РОССИЙСКАЯ АКАДЕМИЯ НАУК

УРАН Институт математических проблем биологии РАН

На правах рукописи

Ройтберг Михаил Абрамович

АЛГОРИТМЫ СРАВНИТЕЛЬНОГО АНАЛИЗА ПЕРВИЧНЫХ СТРУКТУР БИОПОЛИМЕРОВ

03.00.28 - Биоинформатика

Диссертация на соискание ученой степени
доктора физико-математических наук

Пущино 2009

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

1. Общая характеристика работы

1.1. Тема исследования.

Последовательности (первичные структуры) нуклеиновых кислот и белков – наиболее массовый и наиболее доступный в настоящее время вид молекулярно-биологических экспериментальных данных. Эти данные начали накапливаться со второй половины 70-х годов. За последние 30 лет масштабы получения новых последовательностей нуклеотидных кислот (секвенирования) выросли почти в миллиард раз. В настоящее время расшифровано около 900 полных геномов прокариотических организмов и около 100 геномов эукариот, включая геномы человека, мыши, шимпанзе и некоторых других млекопитающих.

Особенностью этого рода экспериментальных данных является то, что темпы их получения с самого начала опережали темпы обработки данных; анализ биологических последовательностей изначально был отделен от их получения. Задачи исследования последовательностей (изучение их внутреннего строения, связи с пространственной структурой, функциональной аннотации, изучения эволюции) решались различными методами, среди которых можно выделить две группы: (1) методы, анализирующие собственно данную последовательность и (2) методы, проводящих сравнение нескольких последовательностей. В последнем случае речь может идти как о выделении сходных (и, возможно, имеющими сходную биологическую функцию) фрагментов, так и о переносе свойств хорошо изученной последовательности на соответствующие фрагменты сходной с ней последовательности, или на всю последовательность в целом (например, при определении типа пространственной структуры белка).

По мере накопления экспериментальных данных сравнительные методы играют все более важную роль. Таким образом, задача разработки алгоритмов сравнительного анализа биологических последовательностей изначально являлась одной из важнейших задач биоинформатики. С другой стороны, по

мере разработки таких алгоритмов все большее значение приобретало исследование адекватности результатов применения алгоритмов содержательным задачам биологии и биофизики.

Практически одновременно с накоплением данных о биологических последовательностях (особенно интенсивно в 60-х годах XX века) происходило развитие прикладной теории алгоритмов – разработка базовых алгоритмов анализа символьных последовательностей и связанных с ними алгоритмов сортировки и алгоритмов на графах. Движущими силами этого развития была, с одной стороны, разработка новых моделей вычислений (конечные автоматы и их варианты с различными видами памяти, в частности, - автоматы Мура, Мили, автоматы с магазинной памятью и др., см. обзор в [152]), а, с другой стороны, появление компьютеров и, следовательно, реальная потребность в обработке значительных (по тем временам) объемов данных. Итоги этого периода были подведены в монографиях Д. Кнута [178,179,180], А. Ахо, Дж. Хопкрофта и Дж. Ульмана [24]. Подробный анализ тенденций и результатов развития прикладной теории алгоритмов в указанный период дан в монографии В. А. Успенского и А. Л. Семенова [18].

С конца 70-х годов XX века аппарат прикладной теории алгоритмов начал применяться для анализа (прежде всего – сравнения) биологических последовательностей. При этом в начале применялись те же алгоритмы, что и для других приложений, например, распознавания речи и поиска сбоев при хранении файлов. Характерно название первого сборника, содержащего работы по биоалгоритмике - «Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison» [283].

Тем не менее, как и в других приложениях математики и теории алгоритмов, достаточно быстро стало понятно, что постановки задач должны максимально учитывать специфику предметной области. Кроме того, собственно алгоритмы поиска (построения) алгоритмически оптимальных объектов должны дополняться исследованием статистической значимости полученного результата и/или его соответствия эталонным (экспериментально

подтвержденным) результатам – для тех случаев, когда эти результаты известны.

Именно с этих позиций в диссертации рассмотрена классическая задача биоинформатики – задача парного выравнивания биологических последовательностей.

1.2. Цели и задачи исследования.

Цель исследования состоит в разработке связанных единым подходом алгоритмов для решения задач сравнительного анализа биологических последовательностей, включая разработку методов оценки достоверности полученных результатов.

В ходе исследования были решены следующие задачи:

1. Созданы эффективные алгоритмы глобального выравнивания символьных последовательностей при различных видах штрафов за делеции, а также в отсутствии явно заданных штрафов.

2. Исследовано соответствие между алгоритмически оптимальными и структурно подтвержденными выравниваниями аминокислотных последовательностей; разработаны алгоритмы, позволяющие повысить точность и достоверность алгоритмически оптимальных выравниваний аминокислотных последовательностей белков относительно эталонных (структурно подтвержденных) выравниваний этих белков.

3. Разработаны специализированные методы анализа нуклеотидных последовательностей (сравнение геномов, сравнение последовательностей РНК с известной вторичной структурой); разработан новый метод предсказания вторичной структуры РНК.

4. Разработаны методы построения затравок для поиска локальных сходств в нуклеотидных и аминокислотных последовательностях; разработаны методы вычисления вероятностей событий, связанных с поиском локальных сходств и обнаружением заданных сигналов.

1.3. Научная новизна и практическая ценность работы.

Научная новизна работы состоит в сформулированном нами едином подходе к решению ряда алгоритмических задач анализа последовательностей, а также в разработанных оригинальных алгоритмах.

Теоретическая значимость работы состоит в исследовании роли штрафов за делеции в задачах выравнивания последовательностей и разработке единого подхода к вычислению вероятностей появления мотивов в последовательностях и их выравниваниях. Это позволило разработать ряд эффективных алгоритмов и исследовать их адекватность биологическим приложениям.

Основными из этих алгоритмов являются:

1) алгоритм построения оптимального выравнивания нуклеотидных последовательностей относительно штрафов за делеции, задаваемых кусочно-линейными функциями и зависящих от контекста в местах разрыва;
2) алгоритм построения множества Парето-оптимальных выравниваний относительно векторной весовой функции сопоставлений;
3) алгоритм выравнивания аминокислотных последовательностей белков с учетом их вторичной структуры
4) алгоритм предсказания внутренних циклов во вторичной структуре РНК;
5) алгоритм выравнивания вторичных структур РНК при заданной вторичной структуре;
6) алгоритм выравнивания геномов;
7) алгоритм вычисления чувствительности затравок для поиска локальных сходств;
8) алгоритм построения затравок для поиска локальных сходств в нуклеиновых и аминокислотных последовательностях.

Для большинства предложенных алгоритмов созданы реализующие их общедоступные компьютерные программы, которые используются в работах как отечественных, так и зарубежных исследовательских групп

1.4. Апробация работы и публикации

Материалы диссертации докладывались на международных и всероссийских конференциях и семинарах, в том числе: Московском семинаре по компьютерной генетике; отчетных конференциях программы «Геном человека»; II Съезде биофизиков России (Москва, 1999), симпозиуме «The Informatics of Protein Classification» (Университет Ратгерс, США, 2000), III, IV,V,VI международных конференциях по биоинформатике регуляции и структуры генома (Новосибирск, 2002, 2004, 2006, 2008), I, II и III Московских международных конференциях по вычислительной биологии (2003, 2005, 2007), международной конференции Combinatorial Pattern Matching-2004 (Стамбул, Турция), международной конференции Workshop on Algorithms in Bioinformatics (Пальма де Майорка, Испания, 2005), международной конференции «Implementation and Application of Automata» (Прага, Чехия, 2007),международном симпозиуме NETTAB (Варенна, Италия, 2008).

С использованием материалов диссертации автором сделаны доклады в NCBI USA (2000), Georgia Tech (2005), INRIA (2003, 2007), на семинарах в Институте белка РАН, Московском физико-техническом институте, факультете биоинженерии и биоинформатики МГУ и ряд других выступлений.

По материалам диссертации опубликовано 37 статей в реферируемых научных журналах (из них 33 в соавторстве), а также более 50 тезисов докладов и препринтов.

2. Основные понятия и краткий обзор результатов.

Парное выравнивание является базовым методом сравнения биологических последовательностей (первичных структур нуклеиновых кислот и белков). Понятие выравнивания было перенесено в исследование биологических макромолекул из технических приложений, в частности, анализа звуковых сигналов, и теоретических работ по анализу символьных последовательностей. По-видимому, первой работой по выравниванию биологических последовательностей была работа [234]. Обзор начального периода анализа биологических последовательностей можно найти в [183].

Пусть даны символьные последовательности (синоним: слова) *u* и *v* в некотором алфавите.

Определение 1.

Склейка (*сопоставление позиций*) в словах v_1 и v_2 – это пара целых чисел $\tau = (\lambda_1, \lambda_2)$ таких, что $1 \le \lambda_1 \le |v_1|$; $1 \le \lambda_2 \le |v_2|$.

Выравнивание слов *u* и *v*– это тройка $<u, v, S>$, где $S = \{<i_1, j_1>, \ldots <i_n, j_n>\}$ – последовательность склеек, таких, что $1 \le i_1 < \ldots < i_n \le |u|$; $1 \le j_1 < \ldots < j_n \le |v|\}$.

Подразумевается, что i_k-я позиция слова *u сопоставлена* с j_k-й позицией слова *v* $(k = 1, \ldots, n)$, и т.д., а остальные позиции в словах *u* и *v удалены*. Говоря неформально, выровнять две последовательности - это изобразить их друг над другом, возможно, вставляя в обе последовательности пробелы, так, чтобы сделать их длины равными. При этом позиции, оказавшиеся друг над другом, считаются сопоставленными друг другу, а остальные символы – удаленными.

Две данные символьные последовательности можно выровнять многими способами. Алгоритмические выравнивания двух данных последовательностей основаны на понятии *веса* выравнивания – строится *оптимальное* выравнивание, т.е. выравнивание, имеющее максимальный возможный вес. Как правило, оптимальное выравнивание определено не однозначно. Когда говорят, что алгоритм строит оптимальное выравнивание двух последовательностей, имеется в виду, что строится некоторое оптимальное выравнивание.

В простейшем случае, близком к работе [7], вес *W(G)* выравнивания $G = <u, v, S>$ определяется как

$$W(G) = a \cdot M - b \cdot N - c \cdot D \qquad (1)$$

где *M* – количество сопоставлений одинаковых букв, *N* - количество сопоставлений различных букв, *D* – количество удаленных букв. Близкое определение использовано и в работе [234]. В этой работе был предложен алгоритм построения оптимального выравнивания последовательностей длин

m и *n* для приведенного выше определения веса выравнивания; время работы этого алгоритма $O(m \bullet n)$.

Определение 1, неформально говоря, означает, что за сопоставление любых одинаковых букв дается бонус $+a$, за сопоставление различных букв дается штраф $-b$, за удаление буквы дается штраф $-c$. Такое определение веса выравнивания успешно работало в технических приложениях, однако оказалось непригодно при сравнении биологических последовательностей. Это связано с двумя причинами. Во-первых, оценивать одинаково сопоставление любой пары различных символов или любой пары одинаковых символов биологически неадекватно. Во-вторых, определение 1 одинаково оценивает удаление d отдельных символов и фрагмента длины d, что не соответствует природе внесения изменений в биологические молекулы.

Первая проблема была решена путем использования весовых матриц сопоставлений (замен). Такая матрица сопоставляет паре букв *x, y* из используемого алфавита ее вес B[x, y]; при этом величина $a \bullet M - b \bullet N$ в формуле (1) заменяется суммой

$$\Sigma_{k=1,n} B[u[i_k], v[j_k]]$$

весов всех сопоставлений данного выравнивания *G*. Впервые использование весовых матриц при анализе аминокислотных последовательностей было предложено в [97]. Впоследствии было предложено значительное количество других весовых матриц, ориентированных на сравнение последовательностей с различными свойствами (см. обзор в [227]). Наиболее популярны в настоящее время семейства матриц PAM [26] и BLOSUM [144]; каждое из этих семейств включает матрицы, ориентированные на различные уровни сходства между сравниваемыми последовательностями [111].

Теория оценивания штрафов за удаленные фрагменты (делеции) разработана значительно хуже. Это относится как к выбору вида весовой функции, так и к подбору значений ее параметров. В частности, в отличие от матриц весов замен, не выяснена, связь между используемыми штрафами за удаление и степенью сходства между сравниваемыми последовательностями.

В работе [322] было предложено вместо штрафа за удаление символа использовать штраф за удаление фрагмента последовательности, оцениваемый некоторой функцией *f(l)* от длины *l* удаленного фрагмента. Эта функция называется *весовой функцией делеций.* Таким образом, определение веса выравнивания $G = <u, v, S>$, *где* $S = \{<i_1, j_1>, \ldots <i_n, j_n>\}$ согласно [322] выглядит так:

$$W(G) = \Sigma_{k=1,n} B[u[i_k], v[j_k]] - \Sigma_{k=0,n} f([i_{k+1} - i_k - 1]) - \Sigma_{k=0,n} f([i_{k+1} - i_k - 1]) \quad (2)$$

где B – матрица сопоставлений, f – весовая функция делеций; $i_0 = j_0 = 0$; $i_{n+1} = |u|$; $j_{n+1} = |v|$.

В цитированной работе [322] был предложен алгоритм построения оптимального выравнивания последовательностей длин m и n при условии, что вес делеции $f(k)$ удовлетворяет условию $f(k) \le c \bullet k$ со временем работы $O(m \bullet n \bullet (m+n))$. Несколько позже (см. [138]) были предложен алгоритм построения оптимального выравнивания для т.н. аффинных весов делеций, т.е. весов вида $f(k) = a + b \bullet k$ с временной сложностью $O(m \bullet n)$.

