивания на основе уникальности. Выравнивание считается *уникальным*, если его ребро имеет не больше одного выравнивания, которое полностью содержится в какой-либо последовательности. Выравнивание принимается точным фильтром, если оно:

- не короткое ($\geq$shortAlignmentLength, по умолчанию 700) и уникальное;
- короткое ($<$shortAlignmentLength), уникальное и идентичность выше uniqueShortAlignmentIdentity (по умолчанию 70%).
- неуникально и идентичность выше nonuniqueAlignmentIdentity (по умолчанию 90%).

[2]Отношение удвоенного количества совпавших символов к сумме длин сопоставленных подстрок.

[3]Часто «конец/начало ребра/последовательности» не является суффиксом/префиксом из-за наличия ошибок, поэтому необходимо допускать небольшой отступ, например, не больше k символов.

Оставшиеся после фильтрации выравнивания рёбер будем называть *якорями*.

Для каждого якоря определяется уровень достоверности:

1. высокий: якорь уникальный;
2. средний: якорь неуникальный и средней длины ($\geq$middleAlignmentLength, по умолчанию 1000 символов);
3. низкий: оставшиеся якоря;

После фильтрации якоря потенциально всё ещё могут быть взаимоисключающими (то есть пересекаться больше, чем на $k + \varepsilon$, где k — количество символов пересечения смежных рёбер в графе, а ε — несколько символов для возможных ошибок, например, 5). Взаимоисключающие рёбра отбрасываются следующим образом:

- для каждой пары якорей проверяется, конфликтуют ли они. Если да, то:
  - если у них разные уровни достоверности, то на удаление помечается тот, что с меньшим;
  - если уровни достоверности одинаковые и длины якорей отличаются больше, чем в два раза, то помечается тот, что с меньшей;
  - иначе помечаются оба;
- все помеченные якоря разом отбрасываются;

Тем не менее, критериев фильтрации на практике оказывается достаточно, чтобы взаимоисключающих рёбер было не больше всего нескольких единиц из класса с низким уровнем достоверности.

3.2. Пути между якорями

Расположим якоря в порядке встречи их в последовательности. Заполнение пространства вне якорей проходит следующим образом.

- Для каждой пары смежных якорей самого достоверного класса находятся пути в графе, которые отличаются не больше, чем на 2∗expectedErrorness% от длины фрагмента последовательности между этими якорями. Найденные пути затем и смешиваются и формируют строки-заменители.

- Если между двумя якорями не было сформировано строки-заменителя, то в этот промежуток добавляются якоря из следующего класса достоверности и процесс поиска-смешивания-формирования повторяется.

Между якорями ищутся все пути, чтобы на этапе смешивания быть уверенным, что различия между похожими участками генома не будут потеряны. Так как графы устроены довольно сложно, путей даже между не очень удалёнными якорями (несколько тысяч символов) может быть чрезвычайно много (десятки и сотни тысяч), то поиск имеет несколько ограничений, выход за которые, как показывает наша практика, не имеет смысла. Сам поиск проходит в два этапа.

1. С помощью алгоритма Дейкстры в графе помечаются все вершины, через которые существует путь с заданным ограничением сверху на длину. Количество найденных вершин не должно превышать maxDijkstraVertices (по умолчанию 3000).

2. Поиском в глубину по помеченным вершинам выбираются все пути с заданными ограничениями на длину. При этом суммарное количество рекурсивных вызовов не должно превышать maxCallCnt (по умолчанию 3000), а если оно превысило vertexUsageEnableThreshold (по умолчанию 500), то дополнительно проверяется, чтобы в найденном пути было не больше maxVertexUsage (по умолчанию 5) одинаковых рёбер.

Если поиск обнаружил превышение данных ограничений, процесс получения строки-заполнителя между текущей парой якорей прерывается. Эти ограничения выбраны таким образом, чтобы большинство случа-

ев поиска завершалось с большим запасом до их превышения, но на слишком запутанных частях графа они быстро прерывались.

3.3. Смешивание путей

После того, как все пути между якорями найдены, необходимо понять, что из этого может исправить последовательность, а что испортить. Для этого каждый путь выравнивается на фрагмент последовательности. Если расстояние между ними слишком велико, то этот путь отбрасывается. Часть с выравниванием является самой трудоёмкой частью всего пайплайна коррекции и занимает больше 95% времени, поэтому происходит параллельно.

Оставшиеся пути упорядочиваются в порядке возрастания расстояния. Если остался всего один путь, или у первого пути расстояние существенно[4] меньше, чем у второго, то этот путь становится *кандидатом* на строку-заменитель.

Не каждый кандидат становится строкой-заменителем. Чтобы это произошло, он должен отличаться от фрагмента последовательности не больше, чем среднее количество ошибок в текущей последовательности (это значение вычисляется после выбора якорей на основании их среднего значения идентичности).

Довольно часто все найденные пути являются порождением подграфа, который представляет из себя набор «пузырей» (альтернатив из двух подпутей), например, как на рисунке 4.

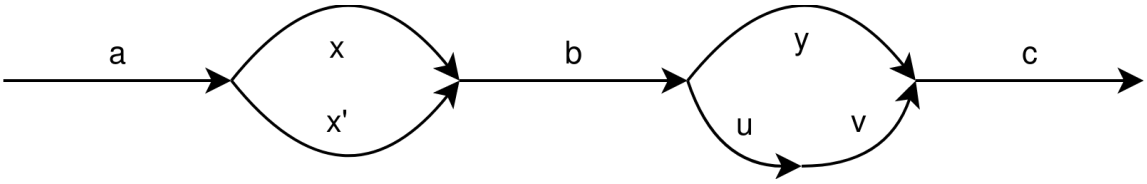

Рис. 4: Подграф, содержащий два «пузыря»: (x, x') и (y, uv).

Пузыри зачастую образуются в результате наличия повторов в геноме, или когда есть похожие фрагменты у двух организмов (например,

[4]Довольно часто лучшие пути имеют расстояние 0-5, что является несущественным отличием, чтобы однозначно сказать, что какой-то путь лучше. Поэтому, помимо процентного соотношения (в два раза), стоит использовать и абсолютный порог (на десять).

двух штаммах бактерии) в метагеноме. Рёбра в них отличаются весьма слабо, поэтому, если разложить такой подграф на два пути (с рисунка 4 это $axbyc$ и $ax'buvc$), можно выравнять эти пути между собой, найти места, которые в них отличаются, и заменить их на фрагменты из исправляемой последовательности. Таким образом вариации между похожими участками генома будут сохранены, а ошибки исправлены. Полученная строка является также лишь кандидатом на строку-заменитель, чтобы избежать случайного внесения большого количества ошибок в последовательность.

Попытка сжатия путей до двух происходит следующим образом. Сначала строится вспомогательный ориентированный граф. Для этого каждое ребро каждого пути представляется парой: id ребра и порядковый номер его вхождения в этот путь. Эти пары становятся вершинами в новом графе. Если два ребра стоят подряд в пути, то в новом графе между соответствующими им парами будет ребро. Пример на рисунке 5.

