ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ И НАУКИ
САНКТ-ПЕТЕРБУРГСКИЙ АКАДЕМИЧЕСКИЙ УНИВЕРСИТЕТ —
НАУЧНО-ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР НАНОТЕХНОЛОГИЙ
РОССИЙСКОЙ АКАДЕМИИ НАУК

На правах рукописи

Диссертация допущена к защите

Зав. кафедрой

_____А.В.Омельченко

“ ”_____2014 г.

ДИССЕРТАЦИЯ

НА СОИСКАНИЕ УЧЕНОЙ СТЕПЕНИ

МАГИСТРА

Тема: **Ragout – алгоритм для сборки генома с использованием нескольких референсных последовательностей**

Направление: 010900.68 – Прикладные математика и физика

Выполнил студент *М.А.Колмогоров*
(подпись)

Руководитель:
Ph.D. *Ш.К.Фам*
(подпись)

Рецензент:
Ph.D., гл. науч. сотр. *А.Л.Лапидус*
(подпись)

Санкт-Петербург
2014 г.

РЕФЕРАТ

С. 30, рис. 7, таб. 6.

В данной работе представлен алгоритм Ragout, предназначенный для референсной сборки генома. В отличие от существующих решений, предложенный алгоритм использует информацию из нескольких референсных геномов одновременно, что в некоторых ситуациях значительно улучшает качество сборки. Также, мы используем синтенные блоки несольких размеров и граф перекрытий для уменьшения числа пропусков. Сравнение с существующими алгоритмами демонстрирует преимущество Ragout на сложных наборах данных. Алгоритм реализован в виде программы на Python/C++ и находится в свободном доступе по адресу: *http://fenderglass.github.io/Ragout.*

БЛАГОДАРНОСТИ

В первую очередь, я хотел бы поблагодарить Шона Фама, который является как моим научным руководителем, так и руководителем всего проекта Ragout. Я глубоко признателен Алле Лапидус, моему рецензенту. Также, я благодарен Павлу Авдееву, который работал над Ragout в рамках своего научного проекта и существенно улучшил некоторые алгоритмы обработки графа перекрытий. Дмитрий Мелешко также оказал помощь в разработке и реализации подходов для построения графа перекрытий. Я благодарен Николаю Вяххи за полезные советы и плодотворные дискуссии.

ВВЕДЕНИЕ

Задача прочтения и последующей сборки генома является одной из важнейших задач как биоинформатики, так и молекулярной биологии. Практически каждое исследование включает в себя анализ геномных последовательностей.

Современные технологии позволяют прочитать лишь короткие фрагменты ДНК – *риды (reads)*, длиной в несколько сотен пар оснований. Для прочтения генома целиком, используется так называемый *Полногеномный метод дробовика (Whole genome shoutgun)*, когда исходная ДНК разбивается на короткие фрагменты, которые прочитываются секвенаторами, а затем объединяются в более крупные участки генома с помощью компьютерных программ. Такие программы называют *ассемблерами (assemblers, сборщики)*. Однако, в большинстве случаев, ассемблеры не могут восстановить весь геном целиком. Вместо этого, они выводят так называемые *контиги (contigs)* – цельные участки генома длиной до нескольких тысяч пар оснований, которые в совокупности целиком покрывают исследуемый геном и в некоторой своей неизвестной перестановке дают верную последовательность ДНК.

В данной работе исследуется возможность объединения описанных выше контигов в так называемые *скэффолды (scaffolds)* – упорядоченные списки контигов – с использованием информации из доступных геномов близкородственных организмов (как то: близкие виды или штаммы). В отличие от существующих алгоритмов, мы рассматриваем возможность использования одновременно нескольких таких *референсных последовательностей*, что позволяет решать описанную выше задачу более точно.

Работа состоит из трех глав. В первой приводится обзор предметной области и существующих методов. Во второй описана постановка задачи и оригинальные подходы для ее решения. В третьей главе мы описываем эксперименты, в ходе которых мы тестируем наши алгоритмы и сравниваем их с существующими. Текст данной работы основан на тексте оригинальной статьи [10], принятой на конференцию ISMB 2014.

ГЛАВА 1

Обзор методов и постановка задачи

Развитие технологий секвенирования нового поколения с использованием коротких ридов открыло множество новых возможностей для экспериментов, но вместе с тем возникли и новые трудности. Даже для сравнительно небольших бактериальных геномов, их сборки из коротких ридов обычно состоят из сотен контигов. Для улучшения качества сборки, последние исследования предлагают использование длинных ридов полученных с помощью технологий PacBio, либо «прыгающих» библиотек парных ридов с большим расстоянием вставки, что помогает объединять контиги в более длинные скэффолды, а также используется в ассемблерах для разрешения неоднозначностей при сборке [3, 6, 11]. Однако, популярность этих технологий в современных исследованиях на данный момент ограничена их высокой стоимостью и большим количеством ошибок.

В качестве альтернативного подхода можно использовать уже собранные геномы родственных организмов для «подсказок» ассемблеру, такой метод называется *референсной сборкой (reference-assisted assembly)*. Первые программы для референсной сборки упорядочивали контиги, выравнивая их на близкородственный геном и выстраивая в соответствующем порядке. Хотя такой метод до сих про широко используется, иногда он приводит к ошибочным результатам, когда собираемый и референсный геномы содержат структурные вариации по отношению друг к другу. В попытках разрешить эту проблему, авторы [8] сформулировали *задачу упорядочивания контигов*, которая предполагает объединение контигов в скэффолды таким образом, чтобы DCJ расстояние (2-break расстояние) [1, 5] между референсным геномом и скэффолдами было минимально. Эта идея была в дальнейшем реализована в нескольких программных инструментах для референсной сборки [16, 17]. К сожалению, алгоритмы основанные на данном подходе также выдают ошибочные результаты, когда между двумя геномами существуют крупные перестройки. Возникает вопрос – всегда ли достаточно одного референсного генома для получения безошибочной сборки?

Недавно Ким с соавторами [9] сделали важный шаг в задаче реконструкции собираемого генома и предложили алгоритм RACA для референсной сборки. В отличие от предыдущих алгоритмов использующих только одну референсную последовательность, RACA использует несколько родственных геномов, один из

которых является референсным, а остальные - *внешними (outgroup)*. При таком подходе информация из внешних геномов также используется при восстановлении порядка контигов.

Хотя алгоритм RACA продемонстрировал значительные улучшения в задаче референсной сборки, он продолжает обладать некоторыми недостатками. Во-первых, хотя RACA использует информацию из нескольких внешних геномов, процедура восстановления продолжает в большой степени полагаться только на единственный референсный геном. Как и другие алгоритмы, анализирующие геномные перестройки, RACA представляет геномы как последовательности синтенных блоков. Однако, для нахождения такого разложения RACA использует попарные выравнивания всех входных геномов на референс. Такой подход в некоторых случаях может не найти синтенные блоки [15], а также возникает вопрос - что делать с контигами, которые не имеют выравнивания на референс? Во-вторых, в отличие от декомпозиции цельных геномов на синтенные блоки, такая декомпозиция для геномов, представленных набором контигов, приводит к фрагментации самих блоков, т.к. контиги имеют разную длину. Таким образом, возникает проблема выбора масштаба для декомпозиции на синтенные блоки. Если рассматривать крупный размер блоков, мы исключаем из анализа короткие контиги, что приведет к *пропускам (gaps)* в сборке. С другой стороны, анализ блоков малого размера может оказаться значительно труднее, т.к. такие синтенные блоки чаще подвержены геномным перестройкам, а также выше шанс их ошибочного (не)распознавания алгоритмами для их поиска. Для получения скэффолдов высокого качества нам необходимо разрешить эту дилемму.