В работе [11], см. ниже раздел 2.1, мы предложили рассмотреть новый класс весовых функций делеции – класс кусочно-линейных функций, учитывающих зависимость от контекста в местах разрыва последовательности; далее этот класс для краткости называется классом кусочно-линейных функций. Этот класс существенно богаче класса линейных функций. Для предложенного класса весовых функций делеции нами создан алгоритм построения оптимального выравнивания с временной сложностью $O(p \bullet m \bullet n)$, где p – количество интервалов линейности, что существенно лучше временной сложности алгоритма [322].

В дальнейшем нами были получены две группы результатов, касающиеся выравнивания последовательностей и выбора весовых функций делеции. В этих работах мы даем как новые постановки задач, так и алгоритмы их решения.

Первая группа связана с тем, что оптимальное выравнивание аминокислотных последовательностей белков в ряде важных случаев по внутренним причинам не может воспроизвести выравнивание этих белков,

согласованное с их пространственной структурой (см. [299] и раздел 3.1); то же относится и к выравниванию РНК. Выход из этой ситуации может быть найден на пути привлечения дополнительной биологической информации.

В работах [8, 41], см. разделы 3.2 и 3.3, даны постановки задач по выравниванию биологических последовательностей, обогащенных сведениями об их пространственной (вторичной) структуре, и предложены эффективные алгоритмы решения этих задач. К этой группе результатов примыкает разработанный нами алгоритм предсказания вторичной структуры РНК. Отметим, что в случае белков существенное улучшение качества выравнивания происходит даже при использовании теоретически предсказанной вторичной структуры [8]. Т.е. улучшение качества выравнивания достигается не за счет прямого привлечения дополнительных экспериментальных данных о сравниваемых последовательности, а за счет использования теории строения белков, отраженной в методах предсказания их пространственной (вторичной) структуры.

Во второй группе работ мы даем новые постановки задач выравнивания, в которых понятие веса делеции не используется вообще. Это задачи о векторных весах выравнивания и об иерархическом выравнивании геномов. В работе [16] введены понятия векторной оценки веса выравнивания и множества Парето-оптимальных выравниваний. Это снимает вопрос о выборе параметров весовых функций и позволяет продемонстрировать вероятностную природу весовой функции делеции, см. разделы 2.3 и 2.4.

Работы [276, 242] посвящены задаче о выравнивании геномов. Мы показываем (см. раздел 4.1), что, вопреки сложившейся точке зрения, при выравнивании геномов задача о построении оптимального (относительно некоторой весовой функции) выравнивания не является биологически адекватной. В качестве формализации задачи выравнивания мы предлагаем задачу иерархического построения цепочки коллинеарных локальных сходств. Последующее использование программы OWEN (см. [56, 264, 37] и ряд других работ) подтвердили эту точку зрения. Нами также был разработан метод распознавания кодирующих участков генома по построенному

геномному выравниванию [38] и методы построения затравок для поиска локальных сходств [186, 187], см. разделы 4.3, 4.4. Метод вычисления чувствительности затравок предложенный в [187] может быть применен для решения широкого класса задач, связанных с вычислением вероятностей семейств слов [60, 14], см. разделы 5.1, 5.2, 5.3.

3. Структура работы

Диссертация состоит из введения, пяти глав, заключения и списка литературы. Глава 1 посвящена обзору литературы, отмечена связь проанализированных работ с предметом исследования диссертации. Результаты работы представлены в главах 2 – 5. Во второй главе дан теоретический анализ проблемы выравнивания двух символьных последовательностей, в частности, - проблема выбора штрафов за удаление фрагментов. Результаты, представленные в этой главе, могут быть применены не только к биологическим последовательностям, но и к последовательностям иной природы.

В главе 3 рассматривается более специальная задача - выравнивание биологических последовательностей. Центральная тема этой главы – учет пространственной структуры сравниваемых молекул при сравнении их первичных структур (последовательностей). Раздел 3.1 посвящен изучению связи между биологически корректными выравниваниями аминокислотных последовательностей и выравниваниями, полученными с помощью алгоритма Смита-Уотермана - наиболее распространенного в настоящее время алгоритма построения оптимального выравнивания последовательностей. В этом разделе вводится понятие биологически корректного выравнивания и указываются причины, не позволяющие восстановить это выравнивание методом Смита-Уотермана, если сходство между сравниваемыми последовательностями невелико. В разделе 3.2 предложен алгоритм, позволяющий преодолеть эти ограничения за счет привлечения информации о пространственной структуре белков. Показано, что существенное улучшение качества выравнивания достигается даже при использовании теоретически

предсказанной структуры. В разделе 3.3. тема построения оптимального выравнивания биологических последовательностей, обогащенных сведениями о структуре соответствующих макромолекул, продолжена применительно к последовательностям РНК. Предложен алгоритм построения оптимального выравнивания последовательностей РНК, обогащенных сведениями об их вторичной структуре. В разделе 3.4 приводится алгоритм предсказания оптимальной вторичной структуры для заданной последовательности РНК.

Рассмотренные в главах 2 и 3 алгоритмы оптимального выравнивания имеют квадратичную временную сложность – их время работы (в худшем случае) пропорционально произведению длин последовательностей. Такое время слишком велико для многих биоинформатических приложений, в частности, - для сравнения синтенных фрагментов геномов (длина ~ 10^7 нуклеотидов. Глава 4 посвящена выравниванию последовательностей, не связанному с оптимизацией весовой функции, и возникающим в этой связи алгоритмическим задачам. В разделе 4.1 представлен иерархический алгоритм геномного выравнивания OWEN, основанный на построении систем коллинеарных локальных сходств. Разделы 4.2 и 4.3 посвящены поиску локальных сходств с помощью т.н. затравок – наиболее распространенному и быстродействующему из известных методов построения локальных сходств.

Глава 5, заключительная глава диссертации, посвящена применению обобщенных статистических сумм в задачах биоинформатики. Наиболее распространенной интегральной характеристикой множества объектов является его вероятность. В разделе 5.1.1 приведены достаточные условия, при которых вычисление вероятности множества символьных последовательностей может быть сведено к вычислению обобщенной статистической суммы. Этот подход применен к вычислению чувствительностей затравок (см. главу 4 и раздел 5.2) и вероятностей обнаружения сигналов в последовательностях (раздел 5.3).

Глава 1. Обзор литературы.

1.1. Выравнивание символьных последовательностей.

Парное выравнивание является базовым методом сравнения биологических последовательностей (первичных структур нуклеиновых кислот и белков). Оно лежит в основе многих методов биоинформатики, используемых при функциональной аннотации геномов [39, 5, 221, 130, 129], анализе и предсказании пространственной структуры белков и нуклеиновых кислот [62, 192, 257, 96, 53, 174] и др. Парное выравнивание является ядром (функциональным параметром) многих методов множественного выравнивания (см., например, [239, 302, 170, 112, 113,114, 277]). Программы парного выравнивания являются неотъемлемыми составляющими биоинформатических пакетов программ [233; 266]

Понятие выравнивания было перенесено в исследование биологических макромолекул из теоретических работ по анализу символьных последовательностей, в частности, работ, связанных с определением расстояния между последовательностями [7, 308]. Эти работы были связаны как с внутренней логикой развития теории алгоритмов, так и с потребностями технических приложений, в частности, анализа звуковых сигналов [183] и сравнения файлов [267].

Наиболее распространенной из рассматривавшихся в теории алгоритмов близких к выравниванию задач была задача о построении наибольшей общей подпоследовательности двух данных символьных последовательностей (Longest Common Subsequence, LCS) , или, что эквивалентно, о построении минимальной по количеству операций цепочки вставок и удалений символов, преобразующей одну последовательность в другую. В 1974 г. Р. Вагнер и М. Фишер [315] показали, что задача LCS может быть решена методом динамического программирования; при этом как для времени работы, так и для необходимой памяти имеет место оценка $O(N^2)$, для простоты мы считаем, что обе последовательности имеют длину $O(N)$. В последующие годы эти оценки были существенно улучшены.

Использование техники «четырех русских» [1], как обычно, позволяет получить небольшой выигрыш во времени: $O(N^2 \, lglgN/lgN)$ в общем случае и $O(N^2/lgN)$, если размер алфавита не растет с ростом N [214]. Д. Хиршберг [148] предложил алгоритм построения наибольшей общей подпоследовательности двух последовательностей с линейной памятью. Алгоритм основан на методе «разделяй и властвуй» [24]; идеи этого алгоритма впоследствии были использованы для построения алгоритмов выравнивания биологических последовательностей [229].

В ряде работ представлены адаптивные (”output dependent”) алгоритмы построения LCS. Эти алгоритмы в худшем случае имеют то же время работы $O(N^2)$, что и алгоритм [315]. Однако, в отличие от этого алгоритма, их время работы определяется не только длинами исходных последовательностей v_1 и v_2, но и степенью их сходства. Поэтому в определенных случаях это время может быть существенно меньше, чем $O(N^2)$, что важно для приложений (см. [23]). В качестве примеров адаптивных алгоритмов укажем алгоритмы Хиршберга [149] и Ханта - Шимански [154]. Пусть L – длина наибольшей общей подпоследовательности двух данных последовательностей v_1 и v_2; $D = N\text{-}L$ и R – количество таких пар позиций (i, j), что $v_1[i] = v_2[j]$. Очевидно, в худшем случае $L = O(N)$; $D=O(N)l$ $R = O(N^2)$. Для времени работы алгоритма Хиршберга верны оценки $O(NL + NlgN)$ и $O(DLlgN)$; для алгоритма Ханта - Шимански - $O((R+N) \, lgN)$; для алгоритма Майерса [228] – $O(ND)$. В работе Апостолико и Гуэрра [34] дан анализ алгоритмов [149] и [154] и предложены их улучшения. Обзор дальнейших (достаточно многочисленных) работ в области построения LCS можно найти в монографии [297] и оригинальной работе [157]. С точки зрения биоинформатических приложений, эти работы представляют лишь теоретический интерес.

Задача выравнивания в приложении к биологическим последовательностям впервые была рассмотрена в работе С. Нидлмана и К. Вунша [234]. Формулировка задачи Нидлмана-Вунша приведена во введении к диссертации; все события (сопоставления, удаления, вставки) в этой работе рассматриваются посимвольно. Алгоритм основан на методе динамического программирования и имеет сложность $O(m \cdot n)$ как по времени, так и по памяти; здесь и далее m и n обозначают длины сравниваемых последовательностей.

В 70-е и начале 80-х годов XX века было опубликовано значительное число работ, посвященных выравниванию биологических последовательностей. При этом одни, подобно работе [234] были напечатаны в биологических журналах, а другие, например, [287] – в математических (в работе П. Селлерса [287] задача о построении выравнивания сформулирована как задача о вычислении расстояния между последовательностями). Обзор работ содержится в монографии М. Уотермана [320].

К основным достижениям этого периода можно отнести следующие результаты:

- введение удаления/вставки фрагмента как отдельной операции изменения («редактирования») последовательностей [322]; предложенный алгоритм имеет время работы $O(m \bullet n \bullet (m+n))$ при условии, что вес $g(k)$ делеции длины k не превосходит величины $c \bullet k$ для некоторой константы c;

- введение понятия матрицы замен аминокислот и создание первых таких матриц [97];

- сведение задачи о поиске наиболее сходных фрагментов двух данных последовательностей (локальное выравнивание) к задаче глобального выравнивания путем введения нулевых штрафов за удаления на концах последовательностей [288,289, 294];

- установление эквивалентности задач минимизации расстояния и максимизации сходства для случая глобального выравнивания [295]; см. также [207]

- введение аффинных весовых функций за удаление/вставку фрагментов и предложение квадратичного алгоритма для построения оптимального выравнивания относительно этих функций; предложенный алгоритм имеет время работы $O(m \bullet n)$ [138; 318];

- постановка задачи о построении всех суб-оптимальных (имеющих несколько меньший вес, чем оптимальное) выравниваний и алгоритм решения этой задачи [317, 76];

- постановка задачи о построении «хорошего» (не обязательно оптимального) выравнивания за время, меньшее, чем $O(m \bullet n)$ [307], [251];

- алгоритмы поиска локальных сходств, основанные на использовании затравок [108, 324] и др.

Как видим, основное внимание в этот период уделялось выработке формальной постановки задачи выравнивания, - с одной стороны, обеспечивающей получение биологически адекватных выравниваний, а с другой – допускающей построение эффективных алгоритмов.

Результаты, представленные в разделах 2.1 и 2.2, (см. [11], [270], [13]) лежат в русле этих тенденций. Предложенный в [11] класс весовых функций делеции существенно богаче класса линейных функций, рассмотренного в [138]. При этом алгоритм построения оптимального выравнивания имеет временную сложность $O(c \cdot m \cdot n)$, где m, n – длины последовательностей; c – коэффициент, зависящий только от весовой функции, что существенно лучше временной сложности алгоритма [322]. Позднее и независимо сходный класс функций был рассмотрен в [220].

Результаты [270] ориентированы на грубое, но быстрое сравнение биологических последовательностей с использованием простейших весовых функций. Такое сравнение полезно для предварительной фильтрации рассматриваемого корпуса последовательностей. Предложенная постановка задачи обобщает задачу построения наибольшей общей подпоследовательности, приближая ее к биологическим реалиям. Предложенный алгоритм, как и цитированные выше алгоритмы [148], [230] и др. является адаптивным, однако основан на других идеях. Он перекликается с алгоритмом Дейкстры [101] для нахождения кратчайшего пути в графе, однако имеет существенное отличие: при выборе продолжаемого начального выравнивания мы используем оценочную функцию для весов продолжений выравниваний (см. раздел 2.2).

