Федеральное агентство по образованию
Государственное образовательное учреждение высшего
профессионального образования
Пущинский государственный университет
Учебный центр математической биологии

На правах рукописи

Фурлетова Евгения Игоревна

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Построение классификационных Затравок для сравнения Аминокислотных последовательностей

Направление подготовки магистра
(010 500 Прикладная математика и информатика)

Магистерская образовательная программа
(Математическое моделирование)

«Работа допущена к защите»:

Научный руководитель
к.ф.-м.н. _____ М.А. Ройтберг

Руководитель магистерской
образовательной программы,
д.ф.-м.н., профессор _____А.М. Молчанов

Пущино - 2008

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ 2

ВВЕДЕНИЕ 4

Актуальность проблемы 4

Цели и задачи исследования 8

ГЛАВА 1. ОБЗОР ЛИТЕРАТУРЫ 10

Задача выравнивания биологических последовательностей 10

Матрицы замен. Вес выравнивания 11

Основные подходы к построению выравниваний 13

Методы динамического программирования 13

Эвристические методы 15

Чувствительность и избирательность эвристических методов 17

Новые виды затравок 18

ГЛАВА 2. ПОСТРОЕНИЕ КЛАССИФИКАЦИОННЫХ ЗАТРАВОК 22

Основные определения 22

Чувствительность и избирательность 23

Базовое распределение вероятностей 24

Целевое распределение вероятностей 25

Построение классификационных алфавитов 26

Пороговый алфавит 27

Транзитивный алфавит. Общие сведения 29

Иерархический транзитивный алфавит 32

Неиерархический транзитивный алфавит 34

ГЛАВА 3. ЭКСПЕРИМЕНТЫ И РЕЗУЛЬТАТЫ 37

Распределения вероятностей 37

Построение классификационных алфавитов 37

Классификационные затравки_____37
Множественные классификационные затравки_____39
Результаты сравнения затравок на теоретических моделях выравниваний_____39
Результаты сравнения затравок на реальных данных_____41
ЗАКЛЮЧЕНИЕ_____43
СПИСОК ЛИТЕРАТУРЫ_____45
ПРИЛОЖЕНИЕ_____50

ВВЕДЕНИЕ

Магистерская диссертация выполнена в рамках проекта **РФФИ 06-04-49249-а** на базе Института математических проблем биологии РАН, в лаборатории прикладной математики, под руководством к.ф.-м.н. Ройтберга Михаила Абрамовича, руководителя лаборатории прикладной математики ИМПБ РАН.

В ходе работы использовалось программное обеспечение, разработанное в рамках сотрудничества лаборатории прикладной математики и группой SEQUOIA университета г.Лилль (Франция). Это программное обеспечение было доработано с учетом нужд магистерской работы. Консультантом в области технологий программирования был м.н.с. лаборатории прикладной математики ИМПБ РАН Яковлев Виктор Вадимович.

Актуальность проблемы

Выравнивание (сравнение) первичных структур нуклеиновых кислот и белков - один из основных способов получения новых сведений об этих важных объектах. Его значение особенно возрастает в настоящее время в связи с осуществлением проектов тотального секвенирования геномов и совершенствованием методов определения первичных структур белков. В наше время базы данных последовательностей ДНК и белков стремительно пополняются. Поэтому растущий объем данных предъявляет новые, более жесткие, требования к эффективности разрабатываемых алгоритмов. Основным инструментом для анализа первичных структур белков и нуклеиновых кислот является выравнивание.

Задача выравнивания двух последовательностей - одна из наиболее старых проблем вычислительной биологии. Выравнивание новой последовательности с последовательностью уже хорошо изученного белка, т.е. такого, о котором известна третичная структура и функция, дает возможность количественно определить уровень сходства этих последовательностей, а также указать участки наиболее вероятного сходства структур или функций. Для того чтобы предсказание было правдивым, необходимо уметь строить биологически адекватное выравнивание последовательностей, т.е. выравнивание, отражающее эволюционное превращение одного белка в другой. Выравнивания используются во многих методах численного анализа биологических последовательностей, таких как функциональное аннотирование генов и белков, анализ доменов белков, моделирование трехмерной структуры белка на основе гомологического анализа.

Основными подходами к решению задач выравнивания биологических последовательностей являются алгоритмы, основанные на идее динамического программирования и более быстрые эвристические алгоритмы. Одни из первых алгоритмов выравнивания биологических последовательностей - алгоритм Нидлемана – Вунша [1] для глобальных выравниваний и алгоритм Смита – Уотермана [2] для локальных выравниваний. Данные алгоритмы основаны на идее динамического программирования, они появились в 70х годах. Однако с начала 80х годов наблюдается интенсивный рост баз данных последовательностей нуклеотидов и белков. Поэтому требовались более быстрые алгоритмы. И с середины 90х годов появились более быстрые эвристические алгоритмы, подобные FASTA [4] и BLAST [5].

Эвристические методы основаны на следующей идее. Сначала находят короткие гомологичные фрагменты последовательностей (*якоря или*

затравочные сходства). Затем, если это возможно, расширяют их до выравнивания методами динамического программирования. Образец для поиска затравочных сходств называется *затравкой*. Для поиска затравочных совпадений используется хэш-таблица, построенная заранее на основе списка последовательностей, с которыми ищут выравнивания.

Приведем пример затравки. Пусть дано выравнивание пары последовательностей. Сопоставим совпадающим парам позиций в фрагментах 1, а не совпадающим – 0. Тогда k совпадениям подряд соответствует слово из k единиц 11..1. Данное слово называется точной затравкой (seed). Обобщением точных затравок являются разреженные(spaced seed). Разреженные затравки представляют собой слово из 0 и 1. Затравка может быть представлена словом (группой слов) в некотором алфавите.

Эвристические методы работают гораздо быстрее, чем методы динамического программирования. Однако, точность выравнивания ниже. Эффективность методов выравнивания оценивается двумя параметрами: чувствительность и избирательность. Чувствительностью называется доля найденных целевых сходств. Например, пусть заранее известны сходные фрагменты последовательностей. Тогда доля найденных сходных фрагментов с помощью данного метода будет чувствительностью этого метода. Избирательностью называется вероятность не обнаружить два случайных фрагмента сходными. Чувствительность показывает точность метода, а избирательность – быстроту. Чем выше чувствительность и избирательность, тем эффективнее метод.

В последние 5 лет алгоритмы поиска в нуклеотидных базах были существенно улучшены за счет использования новых классов затравок: разреженных [10-12], векторных [14,15], множественных [13, 15-18], классификационных [20-22]. В то же время, разработке затравок для выравнивания аминокислотных последовательностей уделялось значительно

меньше внимания. Это связано с двумя причинами. Во-первых, при сравнении аминокислот нельзя ограничиваться критерием совпадения, необходимо учитывать разную степень сходства различных аминокислот. Во-вторых, затравки для аминокислотных последовательностей существенно короче, чем затравки для нуклеотидных последовательностей (2-3 буквы и 10-15 букв), что делает неэффективным прямое использование разреженных затравок. Основным инструментом для выравнивания аминокислотных последовательностей является метод BLASTp. Единственным продвижением в поиске более эффективного метода, чем BLASTp, является работа [15], в которой был предложен набор множественных векторных затравок для поиска в аминокислотных базах данных. Эти затравки обеспечивают лучшую (по сравнению с известной программой BLASTp) избирательность, однако требуют существенно большего количества обращений к хэш-таблицам поиска. Вопрос о возможности эффективного использования при поиске в белковых базах данных не-векторных затравок остается открытым. Данная работа посвящена исследованию возможности применения классификационных затравок к аминокислотным последовательностям. Классификационные затравки показали свое преимущество при выравнивании нуклеотидных последовательностей. В то же время, благодаря универсальности, они могут быть применимы и к аминокислотным последовательностям.

Цели и задачи исследования

Даная работа посвящена применению классификационных затравок к выравниванию аминокислотных последовательностей. Как было указано выше, затравочным алгоритмам выравнивания аминокислотных последовательностей уделялось гораздо меньше внимания, чем алгоритмам выравнивания нуклеотидных последовательностей. Единственным подходом к такому выравниванию является применение векторных затравок. А вопрос о возможности эффективного использования при поиске в белковых базах данных не-векторных затравок остается открытым. Использование векторных затравок имеет свою алгоритмическую сложность. А именно, требуют гораздо большего количества обращений к хэш-таблицам поиска. Поясним, как строится хэш-таблица на основе базы данных. Для каждой аминокислотной последовательности длины n (ключа) в хэш-таблице хранится множество позиций (номер последовательности и номер позиции в ней) ее вхождения. При поиске в стиле BLAST, когда смотрится совместный вклад нескольких позиций (см. раздел 1.5), помимо списка вхождений ключа необходимо просматривать списки вхождений гомологичных ему ключей. То есть последовательностей, составляющих с ним затравочное сходство. Например, при поиске с помощью программы BLAST для каждого ключа просматривается в среднем 19.34 смежных списков. Для решения этой проблемы будут предложены транзитивные классификационные затравки.

Настоящая работа мотивирована следующими вопросами: что нового нам даст применение классификационных затравок к выравниванию последовательностей белков? Можем ли мы достичь качества алгоритма BLAST или даже превзойти его? Как преодолеть трудность при поиске в хэш-таблице, которая возникает, когда используются векторные затравки?

При построении классификационных затравок мы столкнулись со следующими проблемами: 1) выбор классификационного алфавита; 2) построение классификационных затравок над выбранным алфавитом. Главная цель - выбрать алфавит таким образом, чтобы затравки над этим алфавитом имели высокую чувствительность при данной избирательности.

Исходя из вышесказанного, при исследовании необходимо решить следующие задачи:

(1) Разработка алгоритмов построения эффективных классификационных алфавитов.

(2) Построение классификационных затравок над выбранными алфавитами.

(3) Сравнение построенных классификационных затравок с векторными.

ГЛАВА 1.

ОБЗОР ЛИТЕРАТУРЫ

- формальное определение выравнивания биологических последовательностей;
- разные типы выравниваний, существующие методы построения выравниваний;
- Эвристические алгоритмы. Чувствительность и избирательность эвристических алгоритмов.
- Новые виды затравок.

1.1. Задача выравнивания биологических последовательностей

Сначала напомним, что первичная структура биополимера определяется последовательностью мономеров. В случае ДНК первичная структура может быть представлена как последовательность над алфавитом нуклеотидов $\sum = \{A,T,G,C\}$, в случае белка - как последовательность над алфавитом аминокислот $\Sigma = \{A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y\}$.

Выравнивание на уровне первичных структур – это сопоставление символов одной последовательности с символами другой с возможным пропуском или вставкой некоторого числа несопоставимых ни с чем символов, как в первой, так и во второй последовательностях. Выровнять две последовательности - значит расположить их одну под другой, при этом между некоторыми символами могут быть вставлены пробелы, так чтобы получившиеся подпоследовательности имели равную длину [3]. Символы,

расположенные в полученных последовательностях на одинаковых позициях, называются сопоставленными друг другу. Сопоставленные символы могут быть одинаковыми (образуют «совпадение») или различными (образуют «несовпадение»).