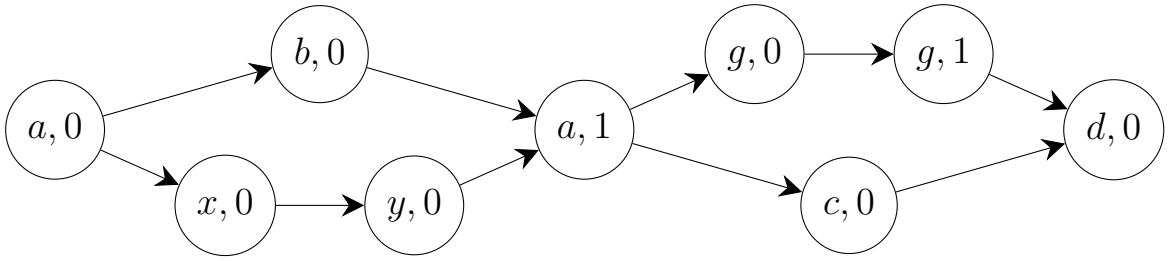

Рис. 5: Пример сжатия четырёх путей (каждая буква является ребром пути): $abacd$, $abaggd$, $axyacd$, $axyaggd$. Первый путь даёт вершины $(a, 0)$, $(b, 0)$, $(a, 1)$, $(c, 0)$, $(d, 0)$ и соединения между ними. Для остальных путей аналогично.

В процессе и после построения вспомогательного графа проверяется его корректность:

- нет циклов и петель;
- начальные и конечные вершины совпадают с начальными и конечными вершинами каждого из путей;
- у каждой общей вершины не больше двух входящих и двух исходящих рёбер;

- у каждой вершины из альтернатив ровно по одному входящему и исходящему ребру;

Если граф получился корректным, то получаем два пути: первый состоит из общих вершин и первых альтернатив, а второй — из общих вершин и вторых альтернатив. Если же корректность нарушена, то сжать пути не удалось и между текущими двумя якорями последовательность исправлена не будет.

Несмотря на то, что есть примеры, когда пути сжать можно, но он этого не делает (например, aaa и aba), на практике этого алгоритма достаточно, чтобы сжать почти все подходящие наборы путей.

3.4. Перенос путей в последовательность

После того, как будут получены строки-заменители, их необходимо разместить обратно в исправляемую последовательность. Для определения фрагмента последовательности, соответствующего данной строке, используются выравнивания с первого шага алгоритма — это диапазон с позиции первого символа первого якоря в составе строки до позиции последнего символа в последнем якоре в составе строки. Стоит отметить, что диапазоны строк-заменителей могут пересекаться вплоть до $k + \varepsilon$, при этом символы в перекрытии могут не совпадать. Чтобы избежать внесения ошибок, эти конфликты решаются следующим образом. Обозначим за a и b фрагменты перекрытия, соответствующие первой и второй строкам-заменителям, а за s соответствующий фрагмент последовательности. Тогда:

1. если $a = b$, то выбирается a;
2. если $a = s$ или $b = s$, то выбирается s;
3. в противном случае также выбирается s;

Для третьего случая можно воспользоваться смешиванием из предыдущего шага, тем самым исправляя общие символы и не меняя несовпадающие, но на практике таких случаев обычно всего несколько штук

на сборку, поэтому если ошибки и останутся неисправленным, то в пренебрежимо малом количестве.

4. Архитектура решения

В данной главе приведена архитектура приложения, которое выполняет непосредственно коррекцию. Все рисунки представлены в виде диаграмм активностей.

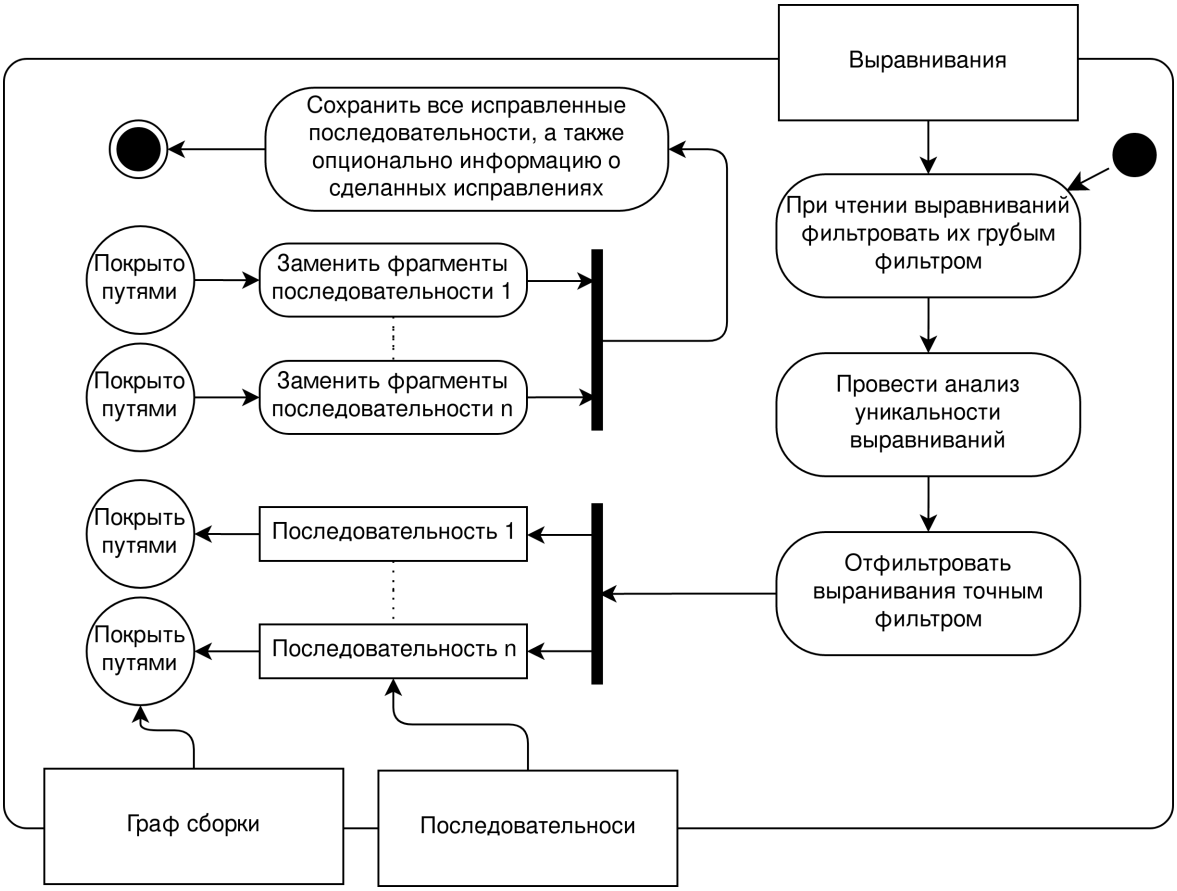

Рис. 6: Диаграмма общей структуры алгоритма

Работа приложения, как показано на рисунке 6, начинается с чтения выравниваний, которые сразу же проходят грубый фильтр, который отфильтровывает бóльшую их часть. Затем определяется, является ли выравнивание уникальным, и применяется точный фильтр. После этого для каждой последовательности независимо находятся пути между её якорями, которые далее заменяют соответствующие им фрагменты. В конце полученные исправленные последовательности сохраняются. При необходимости также сохраняется информация о том, какие фрагменты были исправлены, тип исправления (смешанный путь, чистый путь, перекрытие соседних рёбер, или же само ребро), а также дополни-

тельная информация об исправлении (например, идентичность старого и нового фрагментов).