Несмотря на предпринятые в указанный период усилия, вопрос о выборе весовой функции выравнивания и, в частности, весов делеций, не был решен. Не решен он и в настоящее время (см., например, недавнюю работу Р. Картрайта [79]), хотя предложенные в [137] аффинные веса вида $a+b \cdot l$ стали стандартом *de facto.* Здесь l – длина делеции (т.е. удаляемого участка); a (Gap Opening Penalty) и b (Gap Elongation Penalty) – числовые параметры. Вопрос об адекватном выборе значений числовых параметров также решен чисто эмпирически (см., например, монографию [345], стр. 126-127). Примечательно, что в обзоре Apostolico A., Giancarlo R. Seqeunce Alignment in Molecular Biology [33] вопрос о выборе весов делеций не обсуждается вообще.

Пытаясь обойти эту проблему, М. Уотерман и его соавторы рассмотрели задачу о построении суб-оптимальных (имеющих вес, отличающийся от оптимального на заданную величину) выравниваний и предложили алгоритм решения этой задачи [317, 76], см. также обзор [311]. Практическое применение этого подхода оказалось затрудненным из-за того, что количество суб-оптимальных выравниваний может быть очень велико даже при очень низком пороге на отличие веса допустимых выравниваний от веса оптимального выравнивания. В то же время рассмотрение семейства выравниваний нашло свое применение при определении степени надежности отдельных выравниваний: мера надежности участка оптимального выравнивания определяется количеством субоптимальных выравниваний, содержащих этот участок [310, 340]. Наиболее последовательным применением такого подхода является вычисление аналога статистических сумм для графа данного семейства выравниваний (вершины графа – пары сопоставленных в одном из выравниваний, ребра соединяют соседние сопоставления, см. гл.2). Эта идея использована в работах М.Лассига и соавторов (см., например, [155]; [184]). Отметим, что подобная техника использовалась нами в задаче распознавания кодирующих участков [15], [222], [300]. Взаимосвязь задачи построения оптимального выравнивания и вычисления обобщенных статистических сумм подробно обсуждается в [120], см. также главу 5 настоящей диссертации. В связи с построением суб-оптимальных выравниванй следует отметить интересную с алгоритмической точки зрения работу [231], посвященную поиску суб-оптимальных путей в произвольных графах.

В работе [123], посвященной выравниваниям иммуноглобулинов, была высказана идея о рассмотрении оптимальных выравниваний как функции весовых параметров, однако алгоритма решения этой задачи предложено не было. Позднее в работах М. Уотермана, Д. Гасфилда и других авторов ([321], [119], [141]) были предложены алгоритмы декомпозиции пространства параметров на области, соответствующие одному и тому же оптимальному выравниванию. После построения такой декомпозиции все возможные оптимальные выравнивания могут быть найдены с помощью стандартного алгоритма с использованием одного значения параметров для каждой области. В работе [141] дана оценка на количество областей декомпозиции.

Недостатком этого подхода является то, что для него равноправны все значения параметров, включая те, которые очевидно противоречат биологическому смыслу. Кроме того, для задач биоинформатики представляют интерес сами оптимальные выравнивания, а не структура пространства параметров, которой уделяется основное внимание при параметрическом подходе.

Подход, представленный в разделе 2.3 (см. [12], [273], [16], [139]) можно рассматривать как двойственный к описанному выше параметрическому подходу. Для того, чтобы получить все потенциально оптимальные выравнивания мы используем многокритериальный подход – каждое выравнивание описывается векторным весом; компонентами такого вектора могут быть, например, суммарный вес сопоставлений, количество удаленных фрагментов и их суммарная длина. Традиционный вес является линейной комбинацией компонент векторного веса. Этот подход имеет ряд преимуществ перед параметрическим подходом. Во-первых, он позволяет избежать трудоемкого явного построения разбиения пространства параметров на «однородные» (т.е. определяющие одно и то же оптимальное выравнивание) области. Во-вторых, в рамках этого подхода естественно формулируется задача о выборе биологически адекватного выравнивания и предлагаются пути к ее решению. Отметим, что на основе многокритериального подхода может быть решена задача построения локальных выравниваний максимальной плотности [36], а также задача построения оптимального выравнивания с заданным количеством удаленных фрагментов [240].

1.2. Использование специфики биологических последовательностей.

1.2.1. Вторичная структура белков и выравнивание аминокислотных последовательностей.

В идеале алгоритмическое выравнивание двух биологических последовательностей совпадает с их «эволюционным» выравниванием, т.е. выравниванием, соответствующим эволюции сравниваемых последовательностей от их общего предка. В таком выравнивании сопоставленные позиции (предположительно) происходят от одной и той же позиции последовательности общего предка [196]. К сожалению,

эволюционные выравнивания нам недоступны. При парном сравнении некодирующих фрагментов ДНК это представляет неразрешимую проблему. Базы эталонных выравниваний нуклеотидных последовательностей появились относительно недавно и относятся только к последовательностям РНК. В случае матричных РНК [78] эталонные выравнивания основаны на сведениях о пространственной структуре кодируемых белков (см. ниже). В остальных случаях они основаны на множественном выравнивании последовательностей РНК с учетом сведений об общей вторичной структуре [301], [127], [325].

Иначе обстоит дело с аминокислотными последовательностями белков. В этом случае выравнивание трехмерных структур (если таковые известны) может служить хорошим приближением к эволюционному выравниванию. Это связано с тем, что на пространственную структуру белков действует сильный стабилизирующий отбор. Вследствие этого отбора гомологичные белки часто сохраняют близкие пространственные структуры, несмотря на то, что их последовательности далеко разошлись [104]. Таким образом, выравнивания, основанные на пространственной структуре белков традиционно используются в качестве эталона («золотого стандарта», golden standard) при оценке качества алгоритмических выравниваний. Основными источниками эталонных выравниваний являются базы BAliBase [46], [303] и PREFUB [112]. Анализ методов выравнивания трехмерных структур лежит вне нашей диссертации, обзор таких методов содержится в [216]

В ряде работ ([313], [103], [146] и др.) было проведено систематическое сравнение алгоритмических и эталонных выравниваний аминокислотных последовательностей. При этом в качестве исследуемого алгоритма использовался алгоритм Смита-Уотермана [295], который является в настоящее время стандартом *de facto* в биоинформатических исследованиях. Эти работы показали такие результаты (различия между различными работами невелики). Для белков, у которых процент совпадений в выравнивании превышает 40%, алгоритмическое и эталонное выравнивание практически совпадают. Для процента совпадений менее 15% - 20% алгоритмическое и эталонное выравнивание практически не имеют ничего общего. Наконец, пары белков со степенью сходства 20% - 40% составляют «серую зону», где качество алгоритмического выравнивания варьируется от пары к паре в широких пределах (см. подробнее раздел 3.1). При этом пары белков из серой

зоны часто имеют различный тип пространственной структуры (fold) [134, 269]. В указанных работах, однако, не исследовались причины, приводящие к широкому разбросу качества алгоритмических выравниваний в серой зоне. Такое исследование было предпринято нами [299]. Оно показало, что причина невысокого сходства алгоритмических и эталонных выравниваний – наличие в последних участков («островов») отрицательного веса, которые не могут быть восстановлены, если алгоритм оптимизирует весовую функцию описанного выше (см. п. 1.1) вида. Вопросы качества алгоритмических выравниваний рассматривались также в работах [2, 9,258].

Таким образом, методы выравнивания, в которых вес выравнивания определяется только с помощью матриц весов замен, построенных на основе статистики выравниваний [146] и штрафов за делеции, не могут восстановить структурное выравнивание белков в практически интересных случаях слабо гомологичных белков. Одним из путей преодоления этого недостатка является непосредственный учет физико-химических свойств белка, прежде всего - его вторичной структуры.

Во-первых, вторичная структура (как часть пространственной структуры белка) существенно более консервативна, чем его первичная структура [278]. Во-вторых, для теоретического предсказания вторичной структуры разработаны весьма эффективные методы. Первые методы предсказания вторичной структуры были предложены в 70-х – начале 80-х гг. и использовали только сведения о последовательности анализируемого белка [10, 85, 199, 128, 259]

Однако точность предсказания ранних методов была ограничена отсутствием большого количества решенных пространственных структур белков. С ростом банков данных белковых последовательностей, а главное – пространственных структур, точность предсказания значительно увеличилась. Наиболее успешные из современных методов предсказания вторичных структур (например, PSIPRED [162], Jpred [92]) основаны на использовании нейронных сетей; при обучении этих сетей используются данные об известных экспериментально определенных структурах. Общая картина современных методов предсказания вторичной структуры белков представлена на сервере EVA [117].

Идея использования данных о вторичной структуре при выравнивании аминокислотных последовательностей рассматривается в работах по биоинформатике с середины 90-х годов. При этом, авторы работ основное внимание уделяли использованию этих выравниваний при установлении дальних гомологий между белками, сведения же о качестве алгоритмически полученных выравниваний белков носили лишь отрывочный характер (см., например, [122]). Можно выделить два основных подхода к использованию вторичной структуры при сравнении аминокислотных последовательностей (в частности, для распознавания пространственной укладки цепи). В рамках первого подхода вторичная структура используется сама по себе. Использовались такие подходы, как алгоритмическое выравнивание вторичных структур [292, 40,213], поиск шаблонов вторичных структур с последующим отбором структур, удовлетворяющих необходимым физическим свойствам (компактность модели, схожая гидрофобность и пр.) [279]; использование вторичной структуры для построения скрытых моделей Маркова (HMM) для белковых последовательностей [99,100].

Второй подход состоит в использовании вторичной структуры совместно с последовательностью. Видимо, первой реализацией такого подхода было использование различных штрафов за делеции внутри и вне участков вторичной структуры [193, 48]. Ши и соавт. [293] использовали указанную идею при выравнивании распознающих профилей (position specific scoring matrix, PSSM). В работах [203, 265] сведения о вторичной структуре использовались при построении матрицы весов замен. Явное выравнивание последовательностей, обогащенных сведениями о вторичной структуре было применено в работе [121].. В этой работе сравниваются последовательности, элементами которых являются пара вида <аминокислота, тип вторичной структуры>. Наиболее полно такой подход реализован в работах Д. Валлквиста и соавт. [316] и И.Ана, Р.Фризнера [32]. В этих работах в качестве веса сопоставления элементов последовательностей используется линейная комбинация аминокислотной составляющей веса выравнивания и его структурной составляющей. Для сопоставления вторичной структуры авторы используют матрицу, полученную из банка данных пространственных выравниваний 3D_Ali [248] (см. раздел 3.2). Работа [32] отличается от работы

[316] тем, что элементы вторичной структуры подразделяются на классы по длине (например, короткая спираль – длинная спираль).

Указанные работы направлены на использование выравнивания при поиске гомологов в базах данных. Вопрос о качестве алгоритмического выравнивания аминокислотных последовательностей (т.е. его сходстве с эталонным структурным выравниванием) в них не рассматривается (исключением является ранняя работа [48], где этот вопрос исследован на примере всего пяти пар белков). В то же время качество алгоритмического выравнивания критически важно во многих приложениях, например, при гомологическом моделировании пространственных структур [280], при опознавании пространственной структуры белка на основании уже известных структур белков [163], при анализе белковых доменов [50], при предсказании функциональной роли отдельных фрагментов белка [62].

В работах [8], [201] мы впервые последовательно исследовали качество выравниваний, полученных с использованием вторичной структуры. Нами предложен и реализован алгоритм STRUSWER, который использует дополнительный бонус за сопоставление одинаковых вторичных структур. Разметка вторичных структур может быть как экспериментальной, так и полученной теоретически. Идейно наиболее близким к нашему методу является метод WFMFL, представленный в цитированной выше работе Валлквиста, Фукуниши, Мерфи, Фадела и Леви. Основные отличия нашего метода от WFMFL состоят в том, что: а) WFMFL использует специальную матрицу весов сходства вторичных структур, полученную из анализа пространственных выравниваний, в то время как в предлагаемом методе STRUSWER учет вторичной структуры определяется всего одним параметром (бонусом за сопоставление вторичной структуры); б) STRUSWER использует предсказание вторичной структуры не в абсолютной, а в относительной форме (для каждого остатка указывается его «склонность» к каждому из возможных видов структуры).

1.2.2. Вторичная структура РНК и выравнивание последовательностей РНК.

Задача выравнивания последовательностей РНК тесно связана с задачей предсказания вторичной структуры РНК. Фактически, есть три задачи, различающиеся по применяемым методам:

- предсказание вторичной структуры;
- выравнивание последовательностей РНК при известной вторичной структуре;
- выравнивание последовательностей РНК с одновременным предсказанием вторичной структуры.

Последняя задача требует одновременного анализа многих последовательностей и поэтому не рассматривается в настоящей диссертации. Для полноты картины отметим первый алгоритм для решения этой задачи, предложенный Д. Санкоффом [282], работу [106], в которой дан подробный анализ разработанных на момент публикации алгоритмов и относительно недавнюю работу [102], основанную на методах теории распознавания образов. Ниже приводится обзор работ, посвященных двум первым задачам. Напомним основные определения.

Определение. 1.2.1 [319].

Пусть S - последовательность длины L в РНК-алфавите {A, C, G, U}. Вторичная структура РНК (или просто структура) над S - это пара $<S, P>$, где P – это такое множество пар целых чисел, что:

1) $<i, j> \in P \Rightarrow 1 \le i < j \le L;$

2) $<i_1, j_1>, <i_2, j_2> \in P \Rightarrow (i_1 \neq i_2)\ \&\ (j_1 \neq j_2)\ \&\ (i_1 \neq j_2)\ \&\ (j_1 \neq i_2)$

Структура $<S, P>$ не содержит псевдоузлов (свободна от псевдоузлов), если дополнительно выполнено

3) $(<i_1, j_1>, <i_2, j_2> \in P)\ \&\ (i_1 < i_2) \Rightarrow ((j_1 < i_2)$ ИЛИ $(j_1 > j_2)$

Говоря неформально, (1) P – это множество спариваний (водородных связей) между нуклеотидами; $<i, j> \in P$ означает, что спарены нуклеотиды в позициях i и j; (2) каждый нуклеотид принадлежит не более, чем одной паре и (3) пары $<i_1,j_1>, <i_2,j_2> \in P$ не образуют псевдоузлов, т.е. отрезки $[i_1, j_1]$ и $[i_2, j_2]$ либо не пересекаются, либо один их них есть часть другого.