Приведем пример выравнивания аминокислотных последовательностей s = MQFLAVSTKKCA и t = MRAVSNKKCALK.

s MQFLAVSTKKCA - -

t MR - -AVSNKKCALK

Совпадения в позициях выравнивания подтверждают эволюционное родство последовательностей. Несовпадения говорят о том, что в данных позициях в одном из белков произошли мутации (мутация - замена одной аминокислоты на другую).

1.2. Матрицы замен. Вес выравнивания

Заметим, что существует множество способов выровнять две последовательности. Поэтому целью является выровнять последовательности таким образом, чтобы максимизировать их схожесть (гомологию). Каждому выравниванию ставится в соответствие вес W, который является мерой схожести двух последовательностей. В простейшем случае, вес выравнивания последовательностей нуклеотидов определяется как число совпадений. Обычно вес определяется с помощью матрицы замен (весов) следующим образом.

Пусть даны две последовательности s, t и их выравнивание (S,T). Вес $\boldsymbol{W}$ выравнивания двух последовательностей определяется как суммарный вес

сопоставлений (вычисляется по матрице замен *M*) минус суммарный штраф за делеции/вставки символов:

$$W(S,\sigma) = \sum_{ij} M(S[i], T[j]) - k_d \sigma$$

.

Где $M(S[i],T[j])$ – веса сопоставлений *i*-го символа первой последовательности и *j*-го символа второй последовательности, сумма производится по всем сопоставлениям выравнивания, k_d – количество вставок/делеций (пропусков), σ - штраф за вставку/делецию одного символа [3]. Таким образом, целью является нахождение выравнивания, имеющего больший вес. Такое выравнивание называется *оптимальным*. Обычно, когда говорят о задаче выравнивания последовательностей, слово оптимальное опускают.

Пусть дан алфавит ∑. Тогда элемент (i, j) матрицы замен для данного алфавита равен весу сопоставления *i*-го и *j*-го символа из ∑. Для аминокислот матрицы замен, как правило, вычисляются по частотам замен символов в среднем по банку структурных выравниваний белковых семейств, то есть в основе этих матриц косвенно лежат физико-химические свойства аминокислот. Очевидно, что такая матрица зависит от анализируемого банка и от метода оценки частот замен. На сегодняшний день существует несколько семейств матриц замен аминокислот, наиболее известные из них: PAM [24], BLOSUM [25], GONNET [26]. Матрицы в этих семействах получены, исходя из частот замен аминокислот в гомологичных фрагментах эволюционно близких белков; различные матрицы в серии различаются степенью близости использованных фрагментов.

Выше мы рассматривали σ как константу. Однако, с точки зрения эволюционных событий, наличие пропусков длиной в несколько символов соответствует единичному событию, причем длинные пропуски для родственных последовательностей менее вероятны, чем короткие, поэтому

логично ввести штраф за множественные пропуски, как функцию от ее длины:

$$\sigma(l) = A + B \cdot l \,.$$

Где ***A*** – штраф за открытие делеции, а ***B*** – штраф за продолжение множественной делеции. Такая схема называется *аффинным* штрафом за вставку/делецию(пропуск). В некоторых специфических задачах используются другие схемы штрафования множественных пропусков, например [37]:

$$\sigma(l) = A + B\log(l); \quad \sigma(l) = A + Bl^{C} \,.$$

1.3. **Основные подходы к построению выравниваний**

Задачи выравнивания последовательностей делятся на два основных типа: нахождение глобального и локального выравнивания. Глобальным выравниванием называется выравнивание двух биополимеров. В то время как локальное сравнение – поиск пар фрагментов, удовлетворяющих определенным условиям сходства (*целевые* сходства).

Существует два основных подхода к решению задач выравнивания биологических последовательностей: подход, основанный на идее динамического программирования и более быстрые эвристические подходы. Одними из первых алгоритмов выравнивания биологических последовательностей являются алгоритм Нидельмана – Вунша [1] для глобальных выравниваний и алгоритм Смита – Уотермана [2] для локальных выравниваний.

1.4. **Методы динамического программирования**

Данная работа посвящена исследованию эвристических алгоритмов. Поэтому здесь приведем только общую идею подхода динамического программирования.

Рассмотрим две последовательности: $t = t_1 t_2 \ldots t_n$ и $s = s_1 s_2 \ldots s_m$ над конечным алфавитом $\sum$. Пусть W – вес выравнивания. Вес $M(a,b)$ между двумя символами a и b вычисляется с помощью матрицы замен. Целью является нахождение выравнивания, имеющего максимальный вес. Данная проблема сводится к следующей задаче динамического программирования [35].

Рассмотрим граф, вершины которого соответствуют парам (i, j), где $0 \le i \le n$ и $0 \le j \le m$, а ребра - переходам из одних вершин в другие. Возможны три перехода из вершины (i, j): в вершины $(i+1, j)$, $(i, j+1)$ и $(i+1, j+1)$. Первый переход соответствует делеции символа s_{i+1} в последовательности s, второй – делеции символа t_{j+1} в последовательности t. Переход $(i, j) \to (i+1, j+1)$ соответствует сопоставлению символов s_{i+1} и t_{j+1} при выравнивании. Начальным является состояние (0,0), конечным – состояние (n,m). Тогда выравнивание последовательностей соответствует пути из начального состояния в конечное. Обозначим через q_i вершину под номером i в данном графе, а через (q_i,q_j) – ребро, соединяющее вершины q_i и q_j. Припишем каждому ребру вес $W(q_i,q_j)$. Тогда вес пути $q_0 \to \ldots \to q_i \to \ldots \to q_n$, ведущего из начальной вершины q_0 в конечную вершину q_n, будет равен сумме весов ребер, через которые проходит путь, т.е.

$$W(q_0, q_n) = \sum_{i=0,n} W(q_i, q_{i+1}) .$$

Тогда нахождения оптимального выравнивания сводится к нахождению пути из начальной вершины в конечную, имеющего максимальный вес. На рисунке 1.4.1 приведен пример вышеописанного графа и выравнивания последовательностей.

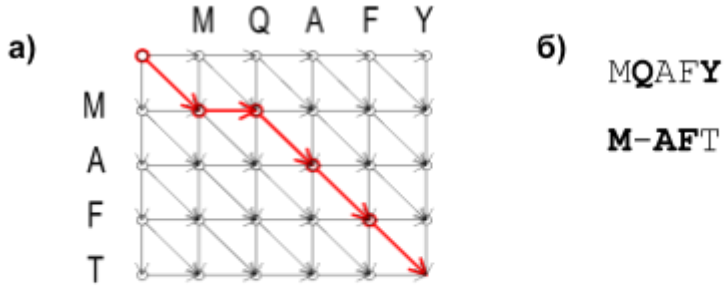


Рисунок 1.4.1. (а) Граф Нидельмана-Вунша [1] для двух коротких последовательностей MQAFY и MAFT. **(б)** Выравнивание этих последовательностей, соответствующее выделенному на графе пути.

Алгоритм динамического программирования имеет сложность $O(mn)$ по времени. Данные алгоритмы очень медленны и трудно применимы к большим последовательностям. Уже в конце 80-х годов банки разрослись до таких размеров, что поиск последовательностей был сопряжен с большими затратами времени и усилий специалистов. Поэтому наибольшую популярность имеют эвристические методы. Хотя эти методы не находят оптимальное выравнивание, а лишь некоторое приближение к нему.

1.5. **Эвристические методы выравнивания**

Наиболее популярными эвристическими алгоритмами выравнивания последовательностей являются FASTA [4] и BLAST [5]. Алгоритм FASTA – один из самых ранних эвристических алгоритмов.

В основе данных алгоритмов лежат следующие действия:

1. На первом этапе ищут *затравочные сходства (якоря)* – пары фрагментов, обладающие сходством определенного вида, которые легко искать. *Затравкой* (seed) называется шаблон, по которому ищутся затравочные сходства. Программы BLAST и FASTA ищут затравочные сходства, представляющие собой пары совпадающих фрагментов последовательностей длины *k*. Сопоставим совпадающим парам позиций в фрагментах 1, а не совпадающим – 0. Тогда таким фрагментам соответствует слово из *k* единиц 11..1. Данное слово будет затравкой для поиска соответствующих затравочных сходств. Такие затравки называются *точными*. Точные затравки используются в алгоритмах FASTA(*k* от 4 до 6) и BLAST(*k* от 9 до 11). Точные затравки применяются для выравнивания последовательностей нуклеотидов, однако они не учитывают разницы между несовпадающими парами. И поэтому не применяются для белковых последовательностей.

2. На втором этапе каждое затравочное сходство расширяют до локального совпадения нужного веса, если это возможно. Расширение проводят с помощью алгоритмов динамического программирования. С помощью специального алгоритма отсеиваются сегменты малого веса так, что остаются только те сегменты, проекции которых не пересекаются (см. Рисунок 1.5.1).

3. Для построения результирующего выравнивания из оставшихся сегментов применяется алгоритм динамического программирования. Результат сборки выравнивания из сегментов показан на Рисунке 1.5.1.

4. Для оценки значимости получившихся выравниваний вычисляется *Evalue(W)* – математическое ожидание числа последовательностей из банка, которые имеют вес выравнивания с запросом больше либо равный данному весу *W*. Тем самым учитывается размер пространства поиска, т.е. размер банка последовательностей.

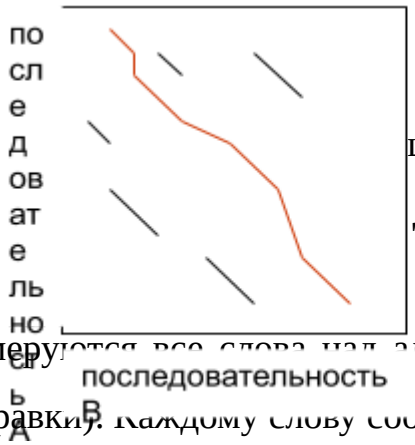


Рисунок 1.5.1. Диагональные линии соответствуют найденным затравочным совпадениям. Красным цветом обозначен результат сборки выравнивания из сегментов.

сходст ю заранее под цы. Г для поиска зат данных фавитом ∑. Хэ базе м. Сначала нумеруются все слова над алфавитом ∑, длина которых равна *k* (*k* - длина затравки). Каждому слову соответствует строка в хэш-таблице, номер которой равен номеру этого слова. В каждой строке хэш-таблицы хранятся списки ссылок на последовательности и на позиции в последовательностях, где встречается слово, образующее затравочное сходство с данным словом.

1.6. Чувствительность и избирательность эвристических методов

Алгоритмы, использующие для выравнивания точные затравки, сталкиваются со следующей проблемой: чем больше длина затравки, тем меньше гомологичных фрагментов можно обнаружить. Хотя алгоритм при этом будет работать быстрее. Однако использование коротких затравок значительно увеличивает время вычислений.

Данная проблема выражается в терминах: чувствительность и избирательность.

Пусть дан список целевых выравниваний. Целевое выравнивание может быть представлено словом над алфавитом A. В случае точных затравок $A = \{0,1\}$, где 1 соответствует паре совпадающих символов, а 0 – паре несовпадающих символов. *Чувствительностью (sensitivity)* затравочного метода называется доля найденных целевых выравниваний. Чувствительность показывает, насколько точно данный алгоритм находит выравнивание. *Избирательностью (selectivity)* затравочного метода называется вероятность не обнаружить два случайных фрагмента сходными. Чем выше избирательность, тем выше скорость работы алгоритма. Чувствительность и избирательность – числа, которые принадлежат промежутку [0,1].