В данной работе мы представляем Ragout (Reference-Assisted Genome Ordering UTility) - алгоритм для референсной сборки генома. Ragout использует несколько референсных геномов родственных видов/штаммов, а также их эволюционную взаимосвязь (фиолгенетическое дерево). В отличие от большинства существующих алгоритмов, Ragout использует синтенные блоки разного размера, а также информацию из *графа перекрытий (overlap graph)* для улучшения качества скэффолдов. Мы покажем, что при наличии нескольких референсных геномов, качество сборки с использованием Ragout выше, чем с остальными существующими алгоритмами. Программная реализация алгоритма находится в свободном доступе по адресу: *http://fenderglass.github.io/Ragout*

ГЛАВА 2

Методы

2.1 Обзор алгоритма

Ragout принимает на вход:

- Начальную сборку (набор контигов)
- Набор референсных геномов
- Фиолгенетическое дерево для всех геномов (включая собираемый)

и выводит скэффолды в виде упорядоченных списков контигов.

В начала работы алгоритм осуществляет декомпозицию данных геномов на синтенные блоки с помощью программы Sibelia [13]. После этого, геномы рассматриваются как последовательности синтенных блоков вместо строк из нуклотидов. Хотя каждый референсный геном преобразуется в цельную последовательность блоков (с точностью до хромосом), собираемый геном соответствует множеству последовательностей (из-за того, что геном фрагментирован на контиги).

Из-за этой фрагментации, некоторая информация о смежностях (т.е. о соседствующих блоках и, соответственно, порядке) отсутствует. Ragout анализирует геномные перестройки для восстановления этой информации. Вначале мы фильтруем все повторные синтенные блоки, а также блоки, которые не представлены в собираемом геноме. Из оставшихся блоков мы конструируем *Неполный цветной граф брейкпоинтов (incomplete multi-color breakpoint graph)*, в котором вершины соответствуют концам синтенных блоков, а ребра соответствуют смежностям между ними. Затем алгоритм решает *Задачу парсимонии для состояния полу-брейкоинтов (Half-breakpoint State Parsimony Problem)* для преобразования данного графа в «обычный» граф брейкпоинтов [1] путем восстановления смежностей в собираемом геноме. Далее, контиги собираются в скэффолды с учетом полученных смежностей. Вся процедура, описанная выше повторяется несколько раз с различным размером синтенных блоков, а наборы скэффолдов полученные в соответствующих запусках объединяются в одну сборку. Затем,

выполняется этап уточнения сборки, в котором короткие и повторные контиги вставляются на соответствующие места в скэффолдах с помощью информации о смежностях, полученных из графа перекрытий. Псевдокод алгоритма представлен ниже.

Algorithm 1 Псевдовод алгоритма Ragout

```
procedure RAGOUT(references, target, phylogeny, blockSizes)
    assemblies ← ∅
    for all blockSize in blockSizes do
        synBlocks ← RUNSIBELIA(references, target, blockSize)
        bpGraph ← BUILDBREAKPOINTGRAPH(synBlocks)
        weightedGraph ← EDGESSCORE(bpGraph, phylogeny)
        adjacencies ← MINPERFMATCHING(weightedGraph)
        scaffolds ← BUILDSCAFFOLDS(target, adjacencies)
        ADD(scaffolds, assemblies)
    end for
    scaffolds ← MERGEITERATIONS(assemlies)
    assemblyGraph ← BUILDASSEMBLYGRAPH(target)
    scaffolds ← REFINESCAFFOLDS(scaffolds, assemblyGraph)
    OUTPUTSCAFFOLDS(scaffolds)
end procedure
```

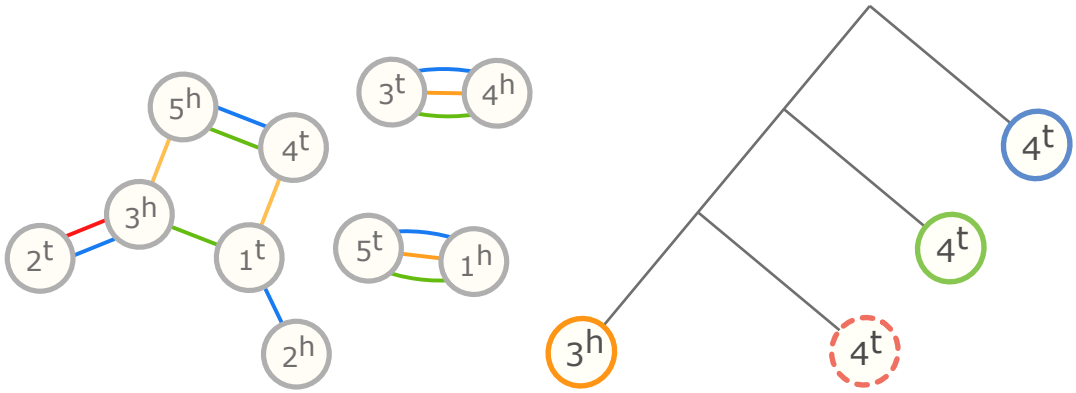
Так как алгоритм декомпозиции на синтенные блоки уже описан в [13], мы не рассматриваем его в данной работе. Ниже представлено подробное описание алгоритма Ragout с того момента, когда разложение входных геномов на синтенные блоки уже известно.

2.2 Алгоритм анализа перестроек для восстановления смежностей

Пусть дан набор референсных, а также собираемый геномы, представленные в алфавите синтенных блоков. Наша задача – восстановить смежности между блоками на границах контигов собираемого генома. Для анализа информации о смежностях хорошо подходит граф брейкпоинтов, однако они определены для цельных геномов. Ниже представлен *Неполный цветной граф брекпоинтов* который удобен для представления информации о смежностях во фрагментированных геномах.

Неполный цветной граф брейкпоинтов

Пусть даны сборка A и k референсных последовательностей – $P_1..P_k$ над алфавитом из синтенных блоков B; определим неполный цветной граф брекпоинтов $BG(A, P_1..P_k) = (V, E)$, где $V = \{b^h, b^t | b \in B\}$. Каждому синтенному блоку в графе соответствуют две вершины, соответствующие началу и концу блока. Ребра – неориентированные и раскрашены в $k + 1$ цветов. Ребро соединяет пару вершин, соответствующих началу/концу двух смежных блоков в одном из геномов и раскрашено соответсвующий цвет. Для упрощения обозначений, мы будем называть *красным* цвет, соответствующий собираемому геному. Пример описанного графа представлен на рисунке 2.1a.



(a) Граф брейкпоинтов (b) Филогенетическое дерево

Рисунок 2.1 — (a) Граф брейкпоинтов для трех референсных и одного собираемого геномов. Три референсных генома (*Ref1*, *Ref2* and *Ref3*) представлены как циклические перестановки синтенных блоков: *Ref1(синий):* $+1$ $+2$ $+3$ $+4$ $+5$, *Ref2(зеленый):* $+1$ $+3$ $+4$ $+5$, и *Ref3(оранжевый):* $+1$ -4 -3 $+5$, соответственно. Сборка (красный) представлена четырьмя разделенными перестановками (соответствующим четырем контигам): *Сборка:* $+1$ | $+2$ $+3$ | $+4$ | $+5$. (b) Филогенетическое дерево, соответствующее состоянием *полу-брейкпоинта* 5^h. В каждом листе написано состояние *полу-брейкпоинта* 5^h в соответствующем референсном/собираемом геноме. (Согласно дереву, состояние 5^h в собираемом геноме 4^t, однако правильное состояние 5^h в нем неизвестно.)