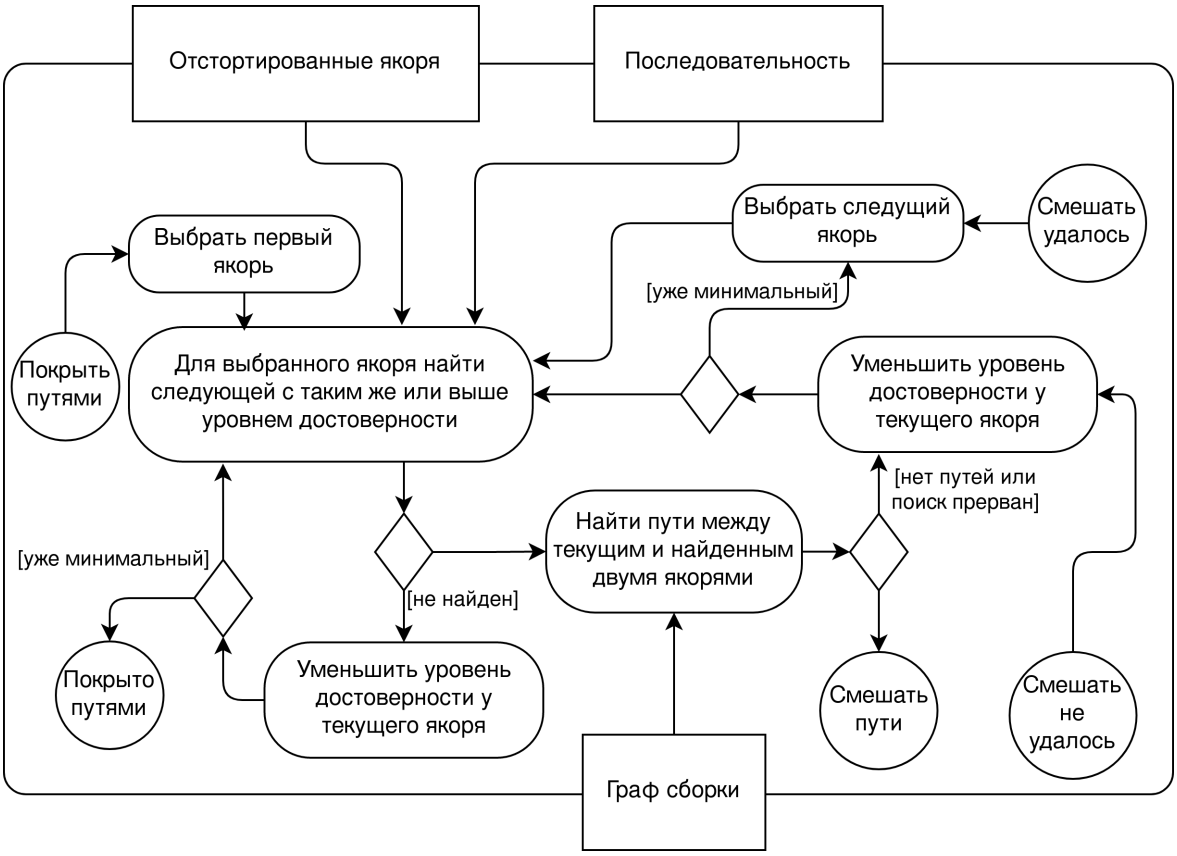

Рис. 7: Диаграмма покрытия последовательности путями

Процесс получения путей между якорями, входом в который является узел «покрыть путями», изображён на рисунке 7. Якоря сортируются в порядке вхождения в последовательность, а затем совершается проход по ним, где для каждого текущего якоря сначала находится его правый сосед с таким же или выше уровнем достоверности. Между этими двумя якорями ищутся все пути в графе, которые затем смешиваются. В случае успеха следующим якорем выбирается правый из текущей пары, тем самым пропускаются якоря меньшего уровня достоверности. Если же какой-то этап завершился неудачей, то для текущего якоря понижается уровень достоверности и поиск происходит заново. Если происходит попытка понизить уже минимальный уровень, то текущий якорь становится последним в текущем пути, а алгоритм переходит к рассмотрению следующего подряд якоря. После того, как был рассмотрен последний якорь, происходит возврат найденных путей через узел

«покрыто путями».