Тогда алгоритмы, использующие длинные точные затравки, имеют низкую чувствительность, но высокую избирательность. А алгоритмы, использующие короткие точные затравки, имеют высокую чувствительность, но низкую избирательность.

Таким образом, эффективность затравочного метода (затравки) характеризуется двумя параметрами: чувствительность и избирательность. Отсюда возникает проблема построения затравки, имеющей наибольшую чувствительность при данной избирательности. В работе [21] был предложен общий алгоритм вычисления чувствительности затравок, основанный на построении конечных автоматов.

1.7. Новые виды затравок

Помимо точных затравок, были предложены разреженные (spaced seed) [10-12], векторные (vector seed) [14, 15], множественные (multiple seed) [13, 15-18] и классификационные затравки (subset seed) [20-22].

Разреженные затравки представляют собой слово над алфавитом {0,1}. *Весом* разреженной затравки называется число единиц в ней. Приведем пример. Пусть дана затравка $\pi = 1110111$ и последовательности $s = ACTGCCT$ и $t = ACTTCCT$. Тогда данные последовательности образует затравочное сходство относительно π. Будем говорить, что π *находит* (обнаруживает) выравнивание (*S,T*) последовательностей *s* и *t*. Таким образом, разреженные затравки являются обобщением точных.

Впервые разреженные затравки появились в алгоритме PatternHunter. В работе [10] показано, что благодаря их применению можно добиться более высокой чувствительности и избирательности, чем с помощью точных затравок. Так же они используются в алгоритмах WABA [7], BLAT [8] и BLATZ [9]. WABA использует затравки вида 110110… Алгоритм WABA предназначен для выравнивания нуклеотидных последовательностей. Поэтому применение таких затравок обосновывается тем, что третья позиция в кодоне менее стабильна, чем первые две. Алгоритм BLAT использует точные затравки, но допускает несколько несовпадений для увеличения чувствительности. В работе [11] предложен метод построения оптимальных разреженных затравок. Несмотря на выше описанные достоинства, разреженные затравки имеют большой недостаток. А именно, они не учитывают различия между парами выровненных символов. Поэтому для выравнивания аминокислотных последовательностей эти затравки хуже применимы, чем для нуклеотидных.

Для решения этой проблемы были предложены *векторные затравки*. Пусть дано выравнивание пары последовательностей. Сопоставим каждой

паре в выравнивании ее вес в матрице замен. Тогда выравнивание будет представлено словом над алфавитом выравнивания: $A = a_1, \ldots, a_n$, где a_i – вес *i*-ой пары в нем. В случае BLOSUM62 для аминокислотных последовательностей, веса пробегают значения от -5 до +12. Векторной затравкой называется пара (*v*, *T*), где *v* – вектор, а *T* – порог. Будем говорить, что (*v,T*) находит затравочное сходство в позиции *i* данного выравнивания, если $v \cdot (a_i, \ldots, a_{i+|v|-1}) \geq T$. Например, дан фрагмент выравнивания A = -1, 3,12. Тогда ((1,1,1), 11) находит этот фрагмент, так как (1,1,1)·(-1,3,12) = 14 > 11.

Векторные затравки являются обобщением разреженных. Произвольную разреженную затравку можно представить в виде векторной (*v,T*), где *v* – сама затравка, а *T* – ее вес. Программа BLASTp использует затравку v = ((1,1,1), 11) для выравнивания последовательностей белков.

Еще одной идеей является использование списка затравок, которые вместе повышают чувствительность. *Множественной затравкой* называется список $Q = \{\pi_1, \ldots, \pi_n\}$, где π_i - некоторая затравка. Будем говорить, что Q находит выравнивание двух последовательностей, если хотя бы одна затравка из списка находит это выравнивание. Впервые эта идея появилась в новой версии алгоритма PatternHunter [13]. Здесь использовалось множество из 16 разреженных затравок веса 11 и длины не более 21.

Экспериментальные результаты показали, что с помощью удачно выбранной множественной затравки можно приблизиться к чувствительности алгоритма Смита-Уотермана при скорости работы алгоритма BLAST.

В работе [15] Браун предложил конструкцию множественных векторных затравок. Как было отмечено ранее, векторные затравки позволяют выравнивать последовательности с более сложными схемами вычисления весов, чем разреженные. Поэтому они применимы для выравнивания аминокислотных последовательностей. С помощью данного

подхода Брауну удалось построить затравки, которые имеют избирательность в 5 раз большую, чем BLASTp, при чувствительности близкой к BLASTp.

Еще одним видом затравок являются *классификационные*. Рассмотрим алфавит выравнивания A; Будем считать, что A содержит символ 1, который будем называть “ полное соответствие” (или match). Тогда классификационной затравкой называется слово над классификационным алфавитом B, такое что

- буквы из B являются подмножествами алфавита A;
- B содержит букву #, которая обозначает подмножество {1}.

Будем говорить, что классификационная затравка $b_1 b_2 \ldots b_m \in B^m$ находит фрагмент выравнивания $a_1 a_2 \ldots a_m \in A^m$, если $\forall i \in [1 \ldots m] \quad a_i \in b_i$.

Классификационные затравки показали хорошие результаты при выравнивании нуклеотидных последовательностей. Они были использованы в алгоритме YASS [20]. Главным преимуществом классификационных затравок является то, что они допускают произвольный алфавит выравнивания.

ГЛАВА 2

ПОСТРОЕНИЕ КЛАССИФИКАЦИОННЫХ ЗАТРАВОК

Данная глава посвящена применению классификационных затравок к выравниванию аминокислотных последовательностей. При построении таких затравок необходимо решить следующие задачи: 1) выбор классификационного алфавита; 2) построение классификационных затравок над выбранным алфавитом. Наша цель – выбрать алфавит таким образом, чтобы затравки над этим алфавитом имели высокую чувствительность при данной избирательности. В этой главе рассматривается три подхода к построению классификационных алфавитов: пороговый алфавит (раздел 2.6); иерархические транзитивный алфавит (раздел 2.8); неиерархический транзитивный алфавит (раздел 2.9).

2.1. Основные определения

Обозначим через $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ алфавит аминокислот.

Пусть $\Pi = \{< x, y > | \, x, y \in \Sigma\}$ - множество пар аминокислот.

Дадим определение классификационного алфавита и классификационной затравки применительно к аминокислотным последовательностям.

Определение 2.1.1.

Классификационным алфавитом называется алфавит B такой, что

- буквы B являются подмножествами Π;

- B содержит букву $\#=\{<x,x>,\forall x\in\Sigma\}$, которая означает полное соответствие двух позиций при выравнивании белковых последовательностей.

- Буквы из B симметричны. Т.е. для аминокислот *x,y* и буквы $\alpha\in\mathrm{B}$ выполняется:

$$\langle x,y\rangle\in\alpha\Leftrightarrow\langle y,x\rangle\in\alpha.$$

Определение 2.1.2.

Классификационной затравкой называется слово над алфавитом B. Пусть дана классификационная затравка $\pi=\alpha_1\square\ \alpha_n$, и даны две последовательности аминокислот $s_1,s_2\in\Sigma^n$. Будем считать, что $s_1\sim_\pi s_2$, если и только если

$$\forall i\in[1..n],\langle s_1[i],s_2[i]\rangle\in\alpha_i.$$

Таким образом, классификационная затравка представляет собой симметричное рефлексивное бинарное отношение на последовательностях из Σ^n.

Пусть дано выравнивание (S_1,S_2) двух последовательностей $s_1,s_2\in\Sigma^n$, и дана классификационная затравка π. Тогда если $s_1\sim_\pi s_2$, то будем говорить, что π *находит* (*обнаруживает*) это выравнивание.

Классификационную букву, которая совпадает с множеством всех пар аминокислот, будем называть Джокер и обозначать символом _.

Пример 2.1.1.

Пусть дана классификационная затравка $\pi = \#\#\_$ над алфавитом $B = \{\_, \#\}$. И пусть даны две последовательности s_1= *AAR и* s_2=*AAS*. Тогда. $s_1 \sim_\pi s_2$. То есть π находит их выравнивание.

2.2. Чувствительность и избирательность

Как было отмечено ранее, эффективность затравки может быть охарактеризована двумя основными параметрами: чувствительностью и избирательностью.

Определение 2.2.1.

Чувствительность затравки – это вероятность обнаружить произвольное целевое выравнивание.

Определение 2.2.2.

Избирательность затравки - это вероятность того, что два случайных слова из Σ^n не будут эквивалентны относительно этой затравки.

Чувствительность показывает точность метода, а избирательность – быстроту. Чувствительность и избирательность затравки могут быть определены через базовую (background) и целевую (foreground) вероятностные модели выравнивания белков.

2.3. Базовое распределение вероятностей

Базовое распределение вероятностей – это распределение пар аминокислот в случайном выравнивании белков. В данной работе в качестве базового распределения будем рассматривать распределение Бернулли. Более сложная Марковская модель рассматриваться не будет. В этом случае базовая вероятность - вероятность встретить данную пару аминокислот в выравнивании при условии, что аминокислоты в этой паре независимы. То есть

$$b(x, y) = b(x)b(y).$$

Где $x, y \in \Sigma$, *b(x)* – вероятность появления аминокислоты *x*.

Определение 2.3.1

Пусть дано базовое распределение аминокислот $\{b_1, \ldots, b_{20}\}$ в рассматриваемых белках. Определим базовую вероятность классификационной буквы α как

$$b(\alpha) = \sum_{(a_i, a_j \in \alpha)} b_i, b_j .$$

Так же определим базовую вероятность классификационной затравки $\pi = \alpha_1 \ldots \alpha_n$ как

$$b(\pi) = \prod_{k=1}^{n} b(\alpha_k) .$$

Заметим, что $b(\alpha)$ вероятность того, что две произвольные аминокислоты находятся в α.

Тогда избирательность буквы α будет равна $1 - b(\alpha)$. И соответственно избирательность классификационной затравки $\pi = \alpha_1 \ldots \alpha_n$ будет равна $1 - b(\pi)$.

2.4. Целевое распределение вероятностей

Пусть дан список выравниваний белков длины *n*. Тогда *целевая вероятность* пары аминокислот - вероятность увидеть данную пару среди этих выравниваний. При целевом распределении пары аминокислот в выравнивании зависимы.

Определение 2.4.1.

Пусть дано целевое распределение пар аминокислот $\{f_{1,1}, \ldots, f_{i,j}, \ldots, f_{20,20}\}$ в рассматриваемом списке выравниваний. Определим целевую вероятность классификационной буквы *α* как

$$f(\alpha) = \sum_{(a_i, a_j \in \alpha)} f_{i,j}$$

.

Так же определим целевую вероятность затравки $\pi = \alpha_1 \ldots \alpha_n$ как

$$f(\pi) = \prod_{k=1}^{n} f(\alpha_k)$$

.

Чувствительность затравки $\pi = \alpha_1 \ldots \alpha_k$ - вероятность того, что произвольное выравнивание без штрафов содержит фрагмент длины *k*, который находит затравка π. К сожалению, чувствительность нельзя вычислить, подобно избирательности. Так как пары аминокислот в выравниваниях зависимы.