Перед построением графа, мы фильтруем синтенные блоки, которые не представлены в собираемом геноме, т.к. такие блоки не несут полезной информации. Также, мы фильтруем блоки, которые присутствуют в нескольких копиях хотя бы в одном из геномов, т.к. такие блоки значительно усложняют дальнейший анализ. После процедуры фильтрации, граф приобретает два важных свойства: (i) каждая вершина инцидентна не более чем одному ребру каждого из цветов (т.к. были отфильтрованы повторные блоки) и (ii) каждая вершина соответствует началу/концу некоторого синтенного блока в собираемом геноме. Таким образом,

если бы у нас была цельная последовательность собираемого генома, набор всех красных ребер образовывал бы совершенное паросочетание. Однако, поскольку геном фрагменирован на контиги, некоторая информация о смежностях для блоков на концах контигов недоступна. Основная задача состоит в нахождении этих «невидимых» красных ребер, используя информацию из референсных геномов, а также их эволюционную взаимосвязь.

Филогенетическое дерево

Пусть T – укорененное филогенетическое дерево для геномов $A, P_1..P_k$. T состоит из $k+1$ листа, $k-1$ внутренней вершины степени три и одной вершины степени два, называемой корень. Ребра, соединяющие две вершины, называются ветвями, длина которых выражает эволюционное расстояние (которое обычно пропорционально количеству нуклеотидных замен).

Парсимонная модель перестроек

Пусть дано дерево T, в листьях которого расположены $k+1$ *целых* геномов (представленных синтенными блоками). Процедура парсиомнии (Parsimony procedure) [7, 18] находит эволюционный сценарий преобразования геномов вдоль дерева, который соответствует минимального количеству изменений, что соответствует некоторой минимальной стоимости. Если один из геномов разделен на контиги, то возможным критерием для его объединения в цельный фрагмент может быть нахождение такой перестановки контигов, что процедура парсимонии вернет минимальную стоимость среди всех возможных перестановок. В этом случае, задача подразумевает две стадии оптимизации: в одной мы находим наиболее бережливый эволюционный сценарий (и соответствующую стоимость), во второй мы находим конфигурацию собираемого генома, соответствующую минимальной из всех стоимостей. Обычно, в качестве операций для первой стадии оптимизации выбирают 2-break операцию (также называемую DCJ операцией), т.к. она хорошо соответствует реальным геномным перестройкам [1]. Однако, оптимизация с такой операцией является NP-полной задачей [12]. Вместо этого, в данной работе мы рассматриваем процедуру парсимонии на каждом из брейкпоинтов отдельно. Хотя такая модель может не всегда отражать реальные перестройки (так как в каждой из них участвует как минимум два брейкпоинта), она более удобна для вычислений, а также уже хорошо показала себя в задачах по восстановлению предковых геномов [12].

Каждую вершину в графе брейкпоинтов $BG(A, P_1..P_k)$ мы называем *полу-брейкпоинтом (half-breakpoint)* (т.к. у «целого» брейкпоинта два конца). Для данного генома P_i, состояние полу-брейкпоинта v_i определено как вершина, инцидентная ему, т.е. вершина v_j такая, что ребро (v_i, v_j) имеет цвет P_i в графе брейкпоинтов. Из-за свойств графа, описанных выше, каждый полу-брейкпоинт имеет не более одного состояния. Если синтеннго блока, соответствующего полу-брейкпоинту v_i нету в геноме P_i, то состояние v_i обозначается как *void* (смотри рисунок 2.1b).

Вместо 2-break операций, в процедуре парсимонии мы рассматриваем изменения состояний полу-брейкпоинтов для каждой вершины брейкоинт графа в отдельности. Стоимость смены состояния вдоль ветви филогенетического дерева b, имеющей длину t равна $W(b) = e^{-t}$. Интуитивно, чем длиннее ветвь, тем больше вероятность смены состояния брейкпоинта и тем меньше будет соответствующая стоимость. Далее, мы описываем *Задачу парсимонии для состояний полу-брейкпоинтов*, которая находит эволюционный сценарий изменений состояния для конкретного полу-брейкпоинта.

Процедура парсимонии для состояний полу-брейкпоинтов
Дан полу-брейкпоинт u и дерево T, в каждом из узлов которого указано состояние u в соответствующем геноме. Требуется задать состояния u во внутренних узлах дерева таким образом, чтобы стоимость эволюционного сценария смены состояний данного полу-брейкпоинта была минимальной.

Ниже представлен алгоритм для решения данной задачи в помощью динамического программирования. Алгоритм сходен с алгоритмом Сэнкоффа для взвешенной малой парсимонии (Sankoff's algorithm for the weighted small parsimony problem), и доказательства оптимальности полностью идентичны [18].

- *Ввод:* $state(u)$ для каждого генома в дереве.
- *Вывод:* $state(u)$ для каждого внутреннего узла дерева и суммарная стоимость
- *Целевая функция:* $s_t(v)$ = минимальная стоимость поддерева с корнем в v, если v находится в состоянии t

- *Инициализация:* Для каждого узла дерева l: $s_t(l) = 0$, если состояние полу-брейкпоинта в $l = t$, иначе $s_t(l) = \infty$
- *Рекурсия:* Стоимость каждого узла вычисляется учитывая стоимость его детей.

$$s_t(\textit{parent}) = \textit{min}_i\{s_i(\textit{leftchild}) + \bar{\delta}_{i,t}W(\textit{leftbranch})\}+ \\ \textit{min}_j\{s_j(\textit{rightchild}) + \bar{\delta}_{j,t}W(\textit{rightbranch})\}$$

  где $\bar{\delta}_{i,j} = 0$ *если* $i = j$ *и 1 иначе.*

- *Завершение:* $min_a s_a(root)$
- *Сложность:* $O(nd)$, где n - число листьев, а d - число возможных состояний.

Когда состояния всех внутренних узлов стали известны, стоимость *Процедуры парсимонии для состояний полу-брейкпоинтов* для полу-брейкпоинта u можно посчитать как сумму по всем ветвям дерева, где его состояние меняется:

$$P(u, T) = \sum_{\textit{branch (i, j)}} \bar{\delta}_{i,j}W(\textit{branchlength})$$

Восстановление смежностей

Поскольку мы можем эффективно решать *процедуру парсимонии для состояний полу-брейкпоинтов*, остается ответить на вопрос, как восстановить смежности таким образом, чтобы $\sum_{u \in G} P(u, T)$ – общая сумма стоимостей смен состояний всех полу-брейкпоинтов была минимальной. Ранее было показано, что ребра, соответствующие этим состояниям, должны задавать совершенное паросочетание на графе брейкпоинтов. Стоимость такого паросочетания естественно задать аналогично – как сумму стоимостей смен состояния полу-брейкпоинтов.

Для каждой вершины без инцидентного ей красного ребра, алгоритм Ragout вычисляет стоимость *процедуры парсимонии для состояний полу-брейкпоинтов* для каждого из возможный состояний собираемого генома, где возможные состояния определяются инцидентыми вершинами в графе брейкпоинтов. Т.к. выбор состояния некоторого полу-брейкпоинта соответствует выбору инцедентно-

го ему ребра, для каждого ребра в графе брейкпоинтов вычисляются две стоимости. Мы определяем вес ребра как сумму этих двух значений. Так как некоторые смежности в собираемом геноме изначально известны (вершины имеют инцидентное красное ребро), мы удаляем соответсвующие вершины из графа до запуска алгоритма. Затем мы используем алгоритм Blossom для нахождения совершенного паросочетания минимального веса за время $O(|V|^4)$. Такое паросочетание определяет оптимальный порядок контигов.