Большинство работ, посвященным предсказаниям вторичной структуры и выравниваниям с учетом вторичной структуры (в том числе – наши работы, представленные в главе 3) ограничиваются рассмотрением структур, свободных от псевдоузлов. Это связано с двумя соображениями. С точки зрения информатики, отсутствие псевдоузлов позволяет применять метод

динамического программирования и получать достаточно эффективные алгоритмы. С точки зрения биологии, связи, образующие псевдоузлы, могут рассматриваться, как связи образующие третичную структуру молекулы РНК «поверх» вторичной структуры, свободной от псевдоузлов [80]. Примером этого может быть центральный псевдоузел 16S рибосомальной РНК, который соединяет три основных домена вторичной структуры молекулы [67]. Таким образом, ограничение набора возможных вторичных структур РНК структурами, свободными от псевдоузлов биологически оправдано. Учет возможных псевдоузлов может проводиться после анализа, проведенного без учета псевдоузлов и с использованием результатов такого анализа.

Ниже все рассматриваемые структуры РНК будут структурами без псевдоузлов. Такие структуры естественно представлять в виде скобочных структур (arc-annotated sequences) или деревьев. При этом соответственные скобки стоят в спаренных позициях, а деревья естественно отображают эти скобочные структуры. Подробнее см. [323], [45] а также раздел 3.3.

Это наблюдение ведет к использованию алгоритмов выравнивания деревьев (см., например, [334], [176], [306], [98]) для построения алгоритмов выравнивания последовательностей РНК, обогащенных сведениями об их вторичной структуре (см. [107], [59]). В то же время, как и в случае сравнения «обычных» биологических последовательностей, необходимо выбрать функцию весов удаления фрагментов последовательности (см. выше п.1.1 и главу 2). В цитированных работах, следуя традиционным для теоретического программирования постановкам задач, допускались только посимвольные удаления, что соответствует штрафу вида $g = b \bullet k$ за удаление фрагмента длины k. В работах [147], [41] мы предложили алгоритм, допускающий традиционные для биоинформатики аффинные штрафы, т.е. штрафы вида $g = a + b \bullet k$.

Выше мы рассматривали задачу построения глобального выравнивания последовательностей РНК с известной вторичной структурой. Как и в случае выравнивания не аннотированных символьных последовательностей, задача построения локального выравнивания может быть сведена к ней путем введения нулевых штрафов за концевые делеции (см. выше п. 1.1). Более подробно задача построения оптимального локального выравнивания РНК рассмотрена в работах [42,43,44, 81, 135]. Построение локального

выравнивания деревьев (построение приближенного общего поддерева) рассмотрено в работе [93].

Задача предсказания вторичной структуры РНК, имеющей минимальную возможную энергию, по последовательности РНК – одна из классических задач биоинформатики, эта задача тесно связана с пониманием функционирования РНК [344, 343, 150, 204, 217, 341]. С другой стороны, алгоритмически эта задача тесно связана с задачей выравнивания (см., например, работы Д.Эппштейна и соавт. [115, 116] и нашу (совместно с А.В. Фикельштейном) работу [120].

Начиная с основополагающих работ группы И. Тиноко [304. 305] , методы компьютерного предсказания вторичной структуры (как правило, свободной от псевдоузлов) развивались в нескольких направлениях.

Во-первых, улучшалась адекватность функций, описывающих энергию структуры. Если в первых работах, например, в [241], в качестве оценки свободной энергии рассматривалось количество спариваний нуклеотидов (т.е. образованных водородных связей), то в настоящее время предсказания основываются на достаточно разработанной модели NNM (nearest neighbor model), см. [159]. Эта модель рассматривает структуру РНК как объединение петель (другой термин – циклов, англ. loop) нескольких типов – стекинг (stacking pairs), выпячивания (bulges), шпильки (hairpins), внутренние петли (internal loops), и ветвящиеся петли (multi-branch loops). Модель NNM включает правила вычисления энергии для петель каждого из указанных типов; энергия структуры в целом получается сложением энергий петель. Параметры модели NNM были определены экспериментально [328, 215].

Во-вторых, наряду с единственной оптимальной структурой, рассматривались и другие целевые объекты предсказания (ср. с разделом 1.1). В качестве таковых объектов использовались, например, множество всех спариваний, которые появляются в суб-оптимальных структурах [339, 327], статистическая сумма всех возможных структур и вероятности отдельных спариваний [217, 151], оптимальная структура, не содержащая ветвящихся петель [116] Алгоритмы, которые строят эти объекты, основаны на различных вариантах метода динамического программирования. Наибольшую трудность вызвала задача построения оптимальных внутренних циклов; алгоритм,

вычисляющий энергию всех внутренних циклов последовательности длины L за время $O(L^3)$ был предложен только в 1999 г. [204].

С задачей анализа внутренних петель тесно связана задача построения оптимальной структуры, не содержащей ветвящихся петель. Для этой задачи Д. Эпштейн и соавторы [116] предложили алгоритм, основанный на методе динамического программирования для разреженных матриц (sparse dynamic programming, метод SDP). Алгоритм имеет время работы $O(M \bullet \log^2(L))$ и использует память $O(M)$, где M – это количество допустимых спариваний нуклеотидов и L – длина молекулы РНК. Ввиду неравенства $M < L^2$, это влечет оценку $O(L^2 \bullet \log^2(L))$ для времени работы алгоритма.

К сожалению, этот элегантный алгоритм не оказал влияния на разработку программ предсказания вторичной структуры РНК. Причина в том, что алгоритм [116] предполагает, что энергия внутренней петли является выпуклой функцией от общего числа t неспаренных нуклеотидов в петле. Это предположение не выполняется в модели NNM. Кроме того, часто полезно строить не одну оптимальную структуру, а набор «разумных» оптимальных структур.

В работе [243] мы предложили алгоритм, который преодолевает эти ограничения алгоритма [116]. Во-первых, мы сделали метод SDP применимым для функции оценки энергии внутренних петель, принятой в модели NMM. Во-вторых, мы показали, как с помощью этого метода построить множество всех т.н. «условно-оптимальных» структур. Все предложенные алгоритмы имеют ту же временную сложность $O(L^2 \bullet \log^2(L))$, что и алгоритм [116] и, тем самым, существенно улучшают время работы алгоритма [204].

1.3. Выравнивание геномов, локальные сходства и построение затравок.

1.3.1. Выравнивание геномов.

Задача выравнивания геномов на рубеже веков стала одной из основных задач алгоритмической биоинформатики. Это связано с массовым определением полных последовательностей геномов различных организмов и развитием методов сравнительной геномики [3].

По современным представлениям все организмы происходят от общего предка [105], поэтому определенное сходство можно найти между любыми двумя геномами. Однако структура этого сходства может быть достаточно разнообразной. У прокариотов порядок генов сохраняется весьма слабо, в то время как сходство между отдельными ортологичными генами может быть весьма велико [326]. Это обстоятельство вместе с малой длиной межгенных интервалов приводит к тому, что прокариотический геном естественно рассматривать как неупорядоченное множество генов.

Наоборот, у эукариотов порядок генов эволюционирует достаточно медленно, особенно это относится к многоклеточным организмам. Поэтому в геномах эукариот, даже не являющихся филогенетически близкими, есть крупномасштабные участки коллинеарности, содержащие десятки ортологичных генов, следующих в одном и том же порядке. В частности, существуют такие участки коллинеарности, общие для всех позвоночных [309] и всех цветковых растений [110]. Таким образом, учитывая, что белок-кодирующме экзоны занимают лишь малую часть геномов многоклеточных эукариот, сравнение таких геномов сводится к выравниванию длинных коллинеарных участков, называемых также участками синтении или синтенными участками [219]. Выделение таких участков представляет самостоятельную задачу [143, 332, 63], которая лежит за пределами круга вопросов, рассматриваемых в нашей диссертации.

При сравнении очень близких геномов (пример: человек- шимпанзе) методы традиционного глобального выравнивания дают хорошие результаты [173]. Иначе обстоит дело при сравнении не столь близких геномов (примеры: человек – мышь, человек – фугу). Именно выравнивание таких геномов (точнее – их синтенных участков) имеется в виду, когда говорится о выравнивании геномов; это и будет предметом нашего интереса.

Уровень сходства между фрагментами коллинеарных (синтенных) участков не слишком близких геномов варьируется в широких пределах. Практически совпадающие фрагменты чередуются с фрагментами, не имеющими сколько-нибудь значимого сходства [160, 290]. Биологически, значимые локальные сходства соответствуют либо функциональным единицам, изменчивость которых ограничена естественным отбором, либо

участкам геномов, биохимически не склонным к мутациям («холодные точки») [291].

Как правило, локальные сходства между коллинеарными (синтенными) участками геномов тоже коллинеарны, т.е. следуют в одном и том же порядке в обоих геномах [286]. Иными словами «макроколлинеарность» влечет «микроколлинеарность», поскольку частота эволюционных событий, сохраняющих коллинеарность (замены, делеции, вставки отдельных нуклеотидов) существенно выше, чем частота событий коллинеарность нарушающих (дупликации, инверсии, транспозиции, конвергентная эволюция) [268]. Часто нарушение коллинеарности локальных сходств связано с присутствием мобильных элементов генома или участков малой сложности (например одно- и двух-буквенных повторов). Такие участки могут быть предварительно выявлены и замаскированы [219].

Таким образом, результат сравнения таких пар геномов естественно представлять как цепочку локальных сходств (выравниваний), оставляя промежутки между ними невыровненными [286, 219, 291, 172]. При таком подходе задача разбивается на две: (i) выделение локальных сходств и (ii) построение «остовной цепи» этих сходств, которая и служит аналогом глобального выравнивания при сравнении геномов. Т.е. при решении второй задачи множество локальных сходств и их отдельные характеристики, например, значимость (см. ниже), считаются известными. Алгоритм поиска локальных сходств становится функциональным параметром задачи глобального выравнивания, обзор алгоритмов построения локальных сходств приведен ниже в п. 1.3.2.

Подобная декомпозиция задачи о построении глобального выравнивания была, по-видимому, впервые применена в [324]. В дальнейших работах ([200, 25, 27] указанный подход применялся к различным видам штрафов за соединение сходств; во всех случаях время работы алгоритма составляло $O(m+n+F^2)$, где F – количество предварительно найденных локальных сходств. В работах [115, 116] Д.Эпштейн и соавт. указали на сходство задачи построения остовной цепи локальных сходств и задачи построения наибольшей общей подпоследовательности (см. выше п. 1.1). Ими был предложен метод динамического программирования для разреженных матриц, в котором построение цепи ведется за время $O(m+n+FlogF)$. Метод

Эпштейна и соавт. применим к произвольным линейным, выпуклым и вогнутым функциям весов делеций. Метод Эпштейна и соавт. был обобщен на случай цепи из сходств в нескольких последовательностях в [336, 230]. Для сравнения геномов указанный подход был применен в [286], а также в интересной работе М. Брудно и соавт. [70], в которой в итоговую цепь разрешается включать участки, подвергшиеся инверсии. Из более поздних программ геномного выравнивания укажем YASS [238], в которой используются семейства разреженныех затравок (см. ниже), а также MAUVE [94], которая позволяет строить как парные, так и множественные выравнивания.

Отметим, что все перечисленные методы строят остовную цепь, максимизируя некоторую глобальную весовую функцию. Этот подход, по нашему мнению, оправдан при сравнении белков, где есть возможность верифицировать значения параметров путем сравнения алгоритмических и эталонных выравниваний, и не оправдан при сравнении геномов, где такой возможности нет. В работе [276] (см. раздел 4.1) мы предлагаем иерархический подход к построению остовной цепи, основанный на локальном разрешении конфликтов между локальными сходствами (из двух сходств, которые не могут одновременно быть включены в остовную цепь побеждает более значимое – независимо от других сходств). Это позволяет получить существенный выигрыш в сложности по сравнению с алгоритмами, основанными на глобальной оптимизации без потери качества выравнивания. Программная реализация подхода описана а [242, 244].

1.3.2. Локальные сходства и построение затравок.

Задача поиска локальных сходств (наряду с основной для нашей работы задачей глобального выравнивания) – одна из классических задач биоинформатики. Потребность в решении этой задачи связана с тем, что в процессе эволюции белки часто сохраняют лишь сходство функционально важных участков, а их большая часть сходство теряет. В частности, именно обнаружение локальных сходств является критерием при поиске в базах данных [252].

Впервые задача поиска локальных сходств была сформулирована в работе [181]. Примечательно, что из трех соавторов этой работы один является биологом (сотрудник Department of Embryology, Carnegie Institution of

Washington), один - математик (сотрудник Department of Mathematics,Cornell University) и один - программист (сотрудник Thomas J. Watson Research Center, International Business Machines). В указанной работе ставится задача поиска всех «разумных» локальных сходств. При этом, что такое разумное сходство определяется неявно – путем описания процедуры, которая строит искомые сходства.