В работе [21] был предложен общий алгоритм для вычисления чувствительности при заданном целевом распределении с использованием конечных автоматов.

2.5. Построение классификационных алфавитов

Наша задача – построить классификационные затравки которые при данной избирательности (selectivity) обладают как можно большей чувствительностью (sensitivity). Среди классификационных алфавитов лучшим будем считать тот, в котором затравки при данной избирательности обладают большей чувствительностью.

Идея нашего подхода состоит в следующем. Каждой классификационной букве α сопоставим две вероятности $f(\alpha)$ и $b(\alpha)$, где $b(\alpha)$ – базовая вероятность, а $f(\alpha)$ – целевая вероятность. Так же сопоставим вероятности $f(\pi)$ и $b(\pi)$ каждой классификационной затравке π.

Пусть $\alpha = \{p_1, \ldots, p_k\}$ – классификационная буква, а $p_1, \ldots, p_k$ – пары аминокислот. Тогда

$$b(\alpha) = b(p_1) + \ldots + b(p_k)$$
$$f(\alpha) = f(p_1) + \ldots + f(p_k).$$

Пусть дана затравка $\pi = \alpha_1 \square \alpha_n$. Заметим, что чем меньше базовая вероятность $b(\pi)$, тем больше избирательность π, так как она равна $1 - b(\pi)$. К сожалению, для чувствительности затравки необязательно будет выполняться подобное замечание, так как пары аминокислот в целевых выравниваниях зависимы. То есть, для двух затравок π_1 и π_2 из выполнения условия $f(\pi_1) > f(\pi_2)$ не следует, что чувствительность π_1 больше чувствительности π_2. Однако целевая вероятность затравки может являться хорошей оценкой ее чувствительности. Отсюда возникает следующая идея:

Желательно, чтобы для буквы α вероятность b(α) была поменьше, а вероятность f(α) - побольше.

Определение 2.5.1.

Будем говорить, что буква α_1 *мажорирует* букву α_2, если $f(\alpha_1) \geq f(\alpha_2)$ и $b(\alpha_1) \leq b(\alpha_2)$.

Как было отмечено ранее (раздел 1.7), для разреженных затравок определен вес. Напомним, что разреженная затравка представляет собой слово над алфавитом {0,1}. А вес этой затравки равен числу 1 в ней. Обобщим понятие веса для классификационной затравки. Аналогом единицы для классификационного алфавита является # - рефлексивное множество, состоящее из всех пар аминокислот вида (*a*, *a*). Буква # является минимальной в классификационном алфавите. Исходя из этого, определим веса

$$WeightB(x) = log(Back(x))/log(Back(Match));$$

$$WeightF(x) = log(Fore(x))/log(Back(Match)).$$

2.6. Пороговый алфавит

Для каждой пары аминокислот *p* рассмотрим отношение правдоподобия

$$h(p) = f(p)/ b(p).$$

Упорядочим все пары аминокислот по возрастанию.

Определение 2.6.1.

Пусть дано множество порогов $T = \{t_1, \ldots, t_n\}$. Для каждого порога *t* рассмотрим букву

$$NP(t) = \{p \mid f(p)/b(p) > t\}.$$

Такие буквы будем называть *пороговыми. Пороговым алфавитом* будем называть множество букв

$$NP(T) = \{NP(t_i) \mid t_i \in T\}.$$

Пороговые буквы являются вложенными, то есть

$$\forall t_1, t_2, \text{ где } t_1 > t_2 \quad NP(t_1) \subseteq NP(t_2).$$

Заметим, что такие буквы можно строить не только для пар аминокислот, а и для множества пар символов над любым алфавитом.

Пример 2.6.1.

Рассмотрим множество пар $\{p_1,p_2,p_3,p_4\}$ символов над некоторым алфавитом, имеющих следующие распределения вероятностей *f* и *b*.

	p_1	p_2	p_3	p_4
F	0.3	0.4	0.2	0.1
B	0.2	0.4	0.2	0.2
f/b	1.5	1	1	0.5

Пусть порог *t* будет равен 1 , тогда $NP(t) = \{p_1,p_2,p_3\}$.

Утверждение 2.6.1.

Пусть задано множество порогов $T = \{t_1, \ldots, t_n\}$ и построен алфавит *NP*(*T*). Тогда не существует двух порогов $t_i, t_j \in T$ таких, что $NP(t_i)$ мажорирует $NP(t_j)$ (или наоборот).

Доказательство

Без потери общности будем считать, что $t_i > t_j$. Тогда $NP(t_i) \subseteq NP(t_j)$. Следовательно, $f(NP(t_i)) < f(NP(t_j))$ и $b(NP(t_i)) < b(NP(t_j))$. Отсюда видно, что ни одна из этих букв не мажорирует другую. Утверждение доказано.

Таким образом, пороговые буквы образуют Парето множество среди классификационных букв.

Для вероятностей, соответствующих обучающей выборке BLOSUM62, мы построили пороговый алфавит для всех порогов с шагом 0.05. При этом получилось 34 различные буквы (см. приложение). На рисунке 2.6.1 изображена пороговая буква для *t* = 1.

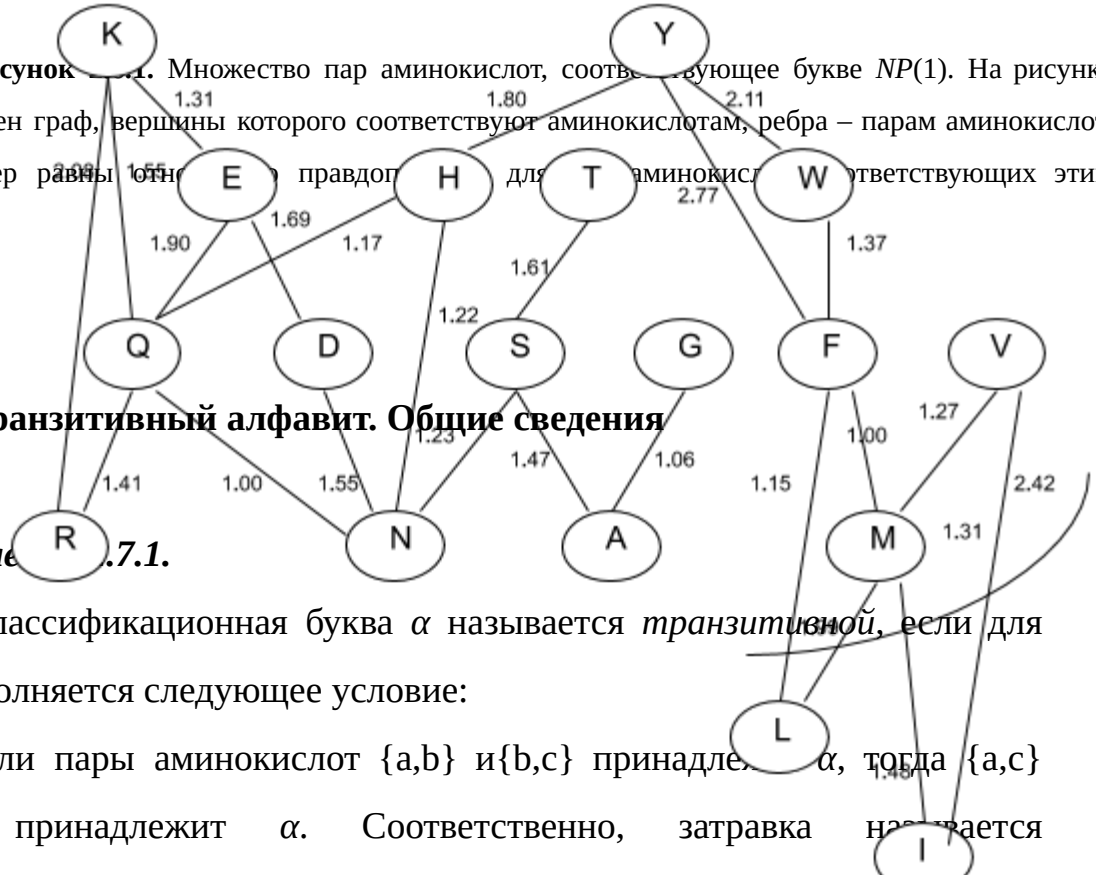


Рисунок 2.6.1. Множество пар аминокислот, соответствующее букве *NP*(1). На рисунке представлен граф, вершины которого соответствуют аминокислотам, ребра – парам аминокислот. Веса ребер равны отношению правдоподобия для аминокислот, соответствующих этим ребрам.

2.7. Транзитивный алфавит. Общие сведения

Определение 2.7.1.

Классификационная буква *α* называется *транзитивной*, если для нее выполняется следующее условие:

если пары аминокислот {a,b} и{b,c} принадлежат *α*, тогда {a,c} также принадлежит *α*. Соответственно, затравка называется *транзитивной*, если она составлена из транзитивных букв. *Размером*

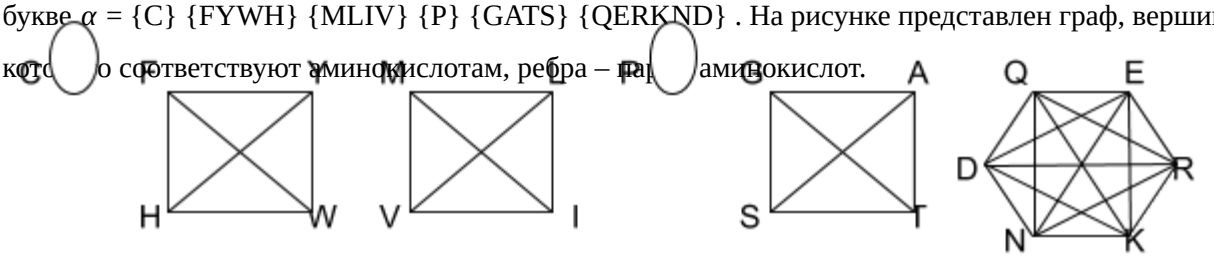
транзитивной буквы называется количество классов в соответствующем разбиении.

Учитывая, что мы рассматриваем только симметричные и рефлексивные буквы, каждая транзитивная буква задает отношение эквивалентности на множестве аминокислот и, следовательно, может быть задана разбиением на этом множестве. При этом букве # соответствует разбиение на одноэлементные классы, а букве Джокер “-“ разбиение, состоящее из единственного класса.

Пример 2.7.1.

Примером транзитивной буквы может служить следующее разбиение на множестве аминокислот α = {C} {FYWH} {MLIV} {P} {GATS} {QERKND}. На рисунке 2.7.1 представлен граф, соответствующий этому разбиению.

Рисунок 2.7.1. Разбиение на множестве аминокислот, соответствующее транзитивной букве α = {C} {FYWH} {MLIV} {P} {GATS} {QERKND} . На рисунке представлен граф, вершины которого соответствуют аминокислотам, ребра – парам аминокислот.



Утверждение 2.7.1.

Транзитивная затравка задает отношение эквивалентности на множестве последовательностей аминокислот, длина которых равна длине этой затравки.