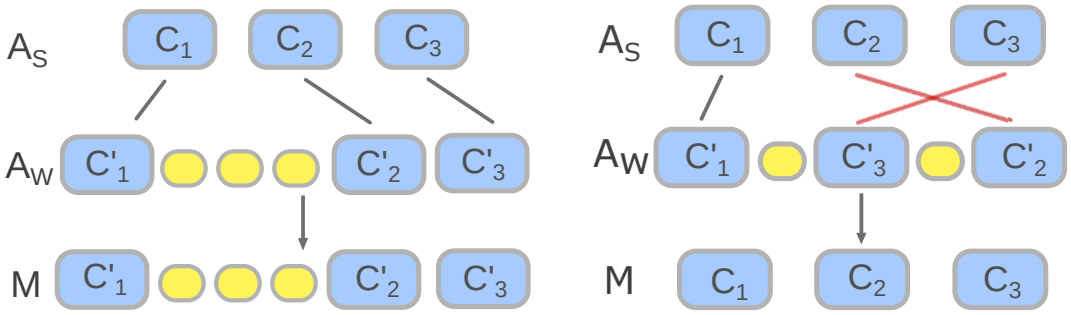
Построение скэффолдов

Наконец, мы объединяем контиги в скэффолды. Начиная с одного контига, мы продолжаем его влево и вправо в соответствии со смежностями, полученными из описанного выше паросочетания.

2.3 Выбор размера синтенного блока и итеративная сборка

Процедура декомпозиция геномов на синтенные блоки является зависимой от различных входных параметров, выбор которых обуславливается «степенью гранулярности» синтенных блоков, которая может варьироваться в каждом отдельном исследовании. Поэтому, возникает вопрос – какой масштаб декомпозиции нужно выбирать для конкретной задачи. Мы утверждаем, что синтенных блоков фиксированного размера не всегда достаточно для достижения наилучших результатов. Если выбрать большой размер блока, то маленькие и фрагментированные блоки (возникшие из-за разбиения генома на контиги и микроперестроек) не будут рассматриваться, что, таким образом, приведет к пропускам в сборке. С другой стороны, при выборе блоков малого размера анализ перестроек значительно усложняется, т.к. синтенные блоки малого размера более подвержены структурным вариациям, а также больше предрасположены к их неправильной идентификации различными алгоритмами. Для разрешения данной дилеммы мы используем несколько наборов синтенных блоков разного масштаба для построения нескольких сборок, а затем объединяем их.

Рассмотрим два скэффолда A_s и A_w, полученных из крупно и мелкомасштабных синтенных блоков соответственно. Контиг называется *сильным*, если он принадлежит скэффолду A_s или же *слабым*, если он присутствует в A_w, но не в A_s. Два скэффолда A_s и A_w называются *согласованными*, если каждая пара со-



(a) *Локально согласованные* (b) *Локально несогласованные*

Рисунок 2.2 — Объединение двух скэффолдов A_s и A_w, полученных из стадий сборки с разным размером блоков в скэффолд M. Желтые прямоугольники соответсвуют *слабым* контигам. (a) A_s и A_w согласованы. (b) A_s и A_w не согласованы.

седних контигов из A_s либо смежна, либо разделена только *слабыми* контигами в A_w. Хотя порядок контигов более надежен в A_s, чем в A_w, есть много контигов, которые отсутствуют в A_s, но есть в «мелкомасштабном» A_w. Таким образом, мы полагаемся на «скелет» из контигов из A_s, и добавляем в него контиги из A_w, если они *согласованы* (смотри рисунок 2.2).

Объединенные скэффолды, таким образом, будут *согласованными* с A_s. Мы последовательно применяем описанную процедуру к скэффолдам с разных стадий сборки (отсортированных к порядке уменьшения размера синтенного блока), пока не достигнем стадии с самым малым размером.

2.4 Уточнение с помощью графа сборки

Хотя процедура итеративной сборки стремится максимизировать число контигов, используемых для построения скэффолдов, некоторые типы контигов все равно остаются неиспользованными: (i) контиги короче чем минимальный размер синтенного блока, который могут определить современные алгоритмы, (ii) контиги, содержащиеся только в собираемом геноме и (iii) повторные контиги. Данные геномные фрагменты не рассматриваются в анализе перестроек, описанном выше и, таким образом, не войдут в финальную сборку. Для того, чтобы добавить их туда, мы используем граф перекрытий, который выводится ассемблером, либо может быть независимо получен из входных контигов. Геном проходит через этот граф некоторым неизвестным путем. Однако, так как теперь нам доступны скэффолды, мы можем использовать их для упорядочивания ма-

леньких и повторных контигов. Данная задача аналогична задаче разрешения повторов при сборке из коротких ридов. Вместо информации о парных ридах у нас есть упорядоченные пары смежных контигов, полученные из анализа графа брейкпоинтов.

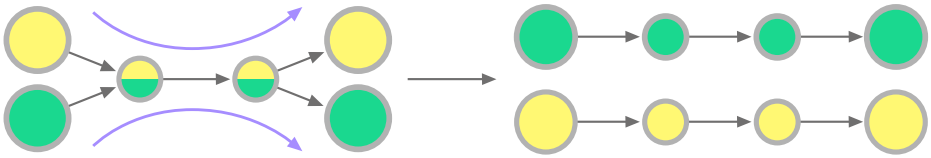


Рисунок 2.3 — Уточнение с помощью графа сборки. Процедура заполняет пропуски в скэффолдах маленькими контигами. Большие круги соответствуют контигам с известным порядком, а малые изображают контиги, которые не рассматривались в анализе перестроек.

Пусть дан граф сборки и набор скэффолдов из предыдущего этапа алгоритма. Для каждой пары смежных контигов в них, мы находим все возможные пути, соединяющие их в графе перекрытий, не содержащие других контигов из скэффолдов. Если такой путь один, то мы вставляем промежуточные контиги, по которым он проходит, между рассматриваемыми контигами из скэффолда (смотри рисунок 2.3). Данная процедура значительно увеличивает количество использованных контигов на большинстве наборов данных.

ГЛАВА 3

Результаты

Мы протестировали Ragout, а также другие алгоритмы для референсной сборки (Mauve Contig Mover [17], OSLay [16], RACA [9]) на одном симулированном и трех реальных наборах данных. Обладая цельными последовательностями собираемых геномов, мы также можем оценить качество получившихся сборок, сравнивая число *ошибочных контигов*, *пропусков в скэффолдах* и *покрытие*.

Поскольку каждый скэффолд является упорядоченным списком контигов, мы определяем число *ошибочных контигов* как минимальные число контигов в скэффолдах, для которых позиции, полученные выравниванием на референс не согласуются с позициями контигов впереди и позади. Также, мы определяем число *пропусков* в скэффолдах как число пар контигов, которые смежны в скэффолде, но разделены некоторыми другими контигами при выравнивании на референсный геном. *Покрытие* определено как общее число выровненных нуклеотидов, деленное на размер генома.

Mauve Contig Mover (далее MCM) и OSLay запускались с параметрами, рекомендованными для бактериальных геномов. Для Ragout мы выбрали три итерации с размерами блока $5000, 500, 100$ соответственно, так как такие размеры являются размерами по умолчанию для бактериальных геномов в Sibelia [13]. Так как RACA работает с фиксированным размером синтенного блока, мы выбрали максимальный из набора выше (наиболее надежный).

3.1 Сборка с использованием одного близкородственного референса

Сначала, мы тестировали указанные выше инструменты на легком случае, когда между собираемым и референсным геномами отсутствуют структурные вариации. В такой ситуации одного референса достаточно для получения правильной сборки. Это также дает нам возможность сравнить Ragout с MCM и OSLay, которые работают только с одной референсной последовательностью.