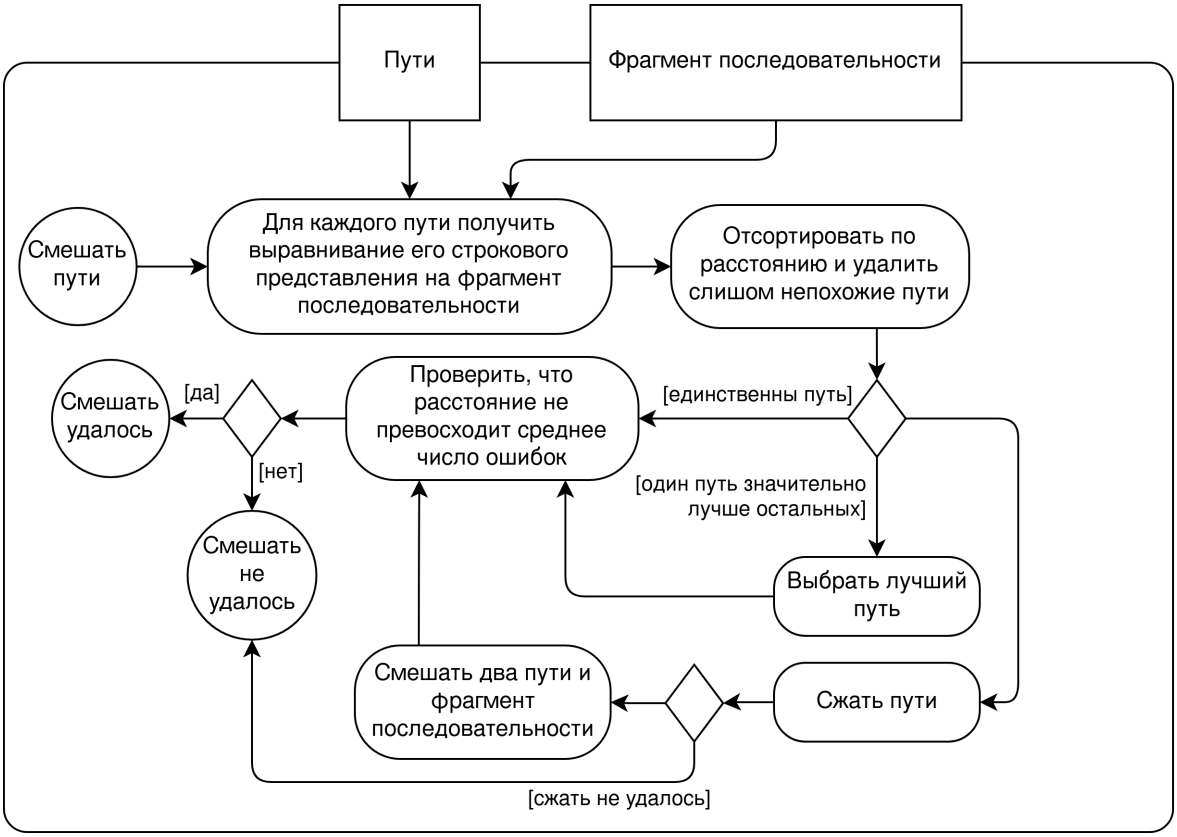

Рис. 8: Диаграмма смешивания путей

Процесс смешивания путей, входом в который является узел «смешать пути», изображён на рисунке 8. Сначала для каждого пути параллельно находится его выравнивание на соответствующий фрагмент последовательности. Если расстояние до него оказывается слишком велико, то путь отбрасывается. Затем, если остался единственный путь, или есть путь, который по расстоянию значительно лучше остальных, то он становится кандитатом на замену. Если же путей много, то происходит попытка их сжать в два пути, а затем смешать с фрагментом последовательности. Результат этого смешения также становится кандидатом на замену. Если кандидат успешно найден, то проверяется, что его расстояние до фрагмента последовательности не превосходит среднее количество ошибок во всей последовательности, а затем происходит возврат через узел «смешать удалось». Если же сжать пути не удалось или же кандидат на замену слишком отличается от последовательности, то в качестве заменителя фрагмента ничего не выбирается

и происходит возврат через узел «смешать не удалось».

5. Наивный алгоритм

Можно заметить, что решение довольно похоже на решение задачи выравнивания последовательностей на граф (т.е. нахождение как можно более длинных путей в графе, которые соответствуют фрагментам последовательностей). Кажется, что отличие лишь в том, что мы дополнительно по-умному смешиваем пути между якорями на уровне нуклеотидов. Поэтому необходимо провести сравнение со следующим алгоритмом:

1. взять существующий выравниватель на граф;
2. взять несколько лучших выравниваний для каждого фрагмента последовательности;
3. смешать найденные пути;

5.1. Кластеризация

Для проверки данного алгоритма был взят выравниватель GraphAligner. Он уже выдаёт несколько лучших найденных выравниваний для каждой области последовательности. Так как пути покрывают похожие, но не идентичные фрагменты, нужно разбить пути на кластера, которые будут покрывать как можно больше длины последовательности, и как можно меньше пересекаться (т.е. конфликтовать) друг с другом. Для этого будем последовательно объединять пути в кластера по следующему критерию: если два пути перекрываются выравниваниями больше, чем на 70%, то они лежат в одном кластере. Как показывает практика, на наших тестах этого критерия было достаточно, чтобы лучшие по идентичности пути в кластерах покрывали последовательность на $95 - 100\%$, все пути в кластере попарно пересекались минимум на 70%, а пути различных кластеров практически не пересекались ($< 1\%$).

5.2. Смешивание

Пути в полученных кластерах смешиваются следующим образом.

1. Выбирается лучший путь в кластере по идентичности с последовательностью.
2. Все остальные пути выравниваются на лучший.
3. По полученным выравниваниям помечаем символы и позиции между символами лучшего пути помечаются недостоверными, если хотя бы в одном выравнивании есть отличие.
4. Недостоверные места заменяются символами из последовательности.

Полученные после смешения строки переносятся обратно способом, описанным в разделе «Перенос путей в последовательность».

6. Апробация

В данной главе будет приведён сравнительный анализ реализации описанного в даной работе алгоритма на симулированных и синтетических метагеномных сообществах как с существующими решениями, так и с наивным алгоритмом.

6.1. Метрики

Сравнение качества коррекции начинается с выравнивания исправленных последовательностей на референсный геном. После этого для общих фрагментов используются следующие метрики:

- *количество замен* — количество **символов** в выравнивании, в которых нуклеотиды последовательности не совпадают с сопоставленными им нуклеотидами референсного генома. Пример на рисунке 10;
- *количество вставок и удалений* — количество **непрерывных участков** в выравнивании, которые присутствуют либо только в последовательностях, либо только в выравненной части референсного генома. Пример на рисунке 10;
- *количество замен на 100 kbp* — среднее количество замен на 100 kbp;
- *количество вставок и удалений на 100 kbp* — среднее количество вставок и удалений на 100 kbp;
- *покрытие генома* — процент нуклеотидов референсного генома, которые покрыты контигами;
- *количество структурных ошибок* — количество релокаций, инверсий, транслокаций. Пример изображён на рисунке 9;

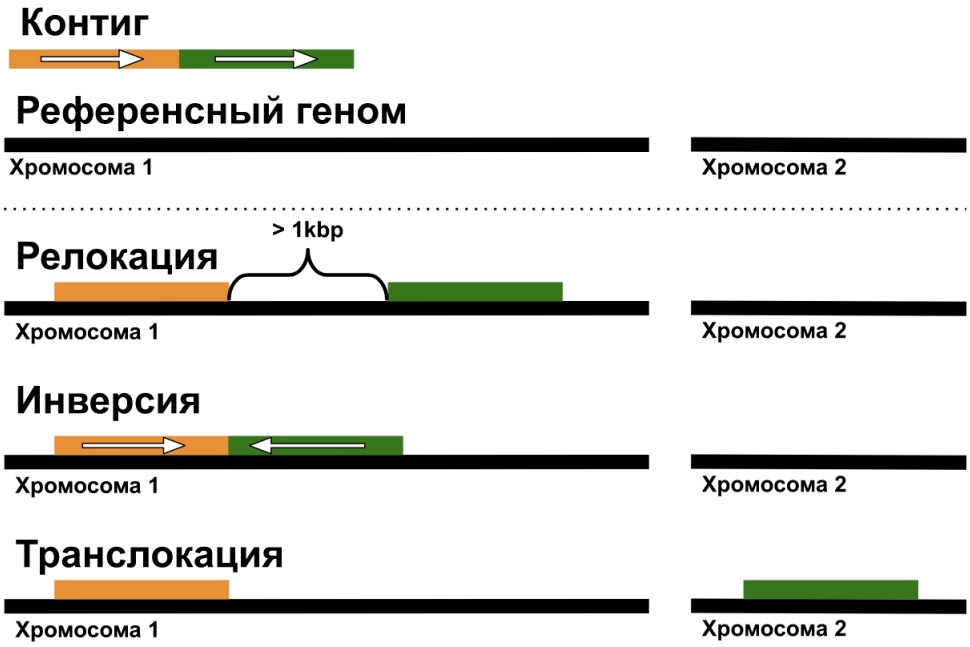

Рис. 9: Дан контиг, состоящий из двух частей, расположение которых при сопоставлении с референсным геномом определяет тип структурной ошибки.