Доказательство

Пусть дана транзитивная затравка $\pi = \alpha_1 \square \alpha_n$. Наша цель – доказать, что отношение $\sim_\pi$ является отношением эквивалентности. Для это необходимо показать, что $\sim_\pi$ рефлексивно, симметрично и транзитивно. Напомним, что если для последовательностей s_1, s_2 длины n выполняется $s_1 \sim_\pi s_2$, то $(s_1[i], s_2[i]) \in \alpha_i$. Тогда для любой последовательности s длины n выполняется $s \sim_\pi s$, так как $(s[i], s[i]) \in \alpha_i$ по определению классификационной буквы. Отсюда, $\sim_\pi$ рефлексивно. Пусть даны две последовательности s_1 и s_2 такие, что $s_1 \sim_\pi s_2$. Так же по определению классификационной буквы $(s_2[i], s_1[i]) \in \alpha_i$. Следовательно, $\sim_\pi$ симметрично. Транзитивность $\sim_\pi$ следует из определения транзитивных букв. Пусть $s_1 \sim_\pi s_2$ и $s_2 \sim_\pi s_3$. То есть $\forall i \quad (s_1[i], s_2[i]) \in \alpha_i \; u \; (s_2[i], s_3[i]) \in \alpha_i$. Тогда $(s_1[i], s_3[i]) \in \alpha_i$. Итак, $\sim_\pi$ рефлексивно, симметрично, транзитивно. Что доказывает утверждение.

Таким образом, последовательности аминокислот длины n разбиваются на классы эквивалентности относительно затравки той же длины π. Такое свойство транзитивных затравок дает выигрыш при поиске в базе данных с помощью хэш-таблицы. Напомним, как строится хэш-таблица на основе базы данных. Для каждой аминокислотной последовательности длины n (ключа) в хэш-таблице хранится множество позиций (номер последовательности и номер позиции в ней) ее вхождения. При поиске в стиле BLAST, когда смотрится совместный вклад нескольких позиций (см. раздел 1.7), помимо

списка вхождений ключа необходимо просматривать списки вхождений гомологичных ему ключей. То есть последовательностей, составляющих с ним затравочное сходство. Например, при поиске с помощью программы BLAST для каждого ключа просматривается в среднем 19.34 смежных списков. При использовании транзитивных затравок, списки, соответствующие эквивалентным последовательностям можно заранее объединить. Таким образом, при каждом поиске необходимо только один раз обращаться к хэш-таблице.

В данной работе будет рассматриваться два подхода к построению транзитивных алфавитов: иерархические (ИТА) и неиерархические (НТА) транзитивные алфавиты. В иерархическом транзитивном алфавите α_i буква вложена в α_{i+1}. В неиерархическом алфавите α_i буква не обязательно вложена в α_{i+1}. Здесь будут описаны алгоритмы для построения обоих алфавитов.

2.8. Иерархический транзитивный алфавит (ИТА)

Определение 2.8.1.

Будем говорить, что транзитивная буква α_1 *вложена* в транзитивную букву α_2 ($\alpha_1 \subseteq \alpha_2$), если разбиение, соответствующее α_1 является подразбиением разбиения, соответствующего α_2. Пусть $P_{\alpha_1}, P_{\alpha_2}$ - разбиения, соответствующие транзитивным буквам α_1, α_2. Тогда

$\forall \sigma \subseteq P_{\alpha_1} \quad \exists \delta \subseteq P_{\alpha_2}$ такое, что $\sigma \subseteq \delta$.

Определение 2.8.2.

Транзитивный алфавит называется *иерархическим*, если в нем буква α_i вложена в α_{i+1}.

Очевидно, что максимальный иерархический алфавит состоит из 20 букв.

Пусть $T = \{\alpha_1, \ldots, \alpha_n\}$ - транзитивный алфавит, тогда

$$\alpha_1 \subseteq \ldots \subseteq \alpha_n .$$

Где α_1 – буква #.

Иерархический алфавит имеет следующие преимущества. Мы можем строить буквы на основе биологически значимых разбиений аминокислот, а так же сузить область поиска транзитивных алфавитов. Причем, иерархические буквы образуют Парето множество среди классификационных букв (доказательство подобно доказательству такого же свойства для порогового алфавита).

Здесь мы предлагаем алгоритм для построения иерархического транзитивного алфавита.

Алгоритм 1.

Мы начнем с разбиения, состоящего из 20 классов. В каждый класс входит только одна аминокислота. Это разбиение соответствует транзитивной букве #. Далее итеративно применим следующую процедуру. Пусть дано текущее разбиение $R = \{C_1, \ldots C_n\}$,

1 Для каждой пары множеств C_k, C_l,

1.1.1 Рассмотрим множество

$$Bridge(C_k, C_l) = \{(a_i, a_j) \mid a_i \in C_k, a_j \in C_l\}.$$

1.1.2 Вычислим

$$Fore\Pr ob(k,l) = \sum \{ f_{ij} \mid a_i \in C_k, a_j \in C_l \}$$ и

$$Back\Pr ob(k,l) = \sum \{ b_i b_j \mid a_i \in C_k, a_j \in C_l \}$$,

1.1.3 Вычислим

$$L(k,l) = Fore\Pr ob(k,l) / Back\Pr ob(k,l)$$.

2 Найдем пару множеств (C_k, C_l), имеющих максимальное $L(k,l)$.

3 Соединив C_k и C_l в новое множество, получим новое разбиение, которое соответствует очередной букве из нашего алфавита.

4 Заканчиваем процедуру, когда текущее разбиение будет состоять лишь из одного множества – множества всех аминокислот.

Таким образом, алгоритм 1 дает нам транзитивный алфавит из 20 букв (см. приложение, рисунок 1).

При построении букв таким способом, мы в текущем разбиении ищем пару классов C_k, C_l, которые максимизируют

$$L(k,l) = Fore\Pr ob(k,l) / Back\Pr ob(k,l)$$.

Альтернативным методом является нахождение пары классов, при объединении которых получается буква, доставляющая максимум функции $L(\alpha) = f(\alpha)/b(\alpha)$. Однако, этот способ дал менее хорошие результаты, чем алгоритм 1.

2.9. Неиерархический транзитивный алфавит (НТА)

В иерархическом алфавите буквы должны быть вложенными. Это требование имеет биологическую мотивацию и, с другой стороны, позволяет сузить пространство возможных букв. Однако, затравка, построенная над неиерархическим алфавитом, так же задает отношение эквивалентности на множестве последовательностей, длина которых равна длине этой затравки. Тем самым, позволяя использовать прямую схему хеширования при поиске в базе данных. Кроме того, построив неиерархический алфавит, мы можем сравнить его с иерархическим. Это позволит понять: сильно ли мы проигрываем, налагая на буквы условие вложенности.

Для построения неиерархического алфавита здесь будет предложен алгоритм 2, который является обобщением алгоритма 1 для построения иерархических букв. Алгоритм 2 состоит из двух стадий: на первой стадии генерируется большое число (~10000) букв кандидатов; на второй из них отбирается алфавит, содержащий ~20 транзитивных букв (не обязательно образующих иерархию).

Алгоритм 2.

Стадия 1.

На первой стадии алгоритма 2 используется стандартная идея генетических алгоритмов. Так же как и в алгоритме 1 начинаем с разбиения, состоящего из 20 классов, которое соответствует транзитивной букве #. На k-ом шаге (k = 1,…,19) для каждой буквы генерируем p потомков, каждый из которых состоит из (20-k) классов.

Для генерации потомков текущей буквы, соответствующей разбиению $R = \{C_1, \ldots C_n\}$, мы используем алгоритм 1. Однако обрабатываем p (вместо 1) пар классов C_k, C_l, имеющих наибольшее значение $L(k,l)$. Объединив каждую пару, мы получим p разбиений(потомков). На k+1 шаге мы выбираем из популяции, полученной на шаге k, q букв, имеющих наибольшие значения $f(\alpha)/b(\alpha)$. Затем для каждой из них генерируем p потомков. Таким образом, после завершения стадии 1 алгоритма 2 получится популяция из pq букв. Если p=50, а q=200, то данная процедура дает около 10000 букв-кандидатов.

Стадия 2.

Выбор 20 транзитивных букв из 10000 основан на двух идеях: (1) буквы должны иметь большое значение $f(\alpha)/b(\alpha)$; (2) буквы должны иметь различные веса. Такой выбор можно делать различными способами.

В данной работе будем производить выбор следующим образом. Отсортируем список букв- кандидатов по значениям весов

WeightB(x) = log(Back(x))/log(Back(Match)

и разобьем его на 20 групп. Пусть w_1 – минимальный вес среди весов букв из списка, а w_2 – максимальный. Тогда в i-ой группе будут находиться буквы, веса которых принадлежат промежутку $[w_1 + (i-1)\tau; w_1 + i\tau]$, где $\tau = 20/(w_2 - w_1)$. Далее в каждой группе выберем букву α, имеющую максимальное значение $f(\alpha)/b(\alpha)$ среди всех букв этой группы. Таким образом, получится алфавит из 20

транзитивных букв. Далее из них выберем Парето множество, где ни одна из букв не мажорирует другую (см. приложение, рисунок 2).

ГЛАВА 3

ЭКСПЕРИМЕНТЫ И РЕЗУЛЬТАТЫ

3.1. Распределения вероятностей

В настоящей работе мы использовали базовое и целевое распределения вероятностей пар аминокислот, соответствующие матрице BLOSUM62 [25]. Эти распределения были получены с помощью измененной программы для вычисления BLOSUM62. Данные вероятностные распределения доступны по адресу http://bioinfo.lifl.fr/reblosum/.

3.2. Построение классификационных алфавитов

С помощью вышеуказанных методов были построены следующие классификационные алфавиты: пороговый алфавит (см. приложение, таблица 1), иерархический транзитивный алфавит (см. приложение, рисунок 1), неиерархический транзитивный алфавит (см. приложение, рисунок 2). Для построения транзитивных алфавитов была написана программа: TRansitive. Программа TRansitive для данных распределений вероятностей пар аминокислот строит неиерархический алфавит с заданными параметрами p и q. Эта же программа строит иерархический транзитивный алфавит при $p=q=1$.

3.3. Одиночные классификационные затравки

Напомним, что классификационная затравка представляет собой слово (группу слов) над классификационным алфавитом B. Мы построили одиночные классификационные затравки для порогового алфавита и транзитивных алфавитов (ИТА, НТА). Каждой затравке ставилось в соответствие два параметра: чувствительность и избирательность. В ходе построения перебирались все затравки, состоящие не более чем из пяти букв. Чтобы сохранить время, было сделано следующее ограничение. Для каждой затравки была вычислена избирательность согласно формуле из раздела 2.3. Затем с помощью программы Iedera (доступной на сайте http://bioinfo.lifl.fr/yass/iedera.php [21]) были вычислены чувствительности рассматриваемых затравок и отобрано Парето множество. То есть множество, в котором не существует затравки, которая одновременно имеет большую чувствительность и избирательность, чем некоторая другая затравка из этого множества. Были рассмотрены различные промежутки для избирательностей.