Набор данных состоит из двух штаммов *Escherichia coli*: *DH1 (NC_017625)* в качестве референса и *K-12 subs. MG1655 (NC_000913)* в качестве собираемого генома. Контиги были получены с помощью ассемблера SPAdes [2]. Результа-

ты представлены в таблице 3.1. Ragout и MCM успешно восстановили один цельный скэффолд, однако, результат OSLay – восемь. Сборка Ragout без уточнения сравнима по качеству со сборками MCM и OSLay, однако с уточнением наш алгоритм использует значительно больше контигов, а финальные скэффолды имеют меньше *пропусков*.

	Ragout	MCM	OSLay
Скэффолды	1 (1)	1	8
Покрытие	97.9 (97.6)	97.6	96.7
Контиги	129 (79)	77	80
Пропуски	52 (71)	73	61
Ошибки	0 (0)	0	1

Таблица 3.1 — Сравнение различных алгоритмов для одного референса. Все алгоритмы запускались с параметрами по умолчанию. Для Ragout, результаты даны как с уточнением, так и без него (в скобках). Общее число контигов – 156. Изначальное покрытие сборки – 98.18%.

3.2 Сборка одной хромосомы с помощью нескольких референсов

Далее мы рассматриваем более сложный случай – когда доступно неколько референсных геномов, однако каждый из них содержит структурные вариации по отношению к собираемому геному. Набор данных состоит из пяти различных штаммов *Helicobacter pylori*: *Puno120 (NC_017378), ELS37 (NC_017063), Gambia94/24 (NC_017371)* и *G27 (NC_011333)* к качестве референсных геномов и *SJM180 (NC_014560)* в качестве собираемого. Соответствующее филогенетическое дерево показано на рисунке A.1a. Точечную диаграмму (dot plot), демонстрирующую перестройки можно увидеть на рисунке A.3. Контиги были собраны с помощью ABySS [19].

Вначале мы запустили Ragout, а также OSLay и MCM на каждом из нескольких референсов в отдельности, чтобы показать, что «обычная» сборка с одним референсом не дает удовлетворительного результата в данном случае. Каждая программа выдает *ошибочные* контиги, что может быть объяснено структурными вариациями между референсными и собираемым геномами (смотри таблицу 3.2).

Так как OSLay и MCM работают только с одним референсом, мы использовали Ragout и RACA для сравнения разных поднаборов всех доступных референсов (смотри таблицу 3.3). Для RACA мы выбрали штамм *G27* в качестве референсного, а остальные были внешними. Оба алгоритма имеют *ошибочные* контиги при использовании трех и менее референсов. С помощью Ragout удалось получить безошибочную сборку на наборе из четырех референсов, однако RACA все еще имеет *ошибочные* контиги на таком наборе данных. Также, сборка Ragout имеет более полное *покрытие* и меньше *пропусков*.

Референс	Скэффолды	Контиги	Ошибки	Покрытие
Mauve Contig Mover				
G27	1	53	7	97.9
Gambia94/24	1	54	8	97.9
ELS37	1	45	9	98.0
Puno120	1	56	11	98.0
OSLay				
G27	5	50	1	96.2
Gambia94/24	8	49	3	96.4
ELS37	6	53	3	98.0
Puno120	8	51	2	87.0
Ragout				
G27	1 (1)	91 (50)	4 (4)	97.7 (97.4)
Gambia94/24	2 (2)	83 (45)	7 (7)	96.8 (96.6)
ELS37	1 (1)	102 (56)	4 (3)	98.1 (97.5)
Puno120	2 (2)	92 (49)	6 (6)	97.2 (96.9)

Таблица 3.2 — Сравнение с MCM и OSLay используя один референс *H. Pylori*, показывающее ошибочные контиги. Все алгоритмы запускались с параметрами по умолчанию. Для Ragout, результаты даны как с уточнением, так и без него (в скобках). Общее число контигов – 183. Начальное покрытие сборки – 98.57%.

3.3 Сборка нескольких хромосом с помощью нескольких референсов

Следующий набор данных был взят из недавнего исследования [4], в котором были использованы длинные риды, полученные с помощью технологии PacBio, для de novo сборки генома *Vibro cholerae H1 str. (AKGH01000001)* в цельную последовательность. Мы хотим показать, что с помощью нескольких

Референсы	Скэффолды	Покрытие	Контиги	Пропуски	Ошибки
Ragout					
G27, ELS37	2 (2)	97.8 (97.6)	95 (53)	22 (39)	1 (1)
G27, Puno120, ELS37	1 (1)	97.8 (97.6)	95 (53)	21 (36)	1 (1)
G27, Puno120, Gambia94/24, ELS37	1 (1)	97.6 (97.3)	93 (46)	22 (38)	0 (0)
RACA					
G27, ELS37	3	83.6	35	29	2
G27, Puno120, ELS37	2	83.6	35	30	1
G27, Puno120, Gambia94/24, ELS37	2	83.8	35	31	1

Таблица 3.3 — Сравнение Ragout и RACA с использованием нескольких референсов *H. Pylori*. Все алгоритмы запускались с параметрами по умолчанию. Для Ragout, результаты даны как с уточнением, так и без него (в скобках). Общее число контигов – 183. Начальное покрытие сборки – 98.57%.

референсных геномов (даже имеющих структурные вариациями относительно собираемого генома) высококачественные скэффолды для каждой из хромосом могут быть получены только лишь с использованием коротких ридов. Мы использовали SPAdes для сборки непарных ридов Illumina со средней длиной 40 пар оснований. Три референса были выбраны таким образом, чтобы каждый из них имел структурные перестройки относительно собираемого генома хотя бы в одной из хромосом: *O1 Biovar (AE003852), O1 Inaba (CM001785)* и *O395 (CP001235)*. Точечные диаграммы показаны на рисунке А.2. Филогенетическое дерево изображено на рисунке А.1b.

Используя три референса, Ragout успешно реконструировал два скэффолда, соответствующих двум хромосомам *V. Cholerae* (смотри таблицу 3.4). Уточнение с помощью графа сборки существенно увеличивает количество использованных контигов.

Для RACA мы выбрали штамм *O1 Inaba* в качестве референсного, остальные были использованы в качестве внешних. Результат RACA – три скэффолда без *ошибочных* контигов с тремя референсами и шесть скэффолдов с двумя референсами. Сборка Ragout превосходит сборку RACA по качеству как с уточнением так и без него.

Референс(ы)	Скэффолды	Покрытие	Контиги	Пропуски	Ошибки
Ragout					
O1 Inaba	3 (3)	95.3 (94.8)	317 (185)	41 (64)	5 (3)
O1 Inaba, O1 biovar	2 (2)	95.5 (94.7)	305 (179)	46 (68)	6 (4)
O1 Inaba, O1 biovar, O395	2 (2)	95.5 (94.7)	300 (174)	46 (66)	2 (0)
RACA					
O1 Inaba, O1 biovar	6	85.8	124	51	0
O1 Inaba, O1 biovar, O395	3	90.0	127	56	0

Таблица 3.4 — Сравнение Ragout и RACA с использованием нескольких референсов *V. Cholorea*. Все алгоритмы запускались с параметрами по умолчанию. Для Ragout, результаты даны как с уточнением, так и без него (в скобках). Общее число контигов – 1407. Начальное покрытие сборки – 96.89%.

3.4 Сборка с использованием далеких референсных геномов

Наконец, мы хотим рассмотреть случай, когда референсные геномы имеют большое количество структурных вариаций по отношению к собираемому. Для этого мы сгенерировали несколько наборов данных, так как обычно бактериальные геномы имеют небольшое число геномных перестроек.