Другой подход к проблеме был предложен П. Селлерсом [288,289]) и, видимо, независимо, Т. Смитом и М. Уотерманом [295]. В отличие от работы [181] в этих работах задача поиска локальных сходств ставится как явная оптимизационная задача: построение локальных сходств формулируется как задача построения оптимального глобального выравнивания при нулевых штрафах за делеции на концах последовательностей. При этом в работах Селлерса задача ставится в терминах расстояния между последовательностями, а в работе [295] – в терминах весов выравниваний, что делает алгоритм более прозрачным. Во всех указанных работах штраф за делецию пропорционален ее длине. Однако после того, как О.Гото [138] предложил использовать аффинные штрафы за делеции (см. раздел 1.1), алгоритм [295] был естественно обобщен на случай аффинных весов [319]. В настоящее именно это обобщение называют алгоритмом Смита-Уотермана.

Позднее В. Гоад и М.Канехиса [136] предложили алгоритм, который находит не только локальные сходства наибольшего возможного веса, но и «разумные» сходства, имеющие меньший вес. Подчеркнем, что все указанные алгоритмы заполняют матрицу размером $m \times n$, где m, n – длины сравниваемых последовательностей, поэтому время их работы составляет $O(m \cdot n)$.

Наиболее распространенным приложением поиска локальных сходств является поиск гомологов в базах данных [252]. При этом вознивает вопрос об оценке достоверности найденного сходства. Полуэмпирические критерии достоверности были предложены У. Вилбуром и Д. Липманом [324] и М. Канехисой [166]; эти критерии использованы в программах, разработанными авторами указанных статей. Важным продвижением в области оценки достоверности сходств была работа С.Карлина и С.Альтшуля [167]. В этой работе было показано, что при естественных предположениях распределение веса наилучшего бездедеционного локального сходства в независимых бернуллиевских случайных последовательностях асимптотически является

распределением крайних значений (extreme value distribution). Позднее этот результат был обобщен Р.Моттом [225, 226] на случай сходств содержащих делеции. В работах [167, 226] приведены формулы для расчета параметров распределения крайних значений от используемой матрицы замен в предположении заданного бернуллиевского распределения символов в сравниваемых последовательностях. Позднее эти формулы были адаптированы к более специальным моделям случайных последовательностей [284, 28]. С целью наиболее полного учета особенностей статистики биологических последовательностей У. Пирсон [249] предложил эмпирические методы оценки параметров распределения весов оптимальных локальных выравниваний. Описанные методы реализованы и успешно применяются в программных пакетах BLAST [29,30,31] и FASTA [250, 253].

В работе [108], по-видимому, впервые была применена техника фильтрации области поиска с помощью *затравок;* которая в дальнейшем получила широкое распространение. Эта техника основана на том, что поиск точных совпадений заданной длины в последовательностях длины m и n может быть выполнен за время $O(m+n+R)$, где R – количество найденных совпадений [24]. В задачах биоинформатики этот алгоритм был впервые применен в цитированной работе [181] для поиска совпадений длины 1. Новация работы [108] состояла в предложении искать совпадения произвольной длины s. При $s \sim log(m+n)$ мы получаем (в среднем) линейное время поиска. При этом совпадения меньшей длины могут встретиться даже в случайных последовательностях длин m и n и, поэтому, как правило, не представляют интереса. Описанный подход был развит в работе Р.Вилбура и Д.Липмана [324]; в этой работе были сформулированы две алгоритмические идеи, позднее получившие широкое распространение. Одна из них – выделение т.н. «значимых полос» - областей матрицы сходства, содержащих аномально большое количество «затравочных» сходств и дальнейшее построение выравнивания только внутри выделенных областей. Вторая – строить выравнивание не как точный путь в матрице сходства, а как цепь, составленную из локальных сходств (см. выше п.1.3.1)

В дальнейшем идея поиска с помощью затравок была применена для поиска неточных локальных сходств: точные сходства рассматривались не как цель поиска, а как затравка, вокруг которой следует вести поиск, см. [31] и

другие работы, в частности, - наши работы [270, 271]. При этом, как выяснилось, именно обработка найденных затравок, большинство из которых при поиске неточных сходств оказываются биологически незначимыми, занимает основное время поиска. Отметим, что в ряде работ в качестве признака целесообразности поиска локального сходства использовалось появление двух и более соседних затравочных сходств [31,172].

Таким образом, качество затравки характеризуют две величины: чувствительность (говоря неформально, - доля «разумных» сходств, содержащих затравочное сходство) и избирательность (говоря неформально, - доля «значимых» затравок среди всех обнаруженных затравочных сходств), подробнее о чувствительности и избирательности затравок см. разделы 4.2 и 5.2. Отметим, что затравки со 100%-ной чувствительностью рассматривались в работах по теоретическому программированию (“pattern matching”), см. например, работы [77, 256].

Долгое время казалось, что увеличение избирательности (увеличение числа требуемых совпадений) неизбежно ведет к уменьшению чувствительности и наоборот. Прорыв в построении затравок произошел в 2002 г. В работе [206] , был введен новый класс затравок – т.н. разреженные затравки (gapped seed). Авторы указанной работы разработали программу геномного выравнивания, в которой в качестве затравок использовали наборы совпадений, расположенных не обязательно подряд. Порядок совпадений задавался специальным шаблоном – словом в бинарном алфавите, где один символ, например, ‘1’ соответствует обязательному совпадению, а другой, например, ‘0’ – несущественной позиции. Например, шаблон “1111011» означает, что из семи подряд идущих позиций совпадения должны быть на всех позициях, кроме, быть может, пятой. Избирательность такой затравки (при обычном предположении независимости позиций выравнивания) такая же, как у шести совпадений подряд, а чувствительность – выше. Тесты на реальных нуклеотидных последовательностях подтвердили этот теоретический результат. Сходный класс затравок вне связи с приложением к биоинформатике примерно в то же время был предложен Буркхардтом и Карккайненом в [74]. В работах [71, 171] было показано, что разреженные затравки имеют преимущество перед классическими затравками при поиске гомологов в базах данных.

После пионерских работ Ма, Тромпа и Ли, а также Буркхардта и Карккайнена разреженные затравки были использованы в других программах геномного выравнивания [285, 237]. Кроме того, были предложены новые классы «неклассических» затравок и стратегии их использования - векторные затравки [64. 69]; затравки над небинарными алфавитами [82, 237]; семейства затравок (multi-seeds) [69. 186, 195, 298, 329,330], затравки переменной длины и т.п. [90, 209, 91, 156].

В работах [171, 72, 64, 185, 84] были предложены алгоритмы вычисления чувствительности затравок различных моделей. В работе [187] мы предложили единый подход для вычисления чувствительности затравок различных типов, алгоритмы из цитированных работ являются его частными случаями (см. раздел 5.2). В той же работе предложена т.н. «классификационная» модель затравок, позволяющая легко организовывать индексирование при поиске в базах данных (см. раздел 4.3).

Отметим, что изначально неклассические затравки применялись для поиска локальных сходств в геномной ДНК, однако есть работы, в которых эта техника применяется и к сравнении. белков, см., в частности, работы [69, 337, 205], а также наши работы [275], [126].

1.4. Динамическое программирование

1.4.1. Динамическое программирование и задачи биоинформатики.

Большинство из рассмотренных выше алгоритмов выравнивания основано на методе динамического программирования. Этот метод применяется и в других задачах биоинформатики, например, распознавании белок-кодирующих областей [131,132, 133,134, 274] , разбиении геномной ДНК на статистически однородные фрагменты [260, 261].

Метод динамического программирования был предложен Ричардом Беллманом в начале 50-х годов XX века [51] как метод решения широкого класса оптимизационных задач. Позднее С. Клини [175] применил сходную технику для описания регулярных языков; более прозрачное изложение метода дано в [218]. В монографии А. Ахо, Дж. Хопкрофта и Дж. Ульмана [24] алгоритмы Клини и МакНотона-Ямады были изложены как алгоритмы анализа путей в графах над замкнутыми полукольцами. Близкая техника (т.н.

матричный метод) был применен в 40-е годы Г.Крамерсом и Г.Ванье [182] при вычислении статистической суммы для одномерной модели Изинга [6, 158], однако, по-видимому, статья Крамерса и Ванье была неизвестна авторам цитированных выше работ.

Часто алгоритмами динамического программирования называют и более сложные алгоритмы, например, алгоритмы определения оптимальной вторичной структуры РНК (см. выше п. 1.2.2) и алгоритм CYK для анализа контекстно-свободных грамматик [168, 331, 87]. Эти алгоритмы не сводятся к схеме А. Ахо, Дж. Хопкрофта и Дж. Ульмана. Чтобы прояснить соотношение между этими алгоритмами, с одной стороны, и алгоритмами, которые сводятся к поиску оптимального пути в графе – с другой стороны, А.В. Финкельштейн и автор настоящей диссертации [120] предложили рассмотреть задачу вычисления обобщенной статистической суммы для ациклического гиперграфа, помеченного элементами произвольного полукольца. Отметим, что, аналогично [24], от требования ацикличности можно избавиться ценой требования замкнутости полукольца. Однако, в задачах биоинформатики, насколько нам известно, нет задач, в которых возникают гиперграфы, содержащие циклы.

Формулировка динамического программирования как задачи анализа путей в графе (гиперграфе), ребра которого помечены элементами некоторого полукольца проясняет аналогию между задачами поиска оптимального пути и обобщенной статистической суммы путей, которая возникает, например, при вычислении вероятностей множеств слов (символьных последовательностей). Другие примеры использования этой техники – вычисление вероятностей образования водородных связей между отдельными парами оснований РНК ([217,327] и др.); выделение достоверных фрагментов выравниваний [136, 155], фильтрация экзонов-кандидатов [15, 222, 300], сегментация геномов [260].

Задача вычисления вероятности множества символьных последовательностей возникает в задачах биоинформатики, как правило, в связи с необходимостью оценки значимости результатов поиска сигналов (паттернов, pattern) некоторого вида. При этом множество, вероятность, которого необходимо найти – это множество всех последовательностей данной длины m, которые содержат сигнал, превосходящий заданный уровень

качества *Q*. В нашей диссертации рассмотрены две такие ситуации – вычисление чувствительности затравки для поиска локальных сходств (см. п. 5.2) и задача оценки достоверности найденного кластера регуляторных сайтов (см. п.5.3). Обзор работ, посвященных этим частным задачам, дан в соответствующих разделах.

1.4.2. Динамическое программирование над полукольцами.

В заключение главы 1, следуя нашей работе [120], сформулируем задачу вычисления обобщенной статистической суммы для гиперграфа над полукольцом. К этой задаче сводятся многие алгоритмы биоинформатики и, в частности, многие алгоритмы, рассмотренные в диссертации.

Определение 1.4.1.[24]. *Полукольцо A* – это множество, на котором определены две бинарные всюду определенные операции $\oplus$ и $\otimes$, удовлетворяющие следующим свойствам:

1) операции $\oplus$ и $\otimes$ ассоциативны;
2) операция $\oplus$ коммутативна, коммутативность операции $\otimes$ не обязательна;
3) в *A* есть левый нейтральный элемент *i* относительно операции $\otimes$;
4) операция $\otimes$ дистрибутивна относительно операции $\oplus$:

$$\forall\ a, b, c \in A\ (\ (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)\ )$$

$$\forall\ a, b, c \in A\ (\ c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)\ )$$

Операции $\oplus$ и $\otimes$ обычно называют сложением т умножением.

Определение 1.4.2. [120]. *Ориентированный гиперграф* (ОГ-граф) над полукольцом *A* – это четверка $G = (V, q_0, E, f)$, где *V* – конечное множество вершин; $q_0 \in V$ – *стартовая* вершина; *E* – конечное множество *гиперребер,* т.е. пар вида (v, W), где $v \in V$, *W* – упорядоченный список вершин из *V*; $f: E \to A$ – функция пометок на ребрах. Вершина *v* называется *начальной* вершиной гиперребра $e=(v, W)$, вершины из *W* называются конечными вершинами этого гиперребра; значение $f(e) \in A$ называется *весом* гиперребра *e*.

Вершина *v* ОГ-графа *G* называется *терминальной,* если в *G* нет гиперребра, в котором *v* является начальной вершиной.

Аналогом пути в графе является поток в гиперграфе.

Определение 1.4.3. *Поток (гиперребро)* в ОГ-графе $G = (V, q_0, E, f)$ –это такое конечное упорядоченное помеченное дерево *T*, что

(i) каждый узел дерева T помечен вершиной ОГ-графа G; одна вершина $v \in V$ может соответствовать нескольким узлам дерева T.

(ii) каждому внутреннему (не являющемуся листом) узлу r дерева T соответствует гиперребро $e = (v, W)$, причем узел r помечен вершиной v, а список пометок в сыновьях узла r, взятых в порядке, предписанном деревом T, совпадает со списком W;

Определение 1.4.3. ОГ-граф $G = (V, q_0, E)$ называется *ациклическим,* если в G не существует потока, в котором корень помечен той же вершиной, что и еще какой-либо узел этого потока.

Очевидно, в ациклическом ОГ-графе существует лишь конечное число различных потоков.

Вес $R(T)$ потока T (аналог веса пути на графах) определяется рекурсивно.

Определение 1.4.4. Пусть $G = (V, q_0, E, f)$ – ОГ-граф; T – поток в G. Определим *вес* $R(T)$ потока T следующим образом.

(i) Если поток T состоит из единственного узла, то $R(T) = i$, где i – нейтральный элемент относительно операции $\otimes$ (умножения).

(ii) Пусть корень T помечен гиперребром $e = (v, W)$; $x_1, \ldots, x_N$ – упорядоченный список сыновей корня дерева T; T_k – поддерево дерева T с корнем в узле x_k $(k = 1, \ldots, N = |W|)$. Тогда

$$R(T) = f(e) \otimes R(T_1) \otimes \ldots \otimes R(T_N)$$

Иными словами, вес потока T это произведение (в смысле операции $\otimes$) весов гиперребер, соответствующих узлам T, причем порядок перемножения соответствует левому обходу дерева T.