Для сравнения классификационных затравок с затравкой BLASTp были подсчитаны чувствительность и избирательность затравки BLASTp с порогами 11, 12, 13 на теоретических моделях выравнивании с помощью программы Iedera. Для вычисления чувствительности моделировались фрагменты выравнивания аминокислотных последовательностей длины 16 и 32. Пары аминокислот в этих фрагментах имеют распределение Бернулли. Далее для всех алфавитов были построены затравки по избирательности близкие к BLASTp с порогом 11 и отобрано Парето множество. Избирательности построенных затравок принадлежат промежуткам [0.98830;0.99120] при длине выравниваний 16 и [0.9988; 1] при длине выравниваний 32. Ниже представлены результаты сравнения наиболее близкие к BLASTp с порогом 11. Первое число – избирательность, а второе – чувствительность.

1) При длине выравниваний 16:
   - Затравка BLASTP: 0.999355; 0.557186;
   - Транзитивные алфавиты: 0.998973; 0.557819;
   - Пороговый алфавит: 0.999033; 0.557814.
2) При длине выравниваний 32:
   - Затравка BLASTP: 0.999355; 0.81925;
   - Транзитивные алфавиты: 0.99907; 0.81899;
   - Пороговый алфавит: 0.999099; 0.820116;

Результаты показали преимущества BLASTp перед одиночными классификационными затравками. Однако можно добиться улучшений в чувствительности, если использовать не одиночные, а множественные классификационные затравки. Преимущества BLASTp, благодаря совместному рассмотрению вкладов нескольких позиций; но результатом этого является наиболее сложная схема хеширования для поиска в базе данных.

3.4. Множественные классификационные затравки

Напомним, что множественная затравка представляет собой список одиночных затравок. Множественная затравка обнаруживает фрагмент выравнивания, если этот фрагмент обнаруживает хотя бы одна затравка из списка. В работе [15] впервые были предложены множественные векторные затравки. Здесь мы предлагаем множественные классификационные затравки.

Для разных промежутков избирательностей были построены множественные классификационные затравки, состоящие из 3x – 5 одиночных классификационных затравок длины 3 – 5.

3.5. Результаты сравнения затравок на теоретических моделях выравниваний

Нашей целью является сравнение методов, описанных в этой работе между собой, а так же с методами, предложенными в других работах. Мы сравнили множественные классификационные затравки с затравками, используемыми в алгоритме BLASTp и множественными векторными затравками. Оба последних алгоритма вычисляют вес затравочного сходства как сумму весов позиций в этом сходстве. Что ведет к дополнительным вычислительным трудностям.

Избирательность и чувствительность этих методов также были вычислены с помощью программы Iedera.

Результаты сравнения различных затравок на теоретических моделях выравнивания длины 16 и 32 представлены на графиках зависимостей чувствительности методов от избирательности (см. приложение, рисунок 3, 4). На этих графиках красная кривая и зеленая кривая, помеченная крестиком, соответствуют затравке длины 3 BLASTp и множественной векторной затравке при различных порогах соответственно. Остальные кривые соответствуют множественным классификационным затравкам, построенным вышеописанными методами. Каждая множественная классификационная затравка состоит из 3-5 одиночных классификационных затравок длины 3 – 5. Для каждого из рассматриваемых методов для различных промежутков

избирательностей были построены затравки, вычислена их чувствительность и отобрано Парето множество.

Анализируя графики, можно увидеть, что пороговый алфавит (синяя кривая, помеченная квадратами) дает затравки, близкие по характеристикам к множественной векторной затравке, а так же при некоторых значениях избирательности превосходящие ее. Однако эти затравки не имеют преимуществ перед множественной векторной затравкой при технической реализации, так как в отличие от транзитивных затравок не позволяют использовать прямую схему при хешировании.

Что касается других алфавитов, затравки, построенные над этими алфавитами, сравнимы по характеристикам. Условие неиерархичности транзитивных алфавитов не дает особых преимуществ. На графиках видно, что при длине выравнивания 16 затравки, полученные из транзитивных алфавитов, сравнимы с затравкой BLASTp и имеют небольшое превосходство при больших значениях порогов (11-12) для BLASTp, немного проигрывая при меньших значениях порогов. Когда длина выравнивания равна 32, транзитивные затравки превосходят BLASTp.

Ниже представлены сводные таблицы по характеристикам затравок. Для избирательностей, соответствующих BLASTp с порогами 10, 11, 12 были выбраны затравки над классификационными алфавитами, имеющие большую чувствительность среди остальных. На таблице 3.5.1 для каждой из выбранных затравок представлен процент найденных целевых выравниваний (чувствительность затравки * 100%). На таблице 3.5.2 для каждой из рассматриваемых затравок представлен процент пропущенных целевых выравниваний. На таблице 3.5.2 видно, что процент пропущенных выравниваний для транзитивных затравок приближенно в 1.5 раза меньше, чем процент пропущенных выравниваний для BLASTp с порогами 10, 11, 12.

А пороговые затравки по проценту пропущенных целевых выравниваний сравнимы с множественными векторными.

BLASTp (порог)	BLASTp	Множественная Транзитивная Затравка	Множественная Пороговая Затравка	Множественная Векторная Затравка
10	97.6	97.7	98.3	98.4
11	94.8	95.6	96.2	96.2
12	89.5	91.5	93.1	93.1

Таблица 3.5.1. Процент найденных целевых выравниваний.

BLASTp (порог)	BLASTp	Множественная Транзитивная Затравка	Множественная Пороговая Затравка	Множественная Векторная Затравка
10	2.4	2.3	1.7	1.6
11	5.2	4.4	3.8	3.8
12	10.5	8.5	6.9	6.9

Таблица 3.5.2. Процент пропущенных целевых выравниваний.

3.6. Результаты сравнения на реальных данных

Лабораторией *LORIA/INRIA, Nancy, France* были протестированы построенные классификационные затравки, а так же затравка BLASTp на реальных базах данных выравниваний белков. Для тестов были взяты бызы данных: HOMSTRAD [28], IRMBase (версия 1)[29], OXBench (версия 1.3) [30], PFAM(реализация 22) [31], PREFAB(версия 4) [32], SABmark(версия 1.65)[33] и SMART (версия 4)[34].

На первом шаге, так как все базы данных, кроме OXBench, содержат множественные выравнивания, из них были извлечены множества попарных выравниваний и удалены пропуски. Число выбранных выравниваний варьировалось от n^2 для коротких выравниваний до $\sqrt{n}$ для длинных. Всего было протестировано от 640 (IRMBase) до более чем 25000 (PFAM) выравниваний.

Для экспериментов была взята затравка BLASTp с порогом 11 и множественные транзитивные затравки, имеющие близкую избирательность на теоретически моделях. Заметим, что чувствительность затравки, вычисленная на конкретной базе данных, равна доли выравниваний, которые затравка обнаруживает.

Классификационные затравки показали превосходства на базах данных BaliBASE, PREFAB и PFAM. На OXBench, HOMSTRAD, SMART и SABmark были получены чувствительности, близкие к BLASTp. Каждая из рассматриваемых затравок обнаружила все выравнивания из IRMBase. Но на

SABmark каждой из затравок было обнаружено лишь 15% выравниваний. Это можно объяснить тем, что в SABmark выравнивания основаны на пространственной гомологии белков, а не на гомологии первичных структур. Гомология первичных структур едва достигает 50%.

ЗАКЛЮЧЕНИЕ

В последние время алгоритмы выравнивания нуклеотидных последовательностей были заметно улучшены. В то же время разработке затравок для аминокислотных последовательностей уделялось значительно меньше внимания. Для выравнивания аминокислотных последовательностей существует два основных подхода: BLASTp и векторные затравки. Применение этих методов дает дополнительную трудность при хешировании. Для решения этой проблемы мы построили классификационные затравки. Так же мы ввели понятие транзитивных затравок, которые позволяют организовать поиск в базе данных путем прямого хеширования.

В ходе работы были построены три классификационных алфавита: Пороговый алфавит, транзитивный иерархический алфавит, транзитивный неиерархический алфавит. Затем над каждым из этих алфавитов мы построили затравки и сравнили их с BLASTp и множественной векторной затравкой. Были построены одиночные и множественные классификационные затравки. Одиночные классификационные затравки оказались незначительно проигрывают по эффективности векторной и BLASTp. Это объясняется рассмотрением суммы вкладов всех позиций при поиске затравочных сходств. Однако разница незначительна, причем использование транзитивных классификационных затравок дает дополнительный выигрыш при хешировании. Множественные классификационные затравки показали значительно лучшие результаты, чем одиночные. Затравки построенные над пороговым алфавитом сравнимы по эффективности с множественной векторной затравкой и превосходят остальные. Транзитивные затравки BLASTp. Такие заключения можно сделать исходя из экспериментов, проведенных на теоретических моделях выравниваний и на реальных данных.

Как было отмечено выше, классификационные затравки, построенные над пороговым алфавитом, эффективнее транзитивных затравок. Но к сожалению, они менее применимы на практике, так как не имеют того преимущества, которое имеют транзитивные затравки. Мы строили пороговый алфавит с целью найти ответы на следующие вопросы: Что мы теряем, налагая на затравки условие транзитивности?

Какие затравки можем получить? Исследования показали, что транзитивные затравки не намного хуже затравок, построенных над пороговым алфавитом.

Важно отметить, что предложенные нами алгоритмы не дают оптимального решения. Поэтому проблема построения оптимальных классификационных затравок остается открытой.

В ходе исследования был обнаружен следующий факт: оптимальные затравки могут быть составлены из неоптимальных букв. Изучение этого факта является интересным теоретическим вопросом для дальнейших исследований.

СПИСОК ЛИТЕРАТУРЫ

1. Needlman S, Wunsch C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* (48): 443-453, 1970.

2. Smith TF, Waterman MS. Identification of common molecular sub sequences. *J. Mol. Biol.* (147), 195-197, 1981.

3. Waterman MS. Sequence Alignments in Mathematical Methods for DNA Sequences. Waterman MS (ed.) CRC Press, Boca Raton FL: 53-92, 1989.

4. Lipman D, Pearson W. Rapid and sensitive protein similarity searches. *Science* (227), 1985.

5. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool, *J. Mol. Biol.* 215(3): 403-410, 1990.

6. Altschul S, Madden T, Schaer A, Zhang J, Zhang Z, Miller W, Lipman D. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, *Nucleic Acids Research* 25(17): 3389-3402, 1997.

7. Kent WJ, Zahler AM. Conservation, regulation, synteny and introns in a large-scale C. briggsae-C, elegans genomic alignment. *Genome Res.* 10(8): 1115-1125, 2000.

8. Kent WJ. BLAT- the BLAST-like alignment tool. *Genome Res.* 12(4): 656-664, 2002.

9. Schwartz S, Kent W, Smit A, Zhang Z, Baertsch R, Hardison R, Haussler D, Miller W. Human-mouse alignment with BLASTZ. *Genome Res.* 13: 103-107, 2003.

10. Ma B, Tromp J, Li M. PatternHunter: Faster and more sensitive homology search. *Bioinformatics* 18(3): 440-445, March 2002.

11. Brejova B, Brown D, Vinar T. Optimal spaced seeds for homologous coding regions. *J. Bioinf. Comput. Biol*. 1(4): 595-610, 2004.