Филогенетическое дерево, использованное для симуляций показано на рисунке A.1d. Для простоты мы выбрали симметричную структуру дерева, которая может рассматриваться как неукорененное (такой взгляд на дерево позволяет провести аналогию с задачей о медиане нескольких геномов), либо как укорененное со средней ветвью бесконечно малой длины. На каждой из четырех внешних ветвей мы сгенерировали пять инверсий и пять транслокаций. Вдобавок, для усложнения, мы сгенерировали 10 вставок/делеций на каждой ветви дерева (включая внутренние). Геномы были разделены на синтенные блоки, а затем разбиты на контиги таким образом, что каждый контиг соответствовал ровно одному блоку.

Наш анализ включает в себя случаи когда доступны четыре, три или два из симулированных геномов. Для каждого случая мы сгенерировали 100 разных наборов данных. Так как филогенетическое дерево симметрично, результат не зависит от того, какой конкретно из референсных геномов недоступен. Мы взяли *E. Coli K-12 str. MG1655 substr. (NC_000913)* в качестве собираемого генома, а количество синтенных блоков в декомпозиции в среднем равнялось 112.

Далее, мы запустили Ragout на каждом из наборов данных. Результаты представлены на рисунке 3.1, из которого ясно видно, что что число *ошибочных* конигов увеличивается, когда некоторые референсы недоступны. Ошибки в случае доступности всех референсов можно объяснить переиспользованием брейкпоинтов (breakpoint reuse), который не позволяет алгоритму различить перекрывающиеся перестройки.

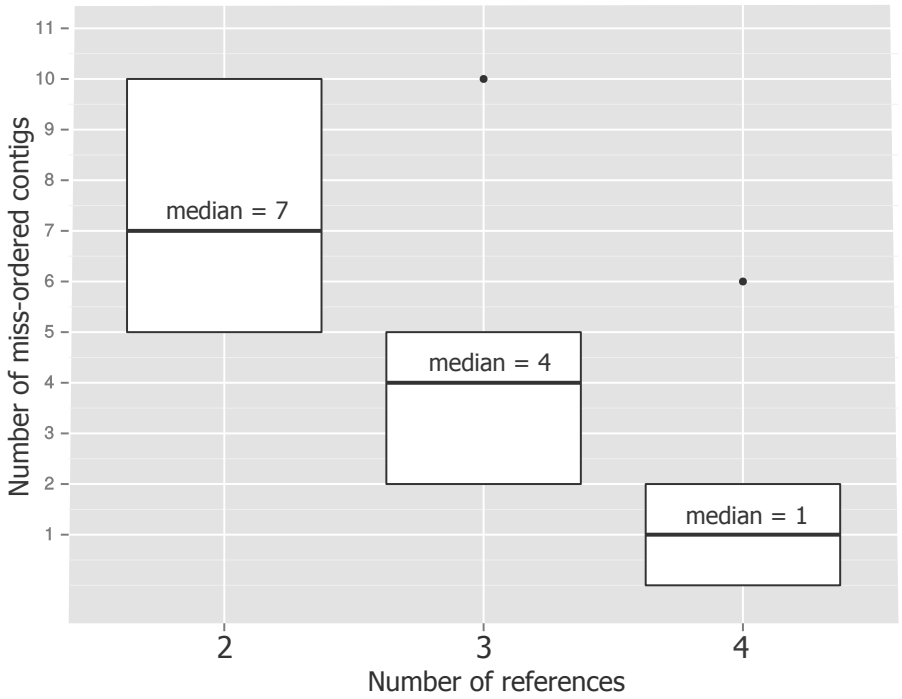


Рисунок 3.1 — Соответствие между количеством доступных референсов и количеством ошибок для Ragout.

Чтобы сравнить Ragout и RACA, мы выбрали два набора данных с четырьмя референсами, так как каждый запуск RACA связан с большим объемом технических приготовлений. Результаты представлены в таблице 3.5. В обоих случаях с помощью Ragout была получена более качественная сборка, чем с помощью RACA. Одним из возможных объяснений такого различия может быть факт того, что RACA сильно полагается на единственный референсный геном (больше, чем на внешние).

	Набор 1		Набор 2	
	Ragout	RACA	Ragout	RACA
Скэффолды	1	6	1	8
Покрытие	95.3	83.5	94.5	75.4
Сонтиги	112	101	112	90
Пропуски	3	6	11	5
Ошибки	0	3	4	2

Таблица 3.5 — Сравнение Ragout и RACA с использованием симулированных данных. (размер известен из симуляций) без уточнения. Общее количество контигов – 114 в обоих случаях. Начальное покрытие сборки – 100%.

3.5 Выбор параметров и итеративная сборка

Далее мы хотим протестировать Ragout с разными параметрами для итеративной/не итеративной сборки. Набор данных состоит из пяти штаммов *Staphylococcu aureus*: COL (NC_002951), JKD6008 (NC_017341), N315 (NC_002745), RF112 (NC_007622) в качестве референсов и и *USA300 (NC_007793)* в качестве собираемого генома. Филогенетическое дерево показано на рисунке A.1c. Контиги были собраны из данных одноклеточного секвенирования (single-cell) с помощью SPAdes.

Размер блока	100	500	5000	Итеративно
Скэффолды	5	2	1	1
Покрытие	96.7	96.3	92.0	96.7
Контиги	108	91	62	89
Пропуски	75	75	56	73
Ошибки	1	0	0	1

Таблица 3.6 — Итеративная сборка *S. Aureus*. Ragout была запущен с четырьмя референсами и различным размером синтенных блоков. Итеративная сборка была применена для комбинации всех предыдущий размеров (5000, 500, 100). Общее число контигов – 767. Начальное покрытие сборки – 98.4%. Мы не проводили уточнение с помощью графа сборки, чтобы сфокусироваться на результатах итеративного подхода.

Результаты тестирования представлены в таблице 3.6. Как и ожидалось, меньший размер синтенного блока позволяет алгоритму упорядочить больше контигов, но, вместе с тем, анализ становится сложнее. В результате алгоритм находит

неправильные смежности, что приводит к увеличению числа скэффолдов и *ошибочным* контигам. Далее мы применили итеративную сборку в три стадии: (5000, 500, 100). Сборка сохраняет цельный скэффолд из первой стадии и добавляет в него некоторые маленькие контиги из следующих стадий. Даже если некоторые *ошибочные* контиги могут быть перенесены в объединенный скэффолд, эти ошибки локальны и не будут нарушать изначальный «скелет» из контигов.

ЗАКЛЮЧЕНИЕ

В данной работе мы представляем алгоритм Ragout – программный пакет для улучшения сборок с помощью нескольких референсных геномов. Мы показываем, что при наличие нескольких референсов можно получить цельный высококачественный скэффолд для каждой из хромосом используя только данные секвенирования с короткими ридами. Этот результат является важным шагом вперед в задаче сборки геномов и даже ставит вопрос о том, действительно ли необходимы длинные риды технологии PacBio или же прыгающие библиотеки ридов с большой длиной вставки для сборки бактериальных геномов, когда в наличии имеется несколько законченных близкородственных геномов.

Текущая версия Ragout использует Sibelia для декомпозиции геномов на синтенные блоки и, тем самым, ограничивает свои возможности до геномов бактериальных размеров. Когда синтенные блоки получены, алгоритм Ragout достаточно быстр и эффективен по памяти. Мы планируем сделать Ragout совместимым с другими инструментами для декомпозиции на синтенные блоки, которые способны работать с геномами размера млекопитающих (к примеру, Cactus [14] и, таким образом, расширить Ragout на геномы такого размера. Еще одно ограничение Ragout состоит в том, что алгоритм использует граф сборки только лишь для восстановления коротких и повторных контигов. Таким образом, алгоритм может допускать ошибки, если геномные перестройки произошли на ветви филогенетического дерева, соответствующей собираемому геному. Так как граф сборки может быть трансформирован в граф брейкпоинтов и обратно, граф сборки полученный из ассемблера также может быть использован для анализа перестроек и мы в дальнейшем планируем сосредоточится на данной задаче.