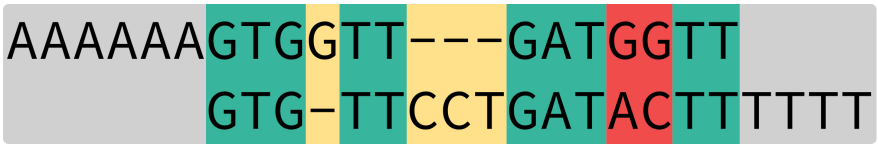

Рис. 10: Пример выравнивания строк AAAAAAGTGGTTGATGGTT и GTGTTCCTGATACTTTTTT. Серым цветом выделены невыравненные фрагменты. Зелёным — совпавшие фрагменты. Красным — замены. Жёлтым — вставки и удаления. Таким образом, в данном выравнивания есть две «замены» и две «вставки и удаления».

6.2. Схема сравнения

На рисунке 11 показана общая схема проведения сборки раздельно из длинных и коротких ридов.

В ходе апробации используются следующие три варианта сравнения:

- применяется коррекция только длинных ридов;
- применяется коррекция только контигов;
- применяется коррекция как длинных ридов, так и контигов;

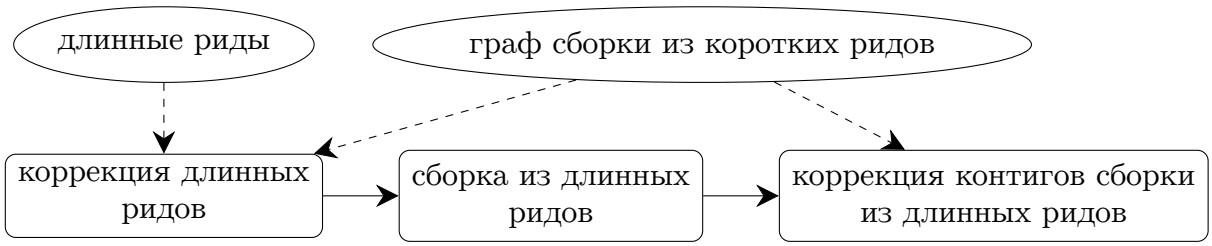

Рис. 11: Процесс раздельной сборки из коротких и длинных ридов начинается с получения графа сборки. Затем, опционально, длинные риды можно подвергнуть коррекции, после чего из них происходит непосредственно сборка генома, контиги которой также можно опционально попытаться исправить.

Среди инструментов, которые исправляют длинные риды при помощи коротких ридов, последним вышедшим (2021 г.) и одним из лучших является Ratatosk [22, 33, 10], поэтому сравнительный анализ будет приведён именно с ним.

Стоит отметить, что реализованный в данной работе алгоритм применяется только для коррекции контигов сборки, так как каждый фрагмент генома в среднем покрывается лишь одним контигом, что неверно для ридов. Из-за этого описанная в разделе «выбор якорей» оценка уникальности выравниваний будет работать некорректно.

Во всех тестах граф сборки из коротких ридов получен ассемблером SPAdes. Для сборки из длинных ридов использовался ассемблер Flye [2]. Инструментом для сбора метрик и сравнения является Quast.

Во всех следующих разделах, кроме «Сравнение с наивным алгоритмом» в таблицах приведены следующие варианты сборки контигов с соответствующими названиями столбцов:

1. [**raw flye**] ни одна коррекция не применяется;
2. [**ratatosk contigs**] ratatosk исправляет только контиги;
3. [**ratatosk reads**] ratatosk исправляет только длинные риды;
4. [**our contigs**] мы исправляем только контиги;
5. [**ratatosk and we**] ratatosk исправляет длинные риды, а мы — контиги;

6. [**ratatosk ratatosk**] ratatosk исправляет как длинные риды, так и контиги.

В таблицах также используется тепловая карта: красным цветом отображён худший результат, синим — лучший, белым — медиана.

6.3. Сравнение с наивным алгоритмом

В данном разделе приводится сравнение двух алгоритмов, представленных в работе. Тестирование проводилось на метагеноме Zymo.

Столбцы на рисунках 12, 13 и 14:

1. [**flye**] контиги Flye без коррекции;
2. [**we**] контиги после коррекции основным алгоритмом;
3. [**mixed**] контиги после коррекции наивным алгоритмом;
4. [**best**] контиги после коррекции наивным алгоритмом с той лишь разницей, что после выбора лучшего пути смешения с остальными не происходит;

	flye	we	mixed	best
Покрытие генома	97.12	97.11	97.12	97.12
Структурные ошибки	69	67	69	69
Замены на 100kbp	109.39	34.02	61.85	60.73
Вставки и удаления на 100kbp	214.49	22.22	20.08	22.61

Рис. 12: Суммарная статистика по всем геномам Zymo для основного и наивного алгоритма

Из таблицы 14 видно, что по вставкам и удалениям наивный алгоритм в паре мест немного обходит основной.

Однако из таблицы 13 видно, что по количеству замен наивный алгоритм не только проигрывает основному, но и в половине случаев вносит ошибок больше, чем исправляет. Особое внимание стоит обратить на бактерию Salmonella Enterica, для неё пути GraphAligner прошлись по похожим рёбрам другой бактерии, что привело к всплеску замен.

	flye	we	mixed	best
Bacillus subtilis	2124	207	269	402
Enterococcus faecalis	593	547	667	667
Escherichia coli	13012	1401	10740	10194
Lactobacillus fermentum	412	104	489	489
Listeria monocytogenes	149	64	840	840
Pseudomonas aeruginosa	1077	606	937	937
Saccharomyces cerevisiae	15441	10943	10670	10602
Salmonella enterica	12816	352	199995	200881
Staphylococcus aureus	891	157	1002	1002

Рис. 13: Замены в метагеноме Zymo для основного и наивного алгоритма

	flye	we	mixed	best
Bacillus subtilis	10111	181	91	791
Enterococcus faecalis	8393	562	82	82
Escherichia coli	9213	484	812	1176
Lactobacillus fermentum	3985	33	52	52
Listeria monocytogenes	7181	78	82	82
Pseudomonas aeruginosa	4928	42	70	70
Saccharomyces cerevisiae	32845	7739	6985	7000
Salmonella enterica	10119	166	10433	10643
Staphylococcus aureus	4643	155	106	106

Рис. 14: Вставки и удаления в метагеноме Zymo для основного и наивного алгоритма

6.4. Симулированные метагеномы

В текущем разделе описан результат сравнения на симулированном метагеноме из 20 организмов.