12. Keich U, Li M, Ma B, Tromp J. On spaced seeds of similarity search. *Discrete Appl. Math.* 138: 253-263, 2004.

13. Xu J, Brown D, Li M, Ma B. Optimizing multiple spaced seeds for homology search. *In: Proceedings of the 15th Symposium on Combinatorial Pattern Matching, Istanbul (Turkey)*: 47-58, 2004.

14. Brejova B, Brown D, Vinar T. Vector seeds: An extension to spaced seeds allows substantial improvements in sensitivity and specificity. *In: Proceeding of the 3rd Annual Workshop on Algorithms in Bioinformatics (WABI)*: 39-54, 2003

15. Brown D. Multiple vector seeds for protein alignment. In: *Proceeding of 4th Workshop on Algorithms in Bioinformatics (WABI)*, 2004.

16. Sun Y, Buhler J. Designing multiple simultaneous seeds for DNA similarity search. In: *Proceedings of the 8th Annual International Conference on Computational Molecular Biology (RECOMB04), SanDiego (California),* 2004.

17. Brown D. Optimizing multiple seed for protein homology search. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 2(1): 29-38, 2004.

18. Kucherov G, Noe L, Roytberg M. Multi-seed lossless filtration. In: *Proceedings of the 15th Annual Combinatorial Pattern Matching Symposium(CPM),Istanbul (Turkey)*: 297-310, 2004.

19. Noe L, Kucherov G. Improved hit criteria for DNA local alignment. *BMC Bioinformatics* 5(149), 2004.

20. Noe L, Kucherov G. YASS: enhancing the sensitivity of DNA similarity search. *Nucleic Acid Research* 33: W540-W543, 2005.

21. Kucherov G, Noe L, Roytberg M. A unifying framework for seed sensitivity and its application to subset seeds. *J. Bioinf. Comput. Biol* (4), 2006.

22. Peterlongo P, Noe L, Lavenier D, Georges G, Jacques J, Kucherov G, Giraud M. Protein similarity search with subset seeds on a dedicated reconfigurable hardware. In: *Proceedings of the 2nd Workshop on Parallel Computational Biology, Gdansk(Poland)*, 2007.

23. Buhler J, Keich U, Sun Y. Designing seeds for similarity search in genomic DNA. In: *Proceedings of the 7th Annual International Conference on Computational Molecular Biology(RECOMB03), Berlin(Germany)*: 67-75, 2003.

24. Dayhoff MO. Atlas of protein sequence and structure., *DC: National Biomedical Research Foundation, Washing-ton*: 345–358, 1979.

25. Henikoff S, Henikoff JG. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Science*, USA 89: 10915–10919, 1992.

26. Brenner SA, Cohen MA, Gonnet GH. Amino acid substitution during functionally constrained divergent evolution of protein sequences. *Protein Eng* 7: 1323–1332, 1994.

27. Bahr A, Thompson J, Thierry J, Poch O. BAliBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. Nucleic Acids Research 29(1): 323-326, 2001.

28. Stebbings A, Mizuguchi K. HOMSTRAD: Recent developments of the homologous protein structure alignment database. *Nucleic Acids Research 32:* D203-D207, 2004.

29. Subramanian A R, Weyer-Menkho J, Kaufmann M, Morgenstern B. DIALIGN-T: an improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics* 6(66), 2005.

30. Raghava G, Searle S, Audley P, Barber J, Barton G. OXBench: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics* 4(47), 2003.

31. Finn R, Mistry J, Schuster-Bckler B, Griffiths-Jones S, Hollich V, Lassmann T, Moxon S, Marshall M, Khanna A, Durbin R, Eddy S, Sonnhammer E, Bateman A. PFAM: clans, web tools and services. *Nucleic Acids Research* 34: D247-D251, 2006.

32. Edgar RC. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* 32(5): 1792-1797, 2004.

33. Van Walle I, Lasters I, Wyns L. SABmark a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics* 21(7): 1267-1268, 2005.

34. Letunic I, Copley R, Pils B, Pinkert S, Schultz J, Bork P. SMART5: domains in the context of genomes and networks. *Nucleic Acids Research* 34(1): D257-D260, 2006.

35. Finkelstein A, Roytberg M. Computation of biopolymers: A general approach to different problems. *BioSystems* 30: 1-19, 1993.

36. Daniel G, Brown D, Ming L, Bin M. A tutorial of recent developments in the seeding of local alignment. *J. Bioinf. Comput. Biol.* 4: 819-842, 2004

37. Waterman MS. *Introduction to Computational Biology.* London - New York – Tokyo: Chapman & Hall; 1985.

ПРИЛОЖЕНИЕ

№ буквы	Порог, *t*	Целевая вероятность, *F*	Основная вероятность, *B*	Отношение правдоподобия, *F/B*
1	0.00	1.0000	1.0000	1.0000
2	0.2	0.9997	0.9986	1.0011
3	0.30	0.9857	0.9497	1.0380
4	0.35	0.9539	0.8541	1.1168
5	0.40	0.9220	0.7692	1.1986
6	0.45	0.8863	0.6866	1.2908
7	0.50	0.8491	0.6086	1.3952
8	0.55	0.8356	0.5834	1.4323
9	0.60	0.7985	0.5190	1.5386
10	0.65	0.7600	0.4571	1.6627
11	0.70	0.7272	0.4085	1.7803
12	0.75	0.6988	0.3698	1.8897
13	0.80	0.6513	0.3083	2.1124
14	0.90	0.6336	0.2880	2.1999
15	0.95	0.5862	0.2367	2.4760
16	1.00	0.5492	0.1986	2.7648
17	1.05	0.5437	0.1932	2.8143
18	1.10	0.5321	0.1822	2.9205
19	1.20	0.5192	0.1710	3.0357
20	1.25	0.5100	0.1636	3.1181
21	1.30	0.5054	0.1599	3.1602
22	1.35	0.4782	0.1392	3.4356
23	1.40	0.4765	0.1379	3.4541
24	1.45	0.4715	0.1344	3.5079
25	1.50	0.4540	0.1225	3.7054
26	1.60	0.4403	0.1137	3.8713
27	1.65	0.4309	0.1079	3.9931
28	1.70	0.3983	0.0887	4.4929
29	1.95	0.3882	0.0832	4.6636
30	2.00	0.3784	0.0783	4.8321
31	2.10	0.3659	0.0723	5.0609
32	2.30	0.3641	0.0715	5.0956
33	2.50	0.3402	0.0616	5.5266
34	3.00	0.3317	0.0585	5.6709

Таблица 1. Пороговый алфавит.

{C F Y W H M L I V P G Q E R K N D A T S}

{C F Y W H M L I V} {P G Q E R K N D A T S}

{C} {F Y W H M L I V} {P G Q E R K N D A T S}

{C} {F Y W H M L I V} {P} {G Q E R K N D A T S}

{C} {F Y W H} {M L I V} {P} {G Q E R K N D A T S}

{C} {F Y W H} {M L I V} {P} {G A T S} {Q E R K N D}

{C} {F Y W H} {M L I V} {P} {G} {A T S} {Q E R K N D}

{C} {F Y W H} {M L I V} {P} {G} {A T S} {Q E R K} {N D}

{C} {F Y W} {H} {M L I V} {P} {G} {A T S} {Q E R K} {N D}

{C} {F Y W} {H} {M L I V} {P} {G} {A} {T S} {Q E R K} {N D}

{C} {F Y W} {H} {M L I V} {P} {G} {A} {T S} {Q E} {R K} {N D}

{C} {F Y W} {H} {M L} {I V} {P} {G} {A} {T S} {Q E} {R K} {N D}

{C} {F Y W} {H} {M L} {I V} {P} {G} {A} {T S} {Q E} {R K} {N} {D}

{C} {F Y W} {H} {M L} {I V} {P} {G} {A} {T} {S} {Q E} {R K} {N} {D}

{C} {F Y} {W} {H} {M L} {I V} {P} {G} {A} {T} {S} {Q E} {R K} {N} {D}

{C} {F Y} {W} {H} {M L} {I V} {P} {G} {A} {T} {S} {Q} {E} {R K} {N} {D}

{C} {F Y} {W} {H} {M} {L} {I V} {P} {G} {A} {T} {S} {Q} {E} {R K} {N} {D}

{C} {F Y} {W} {H} {M} {L} {I} {V} {P} {G} {A} {T} {S} {Q} {E} {R K} {N} {D}

{C} {F} {Y} {W} {H} {M} {L} {I} {V} {P} {G} {A} {T} {S} {Q} {E} {R K} {N} {D}

{C} {F} {Y} {W} {H} {M} {L} {I} {V} {P} {G} {A} {T} {S} {Q} {E} {R} {K} {N} {D}

Рисунок 1. Иерархический транзитивный алфавит (ИТА). Каждая строка соответствует отдельной транзитивной букве

{A R N D C Q E G H I L M K F P S T W Y V}

{A R N D Q E G H I L M K F P S T W Y V} {C}

{A R N D C Q E H I L M K F P S T W Y V} {G}

{A R N D Q E H I L M K F S T Y V} {C G P W}

{A R C Q E H I L M K F S T Y V} {N D G P W}

{A R N D C Q E G H K P S T} {I L M F W Y V}

{A R N D Q E G H K S T} {C I L M F W Y V} {P}

{A R N D Q E H K P S T} {C W} {G} {I L M F Y V}

{A R N D Q E K S T} {C P} {G H W} {I L M F Y V}

{A G P S T} {R N D Q E H K} {C} {I L M F W Y V}

{A P S T} {R N D Q E H K} {C W} {G} {I L M F Y V}

{A G S T} {R N D Q E K} {C} {H F W Y} {I L M V} {P}

{A S T} {R N D Q E K} {C H} {G} {I L M V} {F W Y} {P}

{A S T} {R Q E H K} {N D} {C P} {G} {I L M V} {F W Y}

{A S T} {R Q K} {N H} {D E} {C} {G} {I L M V} {F W Y} {P}

{A} {R Q K} {N} {D E} {C} {G} {H} {I L M V} {F Y} {P} {S T} {W}

{A} {R K} {N} {D E} {C} {Q H} {G} {I L V} {M} {F Y} {P} {S T} {W}

{A} {R Q K} {N D} {C} {E} {G} {H} {I V} {L M} {F W Y} {P} {S T}

{A} {R K} {N D} {C} {Q} {E} {G} {H} {I V} {L M} {F W Y} {P} {S} {T}

{A} {R K} {N} {D} {C} {Q} {E} {G} {H} {I V} {L} {M} {F Y} {P} {S} {T} {W}

{A} {R} {N} {D} {C} {Q E} {G} {H} {I} {L} {K} {M} {F W Y} {P} {S} {T} {V}

{A} {R} {N} {D} {C} {Q} {E} {G} {H} {I} {L} {K} {M} {F} {W} {Y} {P} {S} {T} {V}

Рисунок 2. Неиерархический транзитивный алфавит (НТА). Каждая строка соответствует транзитивной букве.