СПИСОК ЛИТЕРАТУРЫ

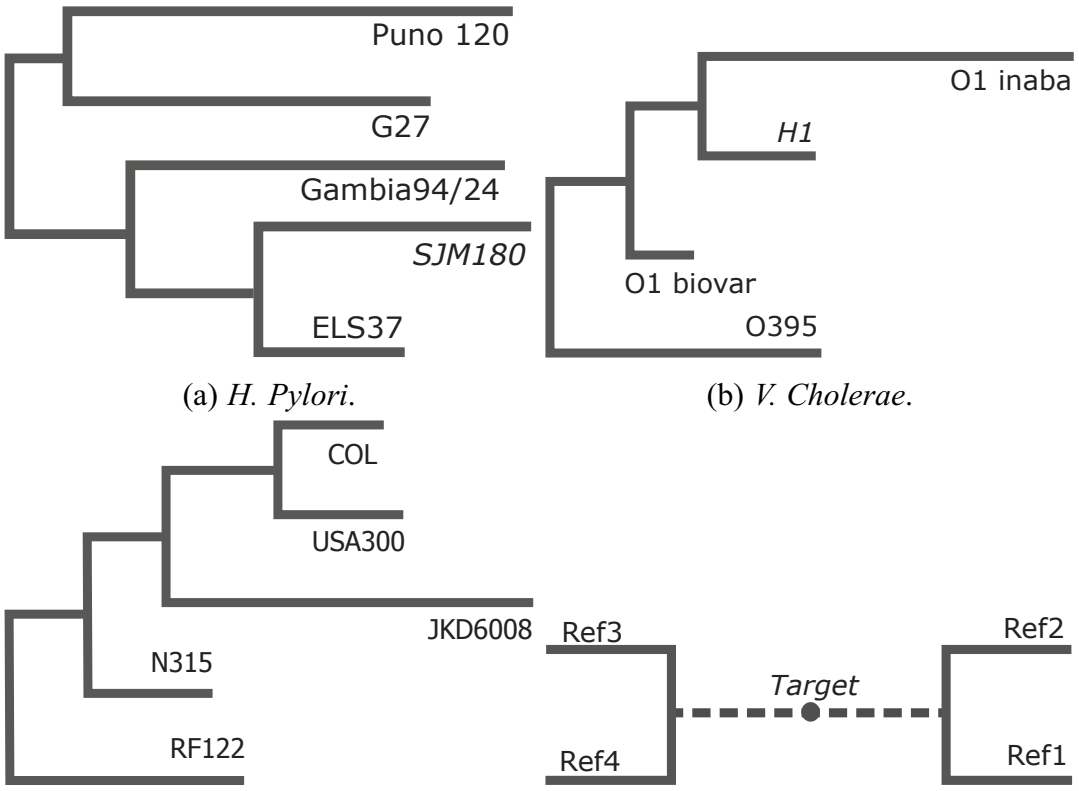
1. Max A Alekseyev and Pavel A Pevzner. Breakpoint graphs and ancestral genome reconstructions. *Genome research*, 19(5):943–957, 2009.
2. Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, Alexey V. Pyshkin, Alexander V. Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev, and Pavel A. Pevzner. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012.
3. Ali Bashir, Aaron A Klammer, William P Robins, Chen-Shan Chin, Dale Webster, Ellen Paxinos, David Hsu, Meredith Ashby, Susana Wang, Paul Peluso, et al. A hybrid approach for the automated finishing of bacterial genomes. *Nature biotechnology*, 2012.
4. Klammer Bashir, Chin Robins, Paxinos Webster, Ashby Hsu, Peluso Wang, Sorenson Sebra, Yen Bullard, Mollova Valdovino, Lin Luong, Joshi LaMay, Frace Rowe, Turnsek Tarr, Kasarskis Davis, Waldor Mekalanos, and Schadt. A hybrid approach for the automated finishing of bacterial genomes. *Nature Biotechnology*, 30(7):701–709, 2012.
5. Anne Bergeron, Julia Mixtacki, and Jens Stoye. A unifying view of genome rearrangements. In *Algorithms in Bioinformatics*, pages 163–173. Springer, 2006.
6. Viraj Deshpande, Eric DK Fung, Son Pham, and Vineet Bafna. Cerulean: A hybrid assembly using high throughput short and long reads. In *Algorithms in Bioinformatics*, pages 349–363. Springer, 2013.
7. Walter M Fitch. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416, 1971.
8. Éric Gaul and Mathieu Blanchette. Ordering partially assembled genomes using gene arrangements. In *Comparative Genomics*, pages 113–128. Springer, 2006.
9. Jaebum Kim, Denis M Larkin, Qingle Cai, Yongfen Zhang, Ri-Li Ge, Loretta Auvil, Boris Capitanu, Guojie Zhang, Harris A Lewin, Jian Ma, et al. Reference-assisted chromosome assembly. *Proceedings of the National Academy of Sciences*, 110(5):1785–1790, 2013.
10. Mikhail Kolmogorov, Brian Raney, Benedict Paten, and Son Pham. Ragout: a

reference-assisted assembly tool for bacterial genomes. *(to appear in ISMB 2014 proceedings)*.

11. Sergey Koren, Michael C Schatz, Brian P Walenz, Jeffrey Martin, Jason T Howard, Ganeshkumar Ganapathy, Zhong Wang, David A Rasko, W Richard McCombie, Erich D Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature biotechnology*, 30(7):693–700, 2012.
12. Jian Ma, Louxin Zhang, Bernard B Suh, Brian J Raney, Richard C Burhans, W James Kent, Mathieu Blanchette, David Haussler, and Webb Miller. Reconstructing contiguous regions of an ancestral genome. *Genome Research*, 16(12):1557–1565, 2006.
13. Ilya Minkin, Anand Patel, Mikhail Kolmogorov, Nikolay Vyahhi, and Son Pham. Sibelia: A scalable and comprehensive synteny block generation tool for closely related microbial genomes. In *Algorithms in Bioinformatics*, pages 215–229. Springer, 2013.
14. Benedict Paten, Dent Earl, Ngan Nguyen, Mark Diekhans, Daniel Zerbino, and David Haussler. Cactus: Algorithms for genome multiple sequence alignment. *Genome research*, 21(9):1512–1528, 2011.
15. Son K Pham and Pavel A Pevzner. Drimm-synteny: decomposing genomes into evolutionary conserved segments. *Bioinformatics*, 26(20):2509–2516, 2010.
16. Daniel C Richter, Stephan C Schuster, and Daniel H Huson. Oslay: optimal syntenic layout of unfinished assemblies. *Bioinformatics*, 23(13):1573–1579, 2007.
17. Anna I Rissman, Bob Mau, Bryan S Biehl, Aaron E Darling, Jeremy D Glasner, and Nicole T Perna. Reordering contigs of draft genomes using the mauve aligner. *Bioinformatics*, 25(16):2071–2073, 2009.
18. David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975.
19. Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J.M. Jones, and İnanç Birol. Abyss: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