Из таблицы 15 видно, что покрытие генома практически не изменилось, а значит во всех случаях была произведена именно коррекция. Тем не менее, при исправлении длинных ридов Ratatosk внёс ошибки, которые привели к увеличению количества структурных ошибок при сборке контигов. Последующая коррекция контигов с использованием Ratatosk усугубило проблему, в то время как описанный в данной рабо-

	raw flye	ratatosk contigs	ratatosk reads	our contigs	ratatosk and we	ratatosk ratatosk
Покрытие генома	83.21	83.15	82.52	83.22	82.52	82.47
Структурные ошибки	34	34	50	34	40	55
Замены на 100kbp	369.97	105.08	112.19	121.67	67.22	101.24
Вставки и удаления на 100kbp	688.01	194.1	199.81	175.13	87.29	176.19

Рис. 15: Суммарная статистика по всем геномам симуляции

	raw flye	ratatosk contigs	ratatosk reads	our contigs	ratatosk and we	ratatosk ratatosk
Acinetobacter baumannii	32183	2039	2025	5711	538	1696
Bacillus subtilis	18235	7466	8615	6438	2361	7864
Brevibacterium linens	28791	1572	1556	3288	300	975
Clostridium perfringens	3	17	8	76	81	2337
Coraliomargarita akajimensis	129	4	0	16	14	0
Corynebacterium accolens	556	653	579	1862	1754	675
Corynebacterium matruchotii	16454	12347	12104	11891	8362	11091
Cutibacterium acnes	1	1	458	2	459	458
Echinicola vietnamensis	38140	8349	10101	8866	2146	8077
Escherichia coli	2333	170	183	606	308	127
Lactobacillus gasseri	4439	236	203	447	54	196
Listeria monocytogenes	22084	688	825	3136	443	2064
Meiothermus silvanus	20035	16169	16793	16767	14208	15748
Micrococcus luteus	21060	989	2014	1846	653	1175
Nocardiopsis dassonvillei	10154	6045	6559	6018	4974	5918
Olsenella uli	0	0	0	0	0	0
Pseudomonas stutzeri	11127	8223	8040	7361	4591	7230
Segniliparus rugosus	208	203	173	249	270	186
Spirochaeta thermophila	130	14	14	307	266	13
Terriglobus roseus	388	87	105	521	439	88

Рис. 16: Замены в симулированном метагеноме

те алгоритм смог исправить более 60% внесённых структурных ошибок.

Представленный алгоритм в одиночку смог сократить количество замен в 3 раза, а количество вставок и удалений — в 4 раза. По сравнению с коррекцией контигов с использованием Ratatosk, это на 15%

	raw flye	ratatosk contigs	ratatosk reads	our contigs	ratatosk and we	ratatosk ratatosk
Acinetobacter baumannii	53628	3533	3200	8424	491	2617
Bacillus subtilis	27656	11196	11989	8179	2060	10178
Brevibacterium linens	58667	1819	1896	5587	242	1335
Clostridium perfringens	34	12	3	11	11	460
Coraliomargarita akajimensis	558	5	2	6	3	2
Corynebacterium accolens	1049	964	1104	584	627	1188
Corynebacterium matruchotii	31387	23089	22229	17612	11817	20002
Cutibacterium acnes	4	3	84	4	84	84
Echinicola vietnamensis	56828	11631	13327	11462	1836	10660
Escherichia coli	7459	161	187	415	136	41
Lactobacillus gasseri	9337	401	293	771	19	233
Listeria monocytogenes	36511	688	690	4626	176	818
Meiothermus silvanus	40528	32756	32696	28852	23976	30466
Micrococcus luteus	42946	303	698	1955	115	232
Nocardiopsis dassonvillei	30458	16456	17866	6586	4279	15462
Olsenella uli	0	0	0	1	1	0
Pseudomonas stutzeri	19637	14255	14193	10823	6760	12789
Segniliparus rugosus	59	50	67	56	47	43
Spirochaeta thermophila	743	180	23	187	34	27
Terriglobus roseus	1757	587	283	535	103	341

Рис. 17: Вставки и удаления в симулированном метагеноме

хуже для замен, но на 10% лучше для вставок и удалений.

Комбинирование коррекций здесь является безусловным фаворитом, которое снизило количество замен в 5.5 раз, а количество вставок и удалений — почти в 8 раз, что примерно в два раза лучше, чем аналогичный сценарий с использованием только Ratatosk.

6.5. Синтетические метагеномные сообщества

Синтетическое метагеномное сообщество — это сообщество, организмы которого были отдельно друг от друга выращены и просеквенированы для получения точного референсного генома. Однако для получения ридов метагенома секвенируется смесь из них. В данном

разделе будут представлены результаты для двух синтетических метагеномных сообществ: Bmock12 и Zymo.

Bmock12

Bmock12 — это синтетическое сообщество, состоящее из смеси одиннадцати бактериальных организмов.

	raw flye	ratatosk contigs	ratatosk reads	our contigs	ratatosk and we	ratatosk ratatosk
Покрытие генома	62.40	62.36	64.43	62.41	64.45	64.40
Структурные ошибки	72	77	129	72	99	145
Замены на 100kbp	342.69	127.27	203.28	144.39	166.3	180.82
Вставки и удаления на 100kbp	654.71	61.02	88.76	41.56	28.6	46.28

Рис. 18: Суммарная статистика по всем геномам в Bmock12

	raw flye	ratatosk contigs	ratatosk reads	our contigs	ratatosk and we	ratatosk ratatosk
Arcobacter sp. ES.032	19922	330	845	779	524	368
Cohaesibacter sp. ES.047	7419	540	562	711	607	395
Halomonas sp. HL 4	18511	19529	18699	29470	31373	18743
Halomonas sp. HL 93	30072	29236	26700	17083	15951	26633
Marinobacter sp. LV10MA510 1	12711	1094	1449	6008	5230	707
Marinobacter sp. LV10R510 8	1159	109	210	362	295	65
Micromonospora echinaurantiaca	60387	63956	106821	59598	93314	104766
Micromonospora echinofusca	43666	32941	62149	28916	53196	59785
Muricauda sp. ES.050	1211	245	128	78	55	44
Propionibacteriaceae bacterium	10076	208	3870	111	229	3264
Psychrobacter sp. LV10R520 6	512	76	57	402	329	35