Определение 1.4.5. Поток T в ОГ-графе G называется *терминальным,* если все его листья соответствуют терминальным вершинам ОГ-графа G. Терминальный поток называется *полным,* если его корню соответствует стартовая вершина ОГ-графа G.

Определение 1.4.6. *Обобщенной статистической суммой* ОГ-графа G называется сумма (в смысле операции $\oplus$) $W(G)$ весов всех его полных потоков.

Как показано в [120], значение обобщенной статистической суммы $W(G)$ ОГ-графа G (V, q_0, E, f); может быть вычислено рекурсивно, начиная от

терминальных вершин G и перебирая остальные вершины в обратном топологическом порядке. Для произвольной вершины $v \in V$ через G_v обозначим подграф ОГ-графа G, порожденный всеми вершинами G, достижимыми из v, т.е. вершинами, которые встречаются в потоках, корень которых соответствует вершине v. Корнем подграфа G_v, естественно, является вершина v. Мы вычислим обобщенные статистические суммы $W(G_v)$ для всех вершин v ОГ-графа G, перебирая вершины в указанном выше порядке. Для любой терминальной вершины v положим $W(G_v)=i$, где i – нейтральный элемент относительно операции умножения $\otimes$ используемого полукольца A. Пусть v – не терминальная вершина G и $e_1 = (v, W_1), \ldots, e_M = (v, W_M)$ – все гиперребра с начальной вершиной v. Чтобы описать рекуррентное уравнение для вычисления $W(G_v)$ нам понадобится еще одно обозначение. Пусть $e = (x, Y)$ – гиперребро в G; $Y = \{y_1, \ldots, y_N\}$. Назовем *специальным весом* $W^*(e)$ гиперребра e величину

$$W^*(e) = f(e) \otimes W(G_{y_1}) \otimes \ldots \otimes W(G_{y_N})$$

Тогда рекурсивное уравнение для вычисления величины $W(G_v)$ записывается в виде

$$W(G_v) = W^*(G_{e_1}) \oplus \ldots \oplus W^*(G_{e_M})$$

Более подробное изложение, включая примеры и доказательства, приведено в работе А.В.Финкельштейна и М.А.Ройтберга [120].

Глава 2. Алгоритмы построения оптимального выравнивания символьных последовательностей.

2.0. Введение.

Тема этой главы - теоретический анализ проблемы выравнивания двух символьных последовательностей; представленные результаты могут быть применены не только к биологическим последовательностям, но и к последовательностям иной природы. Основное внимание уделяется проблеме подбора штрафов за удаление фрагментов.

В разделе 2.1 введено понятие кусочно-линейных функций штрафов (весовых функций) за удаление фрагмента и представлен соответствующий алгоритм построения оптимального выравнивания. Предложенный класс функций является наиболее широким из известных классов весовых функций, для которых построен квадратичный алгоритм построения оптимального выравнивания; под квадратичным алгоритмом понимается алгоритм со временем работы, пропорциональным длинам сравниваемых последовательностей.

Класс весовых функций, рассмотренный в разделе 2.2, наоборот, - простейший класс, при котором задача построения оптимального выравнивания двух последовательностей не сводится к хорошо изученной задаче построения максимальной общей подпоследовательности [148]. Для указанного класса функций предложен адаптивный алгоритм построения оптимального выравнивания. Хотя оценка времени алгоритма в худшем случае является квадратичной, время работы алгоритма для близких последовательностей почти линейно. В частности, если для последовательностей v_1 и v_2 длины m существует оптимальное выравнивание, содержащее t несовпадений и s удаленных символов, то время работы предлагаемого алгоритма составляет $O((t+s)m)$.

Раздел 2.3 посвящен предложенному нами многокритериальному подходу к проблеме парного выравнивания. Этот подход позволяет построить представление всех выравниваний, которые могут быть оптимальными при

заданном виде весовой функции и произвольных числовых коэффициентах (т.н. Парето-оптимальные выравнивания).

В разделе 2.4 обсуждается возможность применения многокритериального подхода п. 2.3 к биологическим задачам. Пусть даны две гомологичные, т.е. происходящие от общего предка, биологические последовательности. Нас будет интересовать, как среди множества Парето-оптимальных выравниваний этих последовательностей выбрать выравнивание, наиболее адекватное с биологической точки зрения. При традиционном подходе к задаче парного выравнивания выбор такого выравнивания производится путем установки штрафов за удаление фрагментов. Результаты п. 2.4 могут трактоваться как подход к обоснованию выбора адекватных значений этих штрафов.

2.1. Глобальное выравнивание при кусочно-линейных штрафах за делеции

2.1.1. Весовые функции удаления фрагмента

Напомним (см. Введение), что *выравниванием* последовательностей *u* и *v* мы называем тройку $<u, v, S>$, где $S = \{<i_1, j_1>, \ldots <i_n, j_n>\}$ – набор пар позиций в словах *u* и *v*, таких, что $1 \le i_1 < \ldots < i_n \le |u|$; $1 \le j_1 < \ldots < j_n \le |v|$. Мы будем считать, что фиксирован некоторый алфавит; в задачах биоинформатики – это, обычно, четырехбуквенный алфавит нуклеотидов или двадцатибуквенный алфавит аминокислот.

Под *весовой функцией удаления сегмента* (синонимы: весовая функция делеции, штраф за делецию) мы будем понимать произвольную функцию $f(v, s, t)$ которая сопоставляет фрагменту $v[s,t]$ слова *v* действительное неотрицательное число. В настоящее время в биоинформатике наиболее употребительны т.н. аффинные весовые функции, т.е. функции вида

$$f(v, s, t) = a \bullet |t-s+1| + b$$

которые зависят только от длины удаляемого сегмента [138]. При рассмотрении некоторых задач, например, при построении оптимального локального выравнивания, штраф за делецию на конце слова (т.е. при $s=1$ или $t = |v|$) полагается нулевым [319].

В работе [11] рассмотрен существенно более широкий класс весовых функций и для этого класса предложен эффективный алгоритм построения оптимального глобального выравнивания. Особенности нашего класса функций состоят в следующем:

1) штрафы за делеции на концах последовательности могут задаваться произвольной функцией;

2) штраф за делецию может зависеть от граничных позиций удаляемого фрагмента;

3) зависимость штрафа от длины фрагмента внутри последовательности может задаваться произвольной кусочно-линейной функцией;

4) штрафы за делеции в каждой из сравниваемых последовательностей могут задаваться по-своему.

Определение 2.1.1. Весовая функция удаления фрагментов называется кусочно-линейной, если она может быть задана формулой:

$$f(v, s, t) = \begin{cases} f_{нач}(v, t-s+1), \text{ если } s=1; \\ f_{кон}(v, t-s+1), \text{ если } t=|v|; \\ f_{внутр}(t-s+1) + r_н(v,s) + r_к(v,t), \text{ если } s>1 \text{ и } t<|v|. \end{cases}$$

Здесь $f_{нач}$, $f_{кон}$ - произвольные функции с неотрицательными действительными значениями; $r_н(v,s)$, $r_к(v,t)$, - произвольные функции с действительными значениями; $f_{внутр}$ – кусочно-линейная функция с неотрицательными значениями. При этом мы будем предполагать, что все весовые функции делеций равны 0 для делеции длины 0.

Отметим, что $f_{внутр}$ на некоторых участках линейности может принимать значение ∞, т.е. делеции определенных длин могут быть запрещены. Функции $r_н(v,s)$ $r_к(v,t)$ позволяют, в частности, учитывать зависимость вероятности разрыва полимерной цепи от контекста в точке разрыва.

2.1.2. Постановка задачи и обозначения.

Алгоритм, который будет описан в последующих разделах, строит оптимальное выравнивание двух данных символьных последовательностей относительно весовых функций делеции, удовлетворяющих определению

2.1.1; при этом допускается своя весовая функция для каждой из сравниваемых последовательностей. Весовая функция сопоставлений $\eta(v_1, v_2, i, j)$ может зависеть не только от сопоставляемых символов, но и от более широкого контекста сравниваемых последовательностей.

Входными данными алгоритма являются:

1) v_1, v_2 –выравниваемые последовательности, m_1, m_2 – их длины;

2) весовая функция сопоставлений η;

3) весовые функции удалений на краях последовательностей: $\xi^н_1$, $\xi^н_2$, $\xi^к_1$ $\xi^к_2$;

4) весовые функции разрывов $r^н_1, r^н_2, r^к_1, r^к_2$; ,

5) невозрастающие кусочно-линейные функции ξ_1 , ξ_2, определяющие веса удалений внутри последовательностей (см. п.2.3); эти функции задаются следующими величинами:

а) s_1, s_2 – количество изломов ξ_1 , ξ_2

б) $k^1_0, \ldots k^1_{s1}$, $k^2_0, \ldots k^2_{s2}$ – начала участков линейности функций ξ_1 , ξ_2, при этом: $0= k^1_0 < k^1_1 < k^1_{s1}$; $0= k^2_0 < k^2_1 < k^2_{s2}$;
мы для удобства считаем, что $k^1_{s1+1}=k^2_{s2+1}=+\infty$;

в) $\gamma^1_0, \delta^1_0, \ldots, \gamma^1_{s1}, \delta^1_{s1}, \gamma^2_0, \delta^2_0, \ldots, \gamma^2_{s2}, \delta^2_{s2}$ – коэффициенты линейности функций ξ_1 , ξ_2 на участках линейности, то есть такие числа, что при $k^i_j \le x < k^i_{j+1}$ выполнено:
$\xi_i(x) = \gamma^i_j \cdot x + \delta^i_j$.

При этом все функции $\xi^н_1$, $\xi^н_2$, $\xi^к_1$ $\xi^к_2$, ξ_1 , ξ_2, удовлетворяют определению 2.1.1.

Выходом алгоритма является некоторое оптимальное выравнивание последовательностей v_1 и v_2 относительно заданной системы весовых функций.

Ниже используются следующие обозначения.

Пары чисел обозначаются $\langle x, y\rangle$ или (x, y). Если $x = \langle x_1, x_2\rangle$, $y = \langle y_1, y_2\rangle$ - пары, то выражение $x > y$ означает, что $x_1 > y_1$ и , $x_2 > y_2$.

Склейка (*сопоставление позиций*) в словах v_1 и v_2 – это пара целых чисел $\tau = (\lambda_1, \lambda_2)$ таких, что $1 \le \lambda_1 \le |v_1|$; $1 \le \lambda_2 \le |v_2|$.

Пусть $G = \langle v_1, v_2, S \rangle$, где $S = \{\langle i_1, j_1\rangle, \ldots \langle i_n, j_n \rangle\}$ – выравнивание слов v_1 и v_2. *Верхней гранью* $Ub(G) = \langle Ub_1(G), Ub_2(G)\rangle$ выравнивания G

называется пара чисел $<i_n, j_n>$. Выравнивание $G_{огр} = < v_1[1, i_n], v_2[1, i_n] , S >$ будет называться *ограничением* выравнивания *G*. Если *(x,y)* - склейка и $(x,y)>Ub(\tau)$, то продолжением выравнивания *G* склейкой *(x,y)* будем называть выравнивание $G^*(x,y) = G = < v_1, v_2, S^*(x,y) >$;здесь $S^*(x,y)$ обозначает результат добавления склейки *(x,y)* в конец последовательности склеек *S*.

Вес выравнивания *G* при заданной системе весовых функций будет обозначаться *W(G)*. *Начальным весом P(G)* выравнивания *G* будем называть вес его ограничения: $P(G) = W(G_{огр})$.

Очевидно, для любого выравнивания τ выполнено:

$$W(\tau)=P(\tau)+ \xi^к_1(v_1, m_1-Ub_1(\tau))+ \xi^к_2(v_2, m_2-Ub_2(\tau)) \qquad (2.1.1)$$

Замечание. Требование того, что функции ξ_1, ξ_2 - невозрастающие, существенно только для использованного внутреннего представления текущего множества т.н. активных начальных выравниваний (см. п. 2.1.8).

2.1.3. Общее описание алгоритма выравнивания.

Определение 2.1.2. Начальной характеристикой $D_{a1,a2}$ склейки (a_1, a_2) или (a_1,a_2)-характеристикой называется максимальный начальный вес среди начальных весов всех выравниваний τ с верхней гранью (a_1,a_2), то есть

$$D_{a1,a2}=max\{P(\tau) \mid Ub(\tau)= (a_1,a_2)\}$$

Выравнивание τ называется *(a_1,a_2)-максимальным,* если $Ub(\tau)= (a_1,a_2)$ и начальный вес $P(\tau)= D_{a1,a2.}$ Выравнивание τ слов v_1 , v_2 называется *a_1-абсолютно максимальным,* если оно имеет максимальный полный вес среди выравниваний, удовлетворяющих условию $Ub_1(\tau)\le a_1$.

Алгоритм индукцией по a_1 (а при фиксированном a_1 – индукцией по a_2, $1\le a_1\le m_1$; $1\le a_2\le m_2$) для каждой пары (a_1,a_2) вычисляет начальную характеристику $D_{a1,a2}$ этой пары, а также некоторое (a_1,a_2) -максимальное выравнивание $\tau_{a1,a2.}$ Одновременно, используя соотношение (2.1.1), для каждого a_1 строится a_1-абсолютно максимальная связь $T_{a1} \in \{\tau_{x,y} | 1\le x\le a_1; 1\le y\le m_2\}$. Связь T_{m1} и будет искомой максимальной связью.

Вычисление величины $D_{a1,a2}$ и соответствующего (a_1,a_2) -максимального выравнивание $\tau_{a1,a2}$ основано на следующих соображениях.