ПРИЛОЖЕНИЕ А

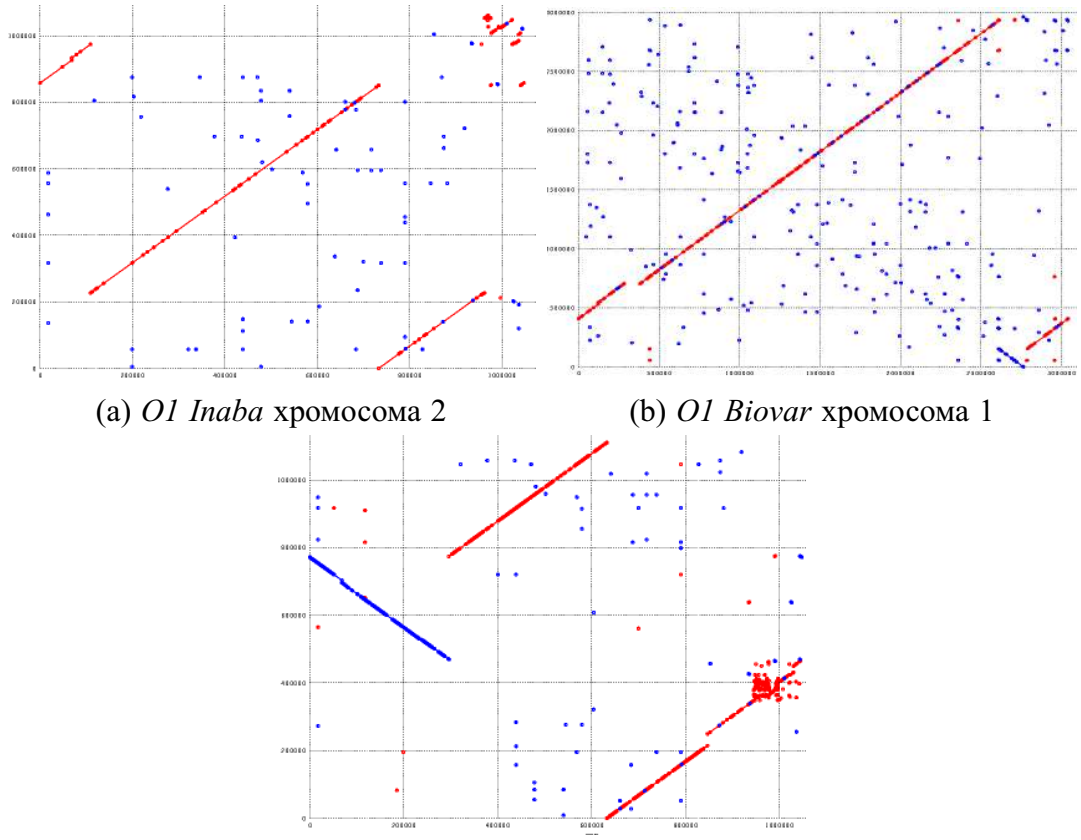
ИЛЛЮСТРАЦИИ



(a) *H. Pylori*. (b) *V. Cholerae*.

(c) *S. Aureus*. (d) Симулированные геномы.

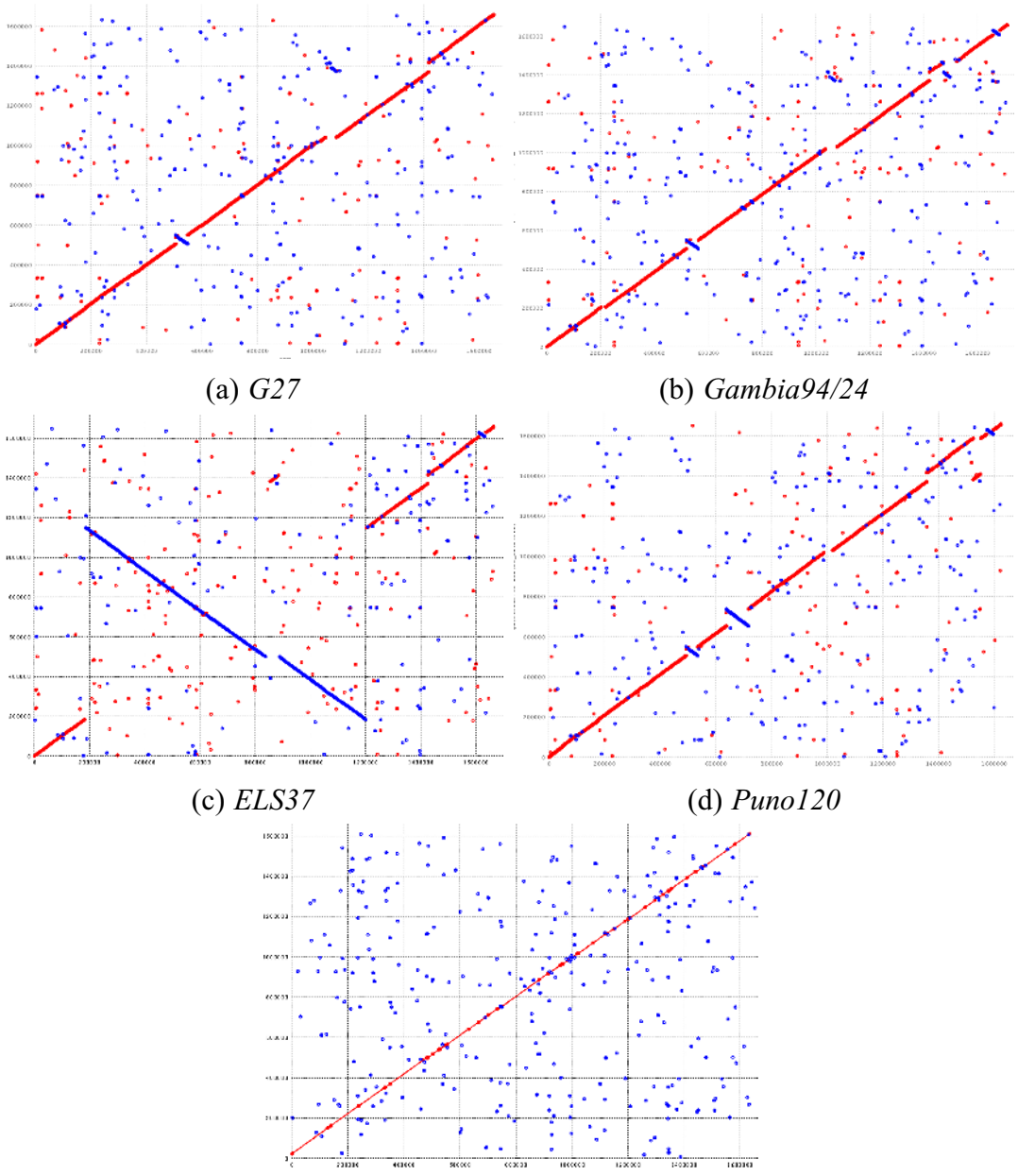
Рисунок А.1 — Phylogenetic trees. (a) *H. Pylori* с *SJM180* в качестве собираемого. (b) *V. Cholerae* с *H1* в качестве собираемого. (c) *S. Aureus* с *USA300* в качестве собираемого. (d) Симулированные геномы. Сплошные ветви содержат все типы геномных перестроек, а пунктирные – только вставки/делеции.



(a) *O1 Inaba* хромосома 2 (b) *O1 Biovar* хромосома 1

(c) *O395* хромосома 2

Рисунок А.2 — Точечные диаграммы разных хромосом референсов *V. Cholerae* (a-c) против соответсвующих хромосом целевого генома для демонстрации перестроек.



(a) *G27* (b) *Gambia94/24*

(c) *ELS37* (d) *Puno120*

(e) Результирующий скэффолд

Рисунок А.3 — (a-d) Точечные диаграммы референсов *H. Pylori* против собираемого генома. (e) Точечная диаграмма скэффолда Ragout против референсного генома для визуальной верификации.

ОГЛАВЛЕНИЕ