Рис. 19: Замены в метагеноме Bmock12

Из таблицы 18 видно, что покрытие генома изменилось незначительно, а значит во всех случаях была произведена именно коррекция. Однако при исправлении длинных ридов Ratatosk внёс ошибки, которые привели к увеличению количества структурных ошибок при

	raw flye	ratatosk contigs	ratatosk reads	our contigs	ratatosk and we	ratatosk ratatosk
Arcobacter sp. ES.032	53541	476	752	551	65	146
Cohaesibacter sp. ES.047	17229	633	509	288	187	456
Halomonas sp. HL 4	8697	6730	6425	1485	1656	4799
Halomonas sp. HL 93	9406	7096	5844	1285	1358	4645
Marinobacter sp. LV10MA510 1	35036	897	912	1779	624	456
Marinobacter sp. LV10R510 8	9320	927	360	203	109	277
Micromonospora echinaurantiaca	30953	9715	21190	11951	12969	14422
Micromonospora echinofusca	44304	6649	18373	8600	9163	9701
Muricauda sp. ES.050	13965	3117	823	34	23	353
Propionibacteriaceae bacterium	17483	148	2371	96	94	876
Psychrobacter sp. LV10R520 6	7217	942	378	146	114	315

Рис. 20: Вставки и удаления в метагеноме Bmock12

сборке контигов. Последующая коррекция контигов с использованием Ratatosk усугубило проблему, в то время как описанный в данной работе алгоритм смог исправить более 60% внесённых структурных ошибок.

Zymo

ZymoEven — это синтетическое сообщество, состоящее из смеси девяти бактериальных и дрожжевых организмов. Концентрация как длинных, так и коротких ридов равномерно распределена по всем геномам.

	raw flye	ratatosk contigs	ratatosk reads	our contigs	ratatosk and we	ratatosk ratatosk
Покрытие генома	97.12	97.09	96.93	97.11	96.91	96.98
Структурные ошибки	69	68	72	67	71	66
Замены на 100kbp	109.39	60.51	42.98	34.02	34.58	36.09
Вставки и удаления на 100kbp	214.49	47.68	21.3	22.22	18.07	21.64

Рис. 21: Суммарная статистика по всем геномам в Zymo

Из рисунка 21 видно, что покрытие генома и количество структурных ошибок поменялось несущественно, а значит была произведена

	raw flye	ratatosk contigs	ratatosk reads	our contigs	ratatosk and we	ratatosk ratatosk
Bacillus subtilis	2124	1031	220	207	193	151
Enterococcus faecalis	593	701	310	547	324	508
Escherichia coli	13012	3072	1910	1401	2930	1101
Lactobacillus fermentum	412	377	318	104	96	325
Listeria monocytogenes	149	92	80	64	81	146
Pseudomonas aeruginosa	1077	638	608	606	612	606
Saccharomyces cerevisiae	15441	14093	13172	10943	9902	11985
Salmonella enterica	12816	5005	1584	352	271	1540
Staphylococcus aureus	891	582	46	157	89	37

Рис. 22: Замены в метагеноме Zymo

	raw flye	ratatosk contigs	ratatosk reads	our contigs	ratatosk and we	ratatosk ratatosk
Bacillus subtilis	10111	2165	248	181	138	400
Enterococcus faecalis	8393	3271	427	562	257	534
Escherichia coli	9213	970	183	484	243	166
Lactobacillus fermentum	3985	1881	332	33	19	378
Listeria monocytogenes	7181	791	57	78	33	84
Pseudomonas aeruginosa	4928	277	47	42	30	44
Saccharomyces cerevisiae	32845	8753	7390	7739	6767	6857
Salmonella enterica	10119	1358	263	166	76	377
Staphylococcus aureus	4643	858	110	155	51	127

Рис. 23: Вставки и удаления в метагеноме Zymo

именно коррекция. Помимо этого, описанный в данной работе алгоритм при коррекции контигов уменьшает количество замен примерно в 3 раза, а количество вставок и удалений — в 10 раз. По количеству замен это на 20% лучше, чем коррекция длинных ридов Ratatosk, и лишь на 4% хуже по вставкам и удалениям.

Комбинирование коррекций является лучшим результатом, при котором удалось количество вставок и удалений уменьшить в 12 раз, что превосходит аналогичные усилия одного лишь Ratatosk на 16%.

Заключение

В ходе данной работы были получены следующие результаты.

- Сформированы критерии фильтрации выравниваний рёбер графа на последовательности.
- Разработан алгоритм исправления ошибок за пределами выравненных рёбер с помощью путей в графе.
- Разработан алгоритм переноса полученных путей в графе обратно в последовательности.
- Итоговый алгоритм реализован в виде отдельного инструмента.
  - Реализация выполнена на языке C++ и является подпроектом для ассемблера SPAdes.
  - Исходный код SPAdes доступен по ссылке: `https://github.com/ablab/spades/`. Реализованный инструмент будет доступен начиная с версии 3.17.
- Проведена апробация алгоритма на симулированных и реальных данных.
  - Исправление метагеномных сборок низкого качества даёт сравнимый и лучший результат по сравнению с коррекцией ридов и сборок существующими решениями.
  - Кооперация с существующими решениями даёт существенно лучший результат.

Глоссарий

Ассемблер — программное обеспечение для получение контигов и скаффолдов из ридов.

Граф де Брюйна — граф с параметром k, который строится по следующим принципам: берётся набор входных строк и выделяется из них множество всевозможных подстрок длины k+1. Тогда k+1-мер — это ребро, ведущее из своего префикса длины k в суффикс длины k.

Граф сборки — в этой работе: граф, полученный в результате упрощения графа де Брюйна.

K-мер — последовательность длины K.

Референсный геном — эталонный геном данного вида организма.

Риды — множество фрагментов ДНК, которые получаются в результате секвенирования.

Контиги — восстановленные части ДНК из ридов.

Скаффолд — объединение нескольких контигов с известным расстоянием между ними, но неизвестными самими нуклеотидами.

Список литературы

[1] Simpson Jared T, Wong Kim, Jackman Shaun D, Schein Jacqueline E, Jones Steven JM, and Birol Inanç. ABySS: a parallel assembler for short read sequence data // Genome research. — 2009. — Vol. 19, no. 6. — P. 1117–1123.

[2] Kolmogorov M., Yuan J., Lin Y., and Pevzner P. A. Assembly of long, error-prone reads using repeat graphs. // Nature Biotechnology. — 2019. — Vol. 37. — P. 540–546.

[3] Bankevich A. et al. SPAdes: A New Genome Assembly Algorithmand Its Applications to Single-Cell Sequencing. // Journal of Computational Biology. — 2012. — Vol. 19, no. 5. — P. 455–477.

[4] Boisvert Sébastien, Laviolette François, and Corbeil Jacques. Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies // Journal of computational biology. — 2010. — Vol. 17, no. 11. — P. 1519–1533.

[5] Buermans H. P. J. and den Dunnen J. T. Next generation sequencing technology: Advances and applications. // Biochimica et Biophysica Acta (BBA) - Molecular Basis of Disease. — 2014. — Vol. 1842, no. 10. — P. 1932–1941.