Очевидно для произвольного $j \in \{1,...m_2\}$ существует единственное начальное выравнивание с верхней гранью $(1,j)$: $\tau_{i,j}=\{(1,j)\}$; вес этого выравнивания $P(\tau_{1,j})=D_{1,j}= \xi^н_i(j-1)+\eta(1,j)$. Аналогично, для произвольного $i \in$

{1,…m_1} существует единственное начальное выравнивание с верхней гранью *(i,1): $\tau_{i,j}$={(i,1)};* вес этой связи $P(_{ji,1})=D_{i,1}= \xi^н_i(i-1)+\eta(i,1)$.

Пусть (a_1,a_2) - произвольная склейка ($1< a_1 \le m_1;\ 1< a_2 \le m_2$) и построены выравнивания $\tau_{x,y}$, и характеристики $D_{x,y}$ для всех $x<a_1;\ y<a_2$. Вычисление начальной характеристики $D_{a1,a2}$ и (a_1,a_2) – максимальной связи $\tau_{a1,a2}$ сводится к вычислению непосредственного предшественника склейки (a_1,a_2) в связи $\tau_{a1,a2}$, т.е. такой склейки $(c_1.\ c_2)$, что $\tau_{a1,a2}= \tau_{c1,c2}*(a_1,a_2)$.

Очевидно,

$P(\tau_{a1,a2})=$

$= P(\tau_{c1,c2}) - \xi_1(a_1-c_1-1) - \xi_2(a_2-c_2-1) - r^н_1(c_1) - r^к_1(a_1) - r^н_2(c_2) - r^к_2(a_2) + \eta((a_1,a_2)$

Замечание. Эта формула несколько упрощена. Во-первых, не указана явно возможная зависимость функций η, $\xi^н_1$, $\xi^н_2$, $\xi^к_1$ $\xi^к_2$ от слов v_1, v_2. Во-вторых, если $a_1-c_1-1=0$ или $a_2-c_2=0$, то соответствующие штрафы за разрыв r должны полагаться равными 0. Чтобы не обременять изложение лишними деталями, мы и в дальнейшем не будем явно указывать зависимость функций η, $\xi^н_1$, $\xi^н_2$, $\xi^к_1$ $\xi^к_2$ от слов v_1, v_2. Кроме того, мы ограничимся разбором случая нулевых штрафов за разрывы. Общий случай не влияет на основные леммы 2.1.1 – 2.1.3. Конец замечания.

Введем разбиение множества всех возможных предшественников (z_1, z_2) склейки (a_1,a_2), то есть множество $\{1, \ldots, a_1-1\}\times\{1, \ldots, a_2-1\}$, на классы. А именно, пары $(x_1\ x_2)$ и (y_1, y_2) попадают в один класс тогда и только тогда, когда длины $a_i- x_i-1$, $a_i- y_i-1$ участков, удаляемых в слове v_i, попадают в один и тот же участок линейности функции ξ_1 ($i=1, 2$). Далее в каждом классе эквивалентности находим «кандидата в предшественники» - выравнивание, имеющее наибольший так называемый (a_1,a_2)–приведенный вес среди всех выравниваний данного класса (см. подробнее п.2.1.4). Искомое выравнивание $\tau_{a1,a2}$ будет иметь вид $\tau_{c1,c2}*(a_1,a_2)$, где (c_1,c_2) – одна из найденных склеек-кандидатов или склейка $(0, 0)$; последнее соответствует тому, что (a_1,a_2) – первая склейка выравнивания. Таким образом, выравнивание $\tau_{a1,a2}$ находится перебором из указанных вариантов. Ниже описаны рекуррентные соотношения, которые используются для эффективного вычисления выравниваний $\tau_{a1,a2}$ и их весов.

2.1.4. Классы выравниваний. Вычисление начальных характеристик.

Фиксируем склейку $a=(a_1,a_2)$, такую, что $a_1>1$, $a_2>1$. Пусть τ – выравнивание и $Ub(\tau)<a$.

Определение 2.1.3. Через $segm_1(\tau,a)$ обозначается номер такого участка линейности функции ξ_1, в который попадает длина удаляемого фрагмента $a_1-Ub_1(\tau)-1$. Иными словами, $segm_1(\tau,a)$ – это такое число α, что $k^1_\alpha \le a_1-Ub_1(\tau)-1<k^1_{\alpha+1}$. Величина $segm_2(\tau,a)$ определяется аналогично. Через $segm\ (\tau,a)$ обозначается пара ($segm_1(\tau,a)$, $segm_2(\tau,a)$). Через $A_i(x)$, где $i=1, 2,$ обозначается такое наибольшее t, что $x-2\ge k^i_t$.

Определение 2.1.4. Пусть $q_1\in\{0, \dots, s_1\}$, $q_2\in\{0, \dots, s_2\}$. Тогда (q_1, q_2)–классом выравниваний относительно склейки $a=(a_1, a_2)$ называется множество выравниваний

$$CC_a(q_1, q_2) = \{\tau \mid segm(\tau, a) = (q_1, q_2)\}$$

Замечание. Класс $CC_a(q_1, q_2)$ непуст тогда и только тогда, когда $\alpha_1 \le A_1(a_1)$ и $\alpha_2 \le A_2(a_2)$ (см. определение 2.1.3).

Говоря неформально, выравнивание $\tau \in CC_a(q_1, q_2)$, если длина $a_1 - Ub_1(\tau)-1$ фрагмента, удаляемого в слове v_1 в выравнивании $\tau*a$, попадает в q_1-й интервал линейности весовой функции ξ_1, а длина $a_2 - Ub_2(\tau)-1$ фрагмента, удаляемого в слове v_2 в выравнивании $\tau*a$, попадает в q_2-й интервал линейности весовой функции ξ_2.

Приведенным весом $P'(\tau,a)$ выравнивания τ относительно склейки a называется величина

$$P'(\tau,a)=P(\tau)+(\xi_1(a_1-Ub_1(\tau)-1)-\xi_1(k^1_{\alpha_1}))+(\xi_2(a_2-Ub_2(\tau)-1)-\xi_2(k^2_{\alpha_2}))\ ,$$

где $(\alpha_1,\alpha_2)=segm(\tau,a)$.

Очевидно,

$$P(\tau*a)=P'(\tau,a)+\xi_1(k^1_{\alpha_1})+\xi_2(k^2_{\alpha_2})+\eta(a_1,a_1)\ , \qquad (2.1.2)$$

а также

$$\xi_i(a_i-Ub_i(\tau)-1)-\xi_i(k^i_{\alpha i})=\gamma^i_{\alpha_i}\cdot(a_i-Ub_i(\tau)-1-k^i_{\alpha_i}) \qquad (2.1.3)$$

Пусть $(q_1, q_2)\in\{0, \dots, s_1\}\times\{0, \dots, s_2\}$. Положим (здесь и далее – максимум по пустому множеству равен $-\infty$):

$$m_a(q_1,q_2)=\max\{P'(\tau,a)\mid \tau\in CC_a(q_1,q_2)\}$$
$$M_a=\max_{(q_1,q_2)}\{m_a(q_1,q_2)+\xi_1(k^1_{q_1})+\xi_1(k^2_{q_2})\}$$
$$\overline{M}_a=\max\{M_a,\xi_1^{\mathrm{H}}(a_1-1)+\xi_2^{\mathrm{H}}(a_2-1)\}$$

Лемма 2.1.1. Пусть $a=(a_1,a_2)$ – склейка, a_1>1, a_2>1. Тогда

$$D_{a_1,a_2}=\overline{M}_a+\eta(a_1,a_2)\ .$$

Доказательство непосредственно следует из введенных определений.

Лемма 2.1.1 является основой нашего алгоритма: сначала находятся величины $m_a(q_1,q_2)$, затем перебором вычисляется величина M_a и, после этого, - величина $\overline{M}_a$. Осталось уточнить способ вычисления максимума приведенных весов $m_a(q_1,q_2)$ для всех (q_1, q_2)-классов начальных выравниваний CC_a. (q_1, q_2) относительно текущей склейки a.

2.1.5. Леммы о максимумах приведенных весов.

Введем ряд дополнительных обозначений.

Пусть $a = (a_1, a_2)$ – склейка; τ – выравнивание; $Ub(\tau) < a$ и $segm(\tau, a) = (q_1, q_2)$, т.е. $\tau\in CC_a(q_1, q_2)$.

Обозначим $\xi_i(a_i\text{-}Ub_i(\tau)\text{-}1)\text{-}\xi_i(k^i_{qi})$ через $Q_i(\tau, a_i)$, $i=1, 2$.

Тогда (2.1.2) и (2.1.3) можно переписать в виде ($i=1, 2$):

$$P'(\tau,a)=P(\tau)+Q_1(\tau,a_1)+Q_2(\tau,a_2)$$

и

$$\mathrm{Q_i}(\tau,a_i)=\gamma^{\mathrm{i}}_{q_{\mathrm{i}}}\cdot(a_i-Ub_i(\tau)-1-k^i_{q_i})$$

Определение 2.1.5. Пусть τ – начальное выравнивание, $i\in\{1, \ldots m_1\}$ и $Ub(\tau) <i$. *Полуприведенным весом* $P''(\tau,i)$ связи τ относительно $i\in\{1, \ldots m_1\}$ называется число

$$P''(\tau,i)=P(\tau)+Q_1(\tau,i)\ .$$

Через $r^i(q,t)$, где $i\in\{1, \ldots m_1\}$, $0\le q\le A_1(i), t\in\{1,\ldots,m_2\}$ обозначается величина

$$r^i(q,t)=\max_{\tau}\{P''(\tau,i)\mid segm_1(\tau,i)=q\ \&\ Ub_2(\tau)=t\}$$

Лемма 2.1.2. Пусть $\Delta_q^1 = \{k_q^1, \ldots, k_{q+1}^1-1\}$; $\Delta_q^2 = \{k_q^2, \ldots, k_{q+1}^2-1\}$. Для произвольных $q, q_1 \in \{0,\ldots,A_1(i)\}$, $j \in \{1,\ldots,m_2\}$, $q_2 \in \{1,\ldots,A_2(j)\}$ выполнено

$$r^i(q,j) = \max_{d: i-d-1 \in \Delta_q^1} \{\gamma_q^1 \cdot (i-d-1-k_q^1) + D_{d,j}\}, \qquad (2.1.4)$$

$$m_{i,j}(q_1,q_2) = \max_{t: j-t-1 \in \Delta_{q_2}^2} \{\gamma_{q_2}^2 \cdot (j-t-1-k_{q_2}^2) + r^i(q_1,t)\}. \qquad (2.1.5)$$

Доказательство.

1) Доказательство соотношения (2.1.4)

Обозначим через U множество всех таких целых d, что $k^1_q \le i-d-1 < k^1_{q+1}$. Имеем:

$$r^i(q,t) = \max_{\tau}\{P''(\tau,i) | segm_1(\tau,j) = \alpha \& Ub_2(\tau) = t\} =$$

$$= \max_{d \in U}\{\max_{\tau}\{P(\tau) + \gamma^1_q \cdot (i-d-1-k^1_q) | Ub(\tau) = (d,t)\}\} =$$

$$= \max_{d \in U}\{\gamma^1_q \cdot (i-d-1-k^1_q) + \max_{\tau}\{P(\tau) | Ub(\tau) = (d,t)\}\} =$$

$$= \max_{d \in U}\{\gamma^1_q \cdot (i-d-1-k^1_q) + D_{d,t}\}.$$

2) Доказательство соотношения (2.1.5).

Пусть V –множество всех целых t, что $k^2_{q2} \le j-t-1 < k^2_{q2+1}$. Очевидно,

$$m_{i,j}(q_1, q_2) = \max\{P'(\tau, i, j) | \tau \in CC_{i,j}(q_1, q_2)\} =$$

$$= \max_{t \in V} \max_{\tau} \{P'(\tau, i, j) | segm_1(\tau,i) = \alpha_1 \& Ub_2(\tau) = t\}$$

Далее (при фиксированном t)

$$\max_{\tau} \{P'(\tau, i, j) | segm_1(\tau,i) = q_1 \& Ub_2(\tau) = t\} =$$

$$= \max_{\tau} \{P''(\tau,i) + Q_2(\tau, j) | segm_1(\tau,i) = q_1 \& Ub_2(\tau) = t\} =$$

$$= \max_{\tau} \{P''(\tau,i) + \gamma^2_{q2} \cdot (j-t-1-k^2_{q2}) | segm_1(\tau,i) = q_1 \& Ub_2(\tau) = t\} =$$

$$= \gamma^2_{q2} \cdot (j-t-1-k^2_{q2}) + \max_{\tau}\{P''(\tau,i) | segm_1(\tau,i) = q_1 \& Ub_2(\tau) = t\} =$$

$$= \gamma^2_{q2} \cdot (j-t-1-k^2_{q2}) + r^i(\alpha_1, t).$$

Лемма доказана

Следствие.

$$r^i(q, j) = \gamma^1_q (i-k^1_q) + \max\{D_{d,j} - \gamma^1_q \bullet (d-1)\}$$

$d: i-d-1 \in \Delta^1_q$

$$m_{i,j}(q_1, q_2) = \gamma^2_{q2}(j-k^2_{q2}) + \max_{t: j-t-1 \in \Delta^1_q} \{ r^i(q_1, t) - \gamma^2_{q2} \bullet (t-1)\}$$

Доказательство – очевидно.

Имея в виду приведенное выше следствие, определим вспомогательные величины $Dp_{d,j}(i)$ и $rp^i(q, t, j)$.