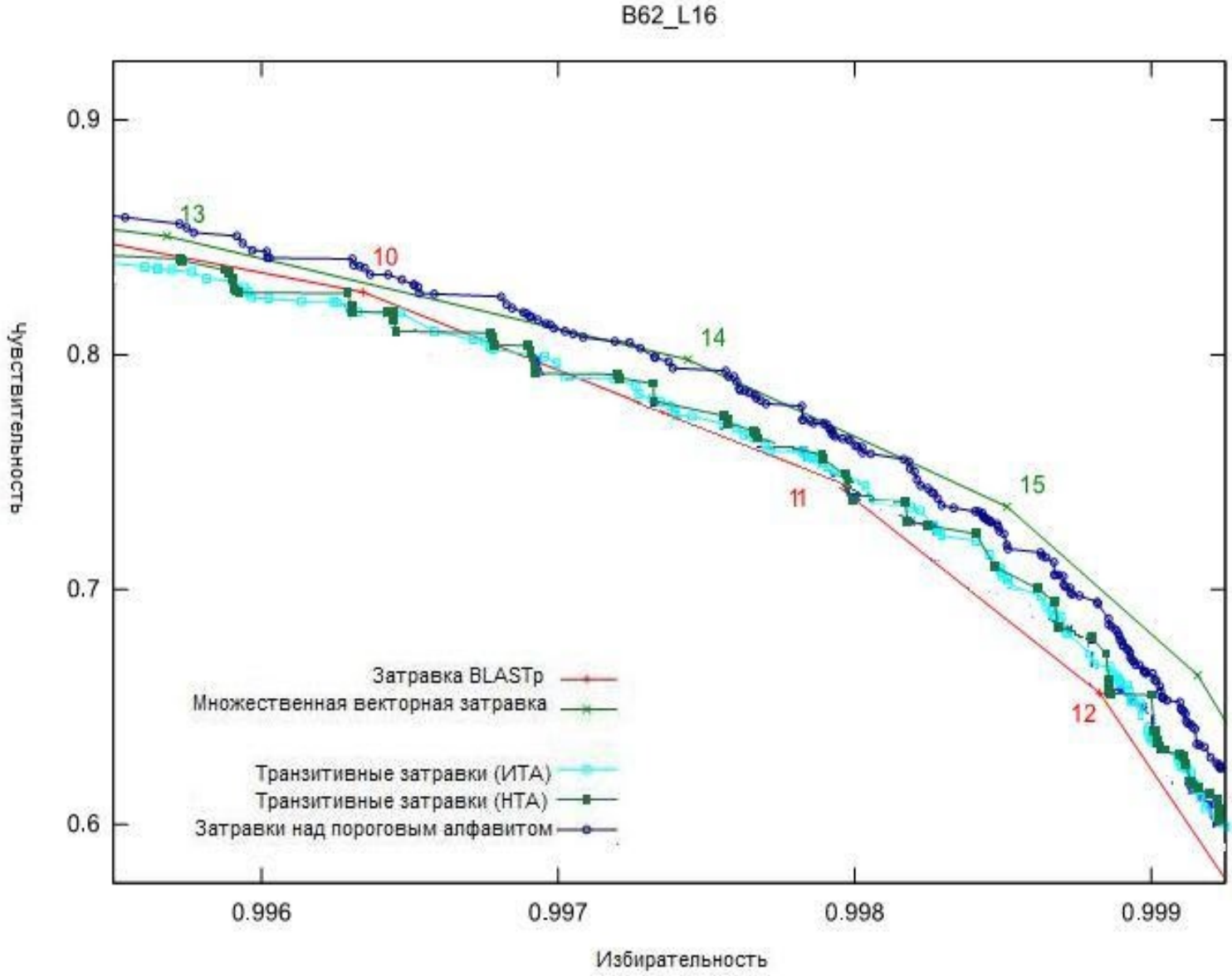


Рисунок 3.

Результаты сравнения методов на теоретической модели выравнивания длины 16.

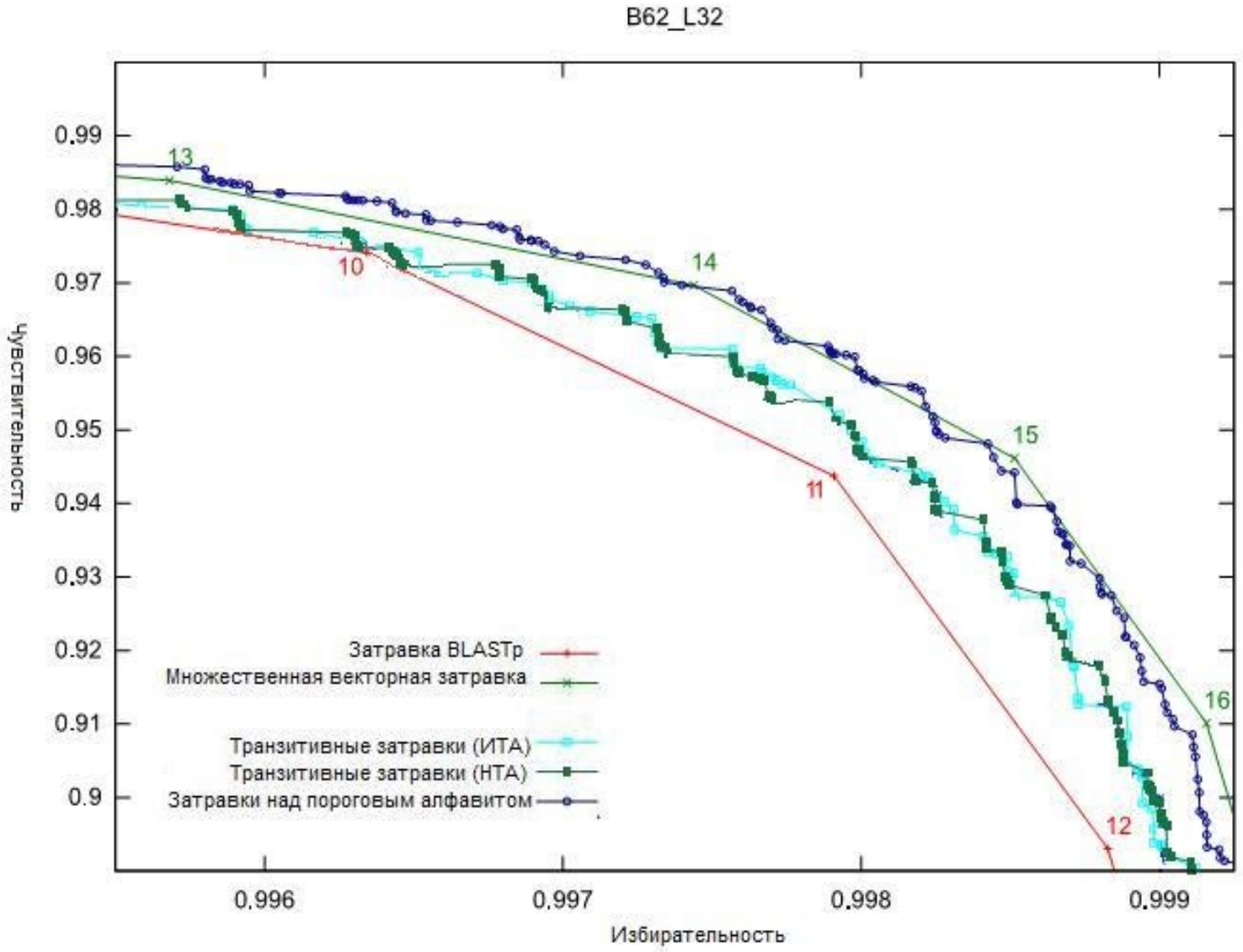


Рисунок 4.

Результаты сравнения методов на теоретической модели выравнивания длины 32.

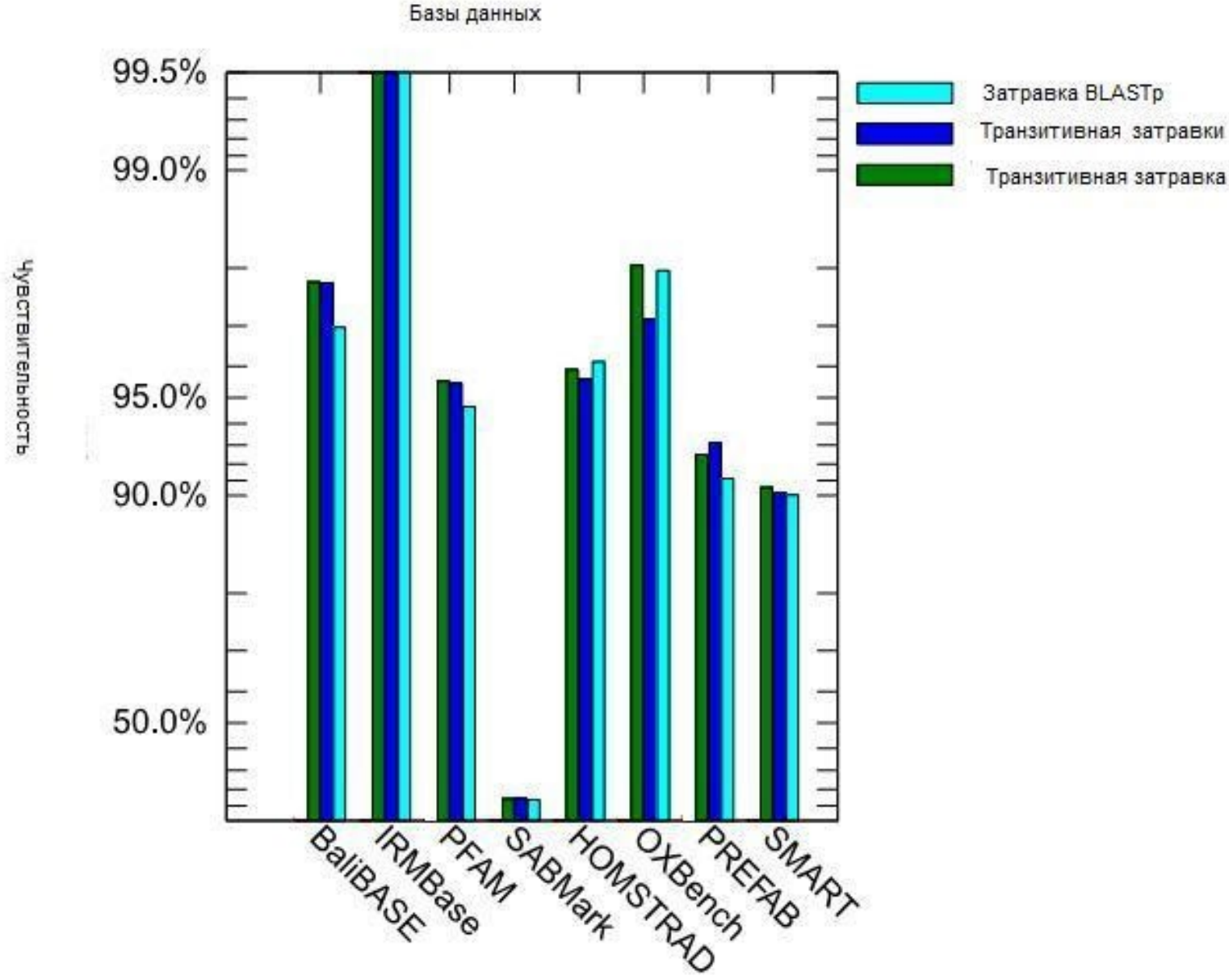


Рисунок 5. Результаты сравнения методов на реальных данных.