[6] Collins Andrew. The Challenge of Genome Sequence Assembly // The Open Bioinformatics Journal. — 2018. — Vol. 11, no. 1.

[7] Haghshenas Ehsan, Hach Faraz, Sahinalp S Cenk, and Chauve Cedric. Colormap: Correcting long reads by mapping short reads // Bioinformatics. — 2016. — Vol. 32, no. 17. — P. i545–i551.

[8] Zook Justin M, Catoe David, McDaniel Jennifer, Vang Lindsay, Spies Noah, Sidow Arend, Weng Ziming, Liu Yuling, Mason Christopher E, Alexander Noah, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials // Scientific data. — 2016. — Vol. 3, no. 1. — P. 1–26.

[9] Wang Jeremy R, Holt James, McMillan Leonard, and Jones Corbin D. FMLRC: Hybrid long read error correction using an FM-index // BMC bioinformatics. — 2018. — Vol. 19, no. 1. — P. 1–11.

[10] Fu Shuhua, Wang Anqi, and Au Kin Fai. A comparative evaluation of hybrid error correction methods for error-prone long reads // Genome biology. — 2019. — Vol. 20, no. 1. — P. 1–17.

[11] Peng Yu, Leung Henry CM, Yiu Siu-Ming, and Chin Francis YL. IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth // Bioinformatics. — 2012. — Vol. 28, no. 11. — P. 1420–1428.

[12] Miclotte Giles, Heydari Mahdi, Demeester Piet, Rombauts Stephane, Van de Peer Yves, Audenaert Pieter, and Fostier Jan. Jabba: hybrid error correction for long sequencing reads // Algorithms for Molecular Biology. — 2016. — Vol. 11, no. 1. — P. 1–12.

[13] Li Heng. Minimap2: pairwise alignment for nucleotide sequences // Bioinformatics. — 2018. — Vol. 34, no. 18. — P. 3094–3100.

[14] Likic Vladimir. The Needleman-Wunsch algorithm for sequence alignment // Lecture given at the 7th Melbourne Bioinformatics Course, Bi021 Molecular Science and Biotechnology Institute, University of Melbourne. — 2008. — P. 1–46.

[15] Zimin Aleksey V, Marçais Guillaume, Puiu Daniela, Roberts Michael, Salzberg Steven L, and Yorke James A. The MaSuRCA genome assembler // Bioinformatics. — 2013. — Vol. 29, no. 21. — P. 2669–2677.

[16] Mikheenko Alla, Saveliev Vladislav, and Gurevich Alexey. MetaQUAST: evaluation of metagenome assemblies // Bioinformatics. — 2016. — Vol. 32, no. 7. — P. 1088–1090.

[17] Morisse Pierre, Lecroq Thierry, and Lefebvre Arnaud. Hybrid correction of highly noisy long reads using a variable-order de Bruijn graph // Bioinformatics. — 2018. — Vol. 34, no. 24. — P. 4213–4222.

[18] Morisse Pierre, Lecroq Thierry, and Lefebvre Arnaud. Long-read error correction: a survey and qualitative comparison // BioRxiv. — 2020.

[19] Alic Andy S, Ruzafa David, Dopazo Joaquin, and Blanquer Ignacio. Objective review of de novo stand-alone error correction methods for NGS data // Wiley Interdisciplinary Reviews: Computational Molecular Science. — 2016. — Vol. 6, no. 2. — P. 111–146.

[20] Pevzner P. A., Tang H., and Waterman M. S. An Eulerian path approach to DNA fragment assembly. // Proceedings of the National Academy of Sciences. — 2001. — Vol. 98, no. 17. — P. 9748–9753.

[21] Gurevich A., Saveliev V., Vyahhi N., and Tesler G. QUAST: quality assessment tool for genome assemblies. // Bioinformatics. — 2013. — Vol. 29, no. 8. — P. 1072–1075.

[22] Holley Guillaume, Beyter Doruk, Ingimundardottir Helga, Møller Peter L, Kristmundsdottir Snædis, Eggertsson Hannes P, and Halldorsson Bjarni V. Ratatosk: hybrid error correction of long reads enables accurate variant calling and assembly // Genome Biology. — 2021. — Vol. 22, no. 1. — P. 1–22.

[23] Rautiainen Mikko, Mäkinen Veli, and Marschall Tobias. Bit-parallel sequence-to-graph alignment // Bioinformatics. — 2019. — Vol. 35, no. 19. — P. 3599–3607.

[24] Rautiainen Mikko and Marschall Tobias. GraphAligner: rapid and versatile sequence-to-graph alignment // Genome biology. — 2020. — Vol. 21, no. 1. — P. 1–28.

[25] Dvorkina Tatiana, Antipov Dmitry, Korobeynikov Anton, and Nurk Sergey. SPAligner: alignment of long diverged molecular sequences to assembly graphs // BioRxiv. — 2019. — P. 744755.

[26] Salmela Leena and Rivals Eric. LoRDEC: accurate and efficient long read error correction // Bioinformatics. — 2014. — Vol. 30, no. 24. — P. 3506–3514.

[27] Sohn Jang-il and Nam Jin-Wu. The present and future of de novo whole-genome assembly // Briefings in bioinformatics. — 2018. — Vol. 19, no. 1. — P. 23–40.

[28] Souvorov Alexandre, Agarwala Richa, and Lipman David J. SKESA: strategic k-mer extension for scrupulous assemblies // Genome biology. — 2018. — Vol. 19, no. 1. — P. 1–13.

[29] Wick Ryan R, Judd Louise M, Gorrie Claire L, and Holt Kathryn E. Unicycler: resolving bacterial genome assemblies from short and long sequencing reads // PLoS computational biology. — 2017. — Vol. 13, no. 6. — P. e1005595.

[30] Watson J. D. and Crick F. H. C. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. // Nature. — 1953. — Vol. 171, no. 4356. — P. 737–738.

[31] Jain Chirag, Rhie Arang, Zhang Haowen, Chu Claudia, Walenz Brian P, Koren Sergey, and Phillippy Adam M. Weighted minimizer sampling improves long read mapping // Bioinformatics. — 2020. — Vol. 36, no. Supplement_1. — P. i111–i118.

[32] Zerbino Daniel R and Birney Ewan. Velvet: algorithms for de novo short read assembly using de Bruijn graphs // Genome research. — 2008. — Vol. 18, no. 5. — P. 821–829.

[33] Zhang Haowen, Jain Chirag, and Aluru Srinivas. A comprehensive evaluation of long read error correction methods // BMC genomics. — 2020. — Vol. 21, no. 6. — P. 1–15.

[34] Antipov Dmitry, Korobeynikov Anton, McLean Jeffrey S, and Pevzner Pavel A. hybridSPAdes: an algorithm for hybrid assembly of short and long reads // Bioinformatics. — 2016. — Vol. 32, no. 7. — P. 1009–1015.

[35] Пржибельский Андрей Дмитриевич. Разработка алгоритмов для сборки геномов и транскриптомов. — 2020.