Определение 2.1.6. Пусть $i \in \{1, \ldots m_1\}$, $j \in \{1, \ldots m_2\}$; $d < i$, $t < j$ и $q \in \{0, \ldots, A_1(i)\}$. Положим

$Dp_{d,j}(i) = D_{d,j} - \gamma_{h1} \bullet (d-1)$, где $h_1 = segm_1(i-d-1)$

$rp^i(q, t, j) = rp^i(q, t) - \gamma_{h2} \bullet (t-1)$, где $h_2 = segm_2(j-t-1)$

Замечание. Зависимость $Dp_{d,j}(i)$ от i и зависимость $rp^i(q, t, j)$ от j проявляется только в выборе значений соответственно h_1 и h_2.

Лемма 2.1.3 является переформулировкой следствия к лемме 2.1.2 в новых обозначениях.

Лемма 2.1.3. Пусть $\Delta^1_q = \{k^1_q, \ldots, k^1_{q+1}-1\}$; $\Delta^2_q = \{k^2_q, \ldots, k^2_{q+1}-1\}$. Тогда для произвольных $q, q_1 \in \{0,\ldots, A_1(i)\}$, $j \in \{1,\ldots, m_2\}$, $q_2 \in \{1,\ldots, A_2(j)\}$ выполнено

$$r^i(q, j) = \gamma^1_q \cdot (i - k^1_q) + \max_{d: i-d-1 \in \Delta^1_q} \{Dp_{d,j}(i)\}, \tag{2.1.6}$$

$$m_{i,j}(q_1, q_2) = \gamma^2_{q_2} \cdot (j - k^2_{q_2}) + \max_{t: j-t-1 \in \Delta^2_{q2}} \{rp^i(\alpha_1, t, j)\}. \tag{2.1.7}$$

Доказательство непосредственно следует из леммы 2.1.2

Обозначим:

$\arg\_ r^i(q,t)$ - значение d, при котором достигается максимум в (2.1.4);

$\arg\_ m_{i,j}(q_1, q_2)$ - значение t, при котором достигается максимум в (2.1.5);

В обоих случаях, если максимум достигается в нескольких точках, то берем наибольшую из них.

Лемма 2.1.4. Фиксируем $i \in \{2,\ldots, m_1\}$; $j \in \{1,\ldots, m_2\}$. Пусть далее $q_1 \in \{0,\ldots, A_1(i)\}$; $q_2 \in \{1,\ldots, A_2(j)\}$; $t \in \{1,\ldots, m_2\}$. Тогда

(i) Если

$$\arg\_ r^i(q_1, t) > i - k^1_{q1+1} \tag{2.1.8}$$

или

$$D_{i-k^1_{q_1},t} \geq r^i(q_1, t) + \gamma^1_{q_1}, \tag{2.1.9}$$

то

$$\text{à)} \quad r^{i+1}(q_1,t) = \max\{D_{i-k^1_{q_1},t}, r^i(q_1,t) + \gamma^1_{q_1}\} \tag{2.1.10}$$

$$\text{á)} \quad \arg\_ r^i(q_1,t) = \begin{cases} i-k^1_{q_1}, \text{åñëè } D_{i-k^1_{q_1}} \geq r^i(q_1,t)+\gamma^2_{q_2} \\ \arg\_ r^i(q_1,t) - \text{â ïðîòèâíîì} \qquad \text{ñëó÷àå} \end{cases}$$

(*ii*) Если

$$\arg\_ m_{i,j}(q_1,q_2) > j - k^2_{q_2+1} \tag{2.1.11}$$

или

$$r^i(q_1, j-k^2_{q_2}) \geq m_{i,j}(q_1,q_2) + \gamma^2_{q_2} \tag{2.1.12}$$

то

$$\text{à)} \quad m_{i,j+1}(q_1,q_2) = \max\{r^i(q_1, j-k^2_{q_2}), m_{i,j}(q_1,q_2) + \gamma^2_{q_2}\} \tag{2.1.13}$$

$$\text{á)} \quad \arg\_ m_{i,j+1}(\alpha_1,\alpha_2) = \begin{cases} j-k^2_{\alpha_2}, \text{åñëè } r^i(\alpha_1, j-k^2_{\alpha_2}) \geq m_{i,j}(\alpha_1,\alpha_2)+\gamma^2_{\alpha_2} \\ \arg\_ mb_{i,j}(\alpha_1,\alpha_2) - \text{â ïðîòèâíîì} \qquad \text{ñëó÷àå} \end{cases}$$

Доказательство.

Мы приводим только доказательство утверждения (i); утверждение (ii) доказывается аналогично. Обозначим через *R* величину:

$R=max\{P(\tau)+ \gamma^1_{q1}\cdot(i- Ub_1(\tau)- 1- k^1_{q1}) | i- k^1_{q1+1} < Ub_1(\tau) \leq i-1- k^1_{q1} \;\&\; Ub_2(\tau)=t\}$

Имеем:

$r^{i+1}(q_1,t)=\max\limits_{\tau}\{P''(\tau, i+1) | segm_1(\tau,i+1)= q_1 \;\&\; Ub_2(\tau)=t\}=$

$= \max\limits_{\tau}\{P(\tau)+ \gamma^1_{q1}\cdot(i+1-Ub_1(\tau)-1- k^1_{q1}) \mid$

$(i+1- k^1_{q1+1}-1 < Ub_1(\tau) - \leq i+1- k^1_{q1+1}-1 \& Ub_2(\tau)=t$

$\} =$

$=max\{ \gamma^1_{q1}+R,$

$max \{P(\tau) | Ub_1(\tau) =i- k^1_{q1} \;\&\; Ub_2(\tau)=t\},$

$\}=$

$= max\{ \gamma^1_{q1}+R, D_{i-k^1_{\alpha 1}}\}.$ (2.1.14)

Очевидно $R \leq r^i(\alpha_1,t)$, причем, если выполнено (2.1.8), то $R=r^i(\alpha_1,t)$.

Поэтому, из (2.1.14) следует импликация (2.1.8)=>(2.1.10) в лемме 2.1.4.

Если выполнено (2.1.8), то имеем:

$D_{i\text{-}k1\alpha,t} \geq r^i(\alpha_1,t) + \gamma^1_{\alpha 1} \geq R + \gamma^1_{\alpha 1}$

и из (2.1.14) получаем:

$r^{i+1}(\alpha_1,t) = max\{ D_{i\text{-}k1\alpha,t}, \gamma^1_{\alpha 1}, + r^i(\alpha_1,t)\} = D_{i\text{-}k1\alpha,t}$,

что и требовалось доказать.

Утверждение (i) леммы 2.1.4 доказано. Утверждение (ii) леммы 2.1.4 доказывается аналогично.

Следствие.

(*i*) Если $A_1(i) = s_1$, то

$$r^{i+1}(s_1,t) = \max\{D_{i-k^1_{s_1},j}, r^i(s_1,t) + \gamma_{s1}^1\}$$

(*ii*) Если $A_2(j) = s_2$ и $q \in \{0, 1, \ldots, s_1\}$, то

$$m_{i,j+1}(q\ , s_2) = \max\{r^i(q\ , j - k^2_{s_2}), m_{i,j}(q\ , s_2) + \gamma^2_{s_2}\}\ .$$

Доказательство очевидно, так как при $q = s_1$ всегда выполнено (2.1.8), а при $q = s_2$ всегда выполнено (2.1.11).

```
procedure
        пролог;
        for i from 2 to m1 do begin
                этап (i);
        end
        эпилог;
end
```

Рис.2.1.1.

2.1.6. Общая структура алгоритма. Данные.

На рис 2.1.1 представлена общая структура алгоритма, который строит оптимальное выравнивание относительно заданной кусочно-линейной системы весовых функций делеций. Основная подпрограмма *этап* представлена на рис. 2.1.2. В качестве псевдокода здесь и далее использован алголоподобный язык, описанный в [24].

```
procedure этап(цел i)
        D_{i,1} = ξ^{a_t}_2(i-1)+η(1, i);
        for j from 2 to m2 do begin
                for q1 from 0 to A1(i) do begin
                        вычислить r^i(q1, j)
```

```
                    for q2 from 0 to A2(j) do begin
                    вычислить m_ij(q1, q2)
                    end
                end
                вычислить D_i,j
            end
        end
```

Рис.2.1.2. Алгоритм этап. Одновременно с вычислением весов $r^i(s1, j)$, $m_{ij}(s1, s2)$, $D_{i,j}$ вычисляются соответствующие выравнивания $arg\_r^i(s1, j)$, $arg\_m_{ij}(s1, s2)$ и $\tau_{i,j}$.

Пусть *GLOB(i)* обозначает состояние глобальных данных алгоритма после выполнения вызова этап(*i*) (при $i = 2, \ldots, m_1$) или после вызова подпрограммы пролог (при $i = 1$).

Данные *GLOB(i)* включают:

1) величины $D_{x,y}$ для всех

$$(x, y) \in \{i - k_{s_1}^1, \ldots, i\} \times \{1, \ldots, m_2\};$$

2) величины $r^i(q_1, j)$ и $\arg\_r^i(q_1, j)$ - для всех $j \in \{1, \ldots, m_2\}$ и для всех таких q_1, что эти величины определены;

3) выравнивания $\tau_{d,j}$, где $d = max(1, i - k_{s_1}^1), \ldots, i$; а также, если $i - k_{s_1}^1 - 1 \geq 1$, выравнивания $\tau_{f,j}$, где $f = \arg\_r^i(s_1, j)$ ($j \in \{1, \ldots, m_2\}$);

4) *i*-максимальное выравнивание T_i и его вес $Score(T_i)$;

5) специальные структуры, предназначенные для вычисления $r^{i+1}(q, j)$ (см. п. 2.1.7).

Величины $\{D_{x,y}\}$ хранятся в двумерном массиве *ALIGN_SCORE*, состоящем из $k_{s_1}^1 + 1$ строк длины m_2; значение $D_{x,y}$ хранится в ячейке $ALIGN\_SCORE[x \bmod (k_{s_1}^1 + 1), y]$. Этот же массив используется для хранения других сведений о выравнивании $\tau_{x,y}$ (см. п. 2.1.8). Величины $r^i(q_1, j)$ хранятся в двумерном массиве *R_SCORE*, состоящем из $s_1 + 1$ строк длины m_2; значение $r^i(q_1, j)$ хранится в ячейке $R\_SCORE[(q_1, j)]$. Для вычисления

$r^i(q_1, j)$ используется рабочий массив *R_SCORE_TEMP*, имеющий те же структуру и размер, что и массив *R_SCORE*.

Специальные структуры, предназначенные для вычисления $r^{i+1}(s, j)$, описаны в п.2.1.7. Множество выравниваний, перечисленных выше в пп. 2-4 будем обозначать *ALIGNMENT(i);* представление этого множества описано в п.2.1.8. Отметим, что мы вместе с весом $D_{x,y}$ храним ссылку на выравнивание $\tau_{x,y}$, а вместе с весом $r^i(s_1, j)$ - ссылку на выравнивание $\tau_{f,,j}$, где $f = \arg\_ r^i(s_1, j)$.

Таким образом, алгоритм *пролог* строит *GLOB(1),* то есть вычисляет выравнивания $\tau_{1,j}$, их веса $D_{1,j}$, где $j \in \{1,...,m_2\}$ и выравнивание T_1; при вызове алгоритма *этап* (*i*) по *GLOB(i-1)* строит *GLOB(i)* (и для этого вычисляет выравнивания $\tau_{i,j}$ и веса $D_{i,j}$, где $j = 1,...,m_2$), алгоритм *эпилог* по $GLOB(m_1)$ строит искомое выравнивание T_{m_1}.

В соответствии с п. 2.1.3, алгоритмы *пролог* и *эпилог* могут быть выполнены за время $O(m_2)$.

Выполнение вызова алгоритма *этап*(*i)* рассмотрено в следующих разделах. Вычислению весов $r^i(q_1, j)$, $m_{i\,j}(q_1, q_2)$, $D_{i,j}$ (здесь $j = 1, ...,m_2$; $q_1 = 0, ..., s_1$; $q_2 = 0, ..., s_2$) посвящен раздел 2.1.7, поддержка представления множества выравниваний *ALIGNMENT(i)* описана в разделе 2.1.8.

2.1.7. <u>Вычисление максимумов приведенных весов. Сложность алгоритма.</u>

Вычисление значений $m_{i,j}(q_1, q_2)$ и $r^i(q_1, j)$ основывается на леммах 2.1.3 и 2.1.4. При известных значениях $m_{i,j}(q_1, q_2)$ значение $D_{i,\ j}$ может быть вычислено за время $O(q_1 \bullet q_2)$ на основании леммы 2.1.1.

Рассмотрим сначала вычисление $r^i(q_1, j)$. Значения $r^i(s_1, j)$ вычисляются на основании следствия к лемме 2.1.4; каждое такое значение вычисляется за время $O(1)$. В случае $q_1 < s_1$ значение $r^i(q_1, j)$ вычисляется на основании леммы 2.1.3, см. формулу (2.1.6).

Пусть $i \in \{1, ..., m_1\}$. Для каждого $j \in \{1, ..., m_2\}$ и для каждого $q \in \{0, ..., s_1\text{-}1\}$ мы храним множество $U(i, j, q)$ значений величин

$$U(i, j, q) = \{ Dp_{d,j}(i) | i\text{-}1\text{-}k^1_{q+1} < d \le i\text{-}1\text{-}k^1_q \}.$$

Над множеством *U(i, j, s)* требуется выполнять операции (i) взятия максимума и (ii) обновления при переходе от *i* к *i+1*. В последнем случае нужно только исключить элемент, соответствующий значению $d= i- k^1_{q+1}$ и добавить элемент, соответствующий значению $d= i- k^1_q$. Действительно, (см. замечание) после определения величины $Dp_{d, j}(i))$, если $segm_1(\tau_{d, j}, i) = segm_1(\tau_{d, j}, i+1)$, то $Dp_{d, j}(i) = Dp_{d, j}(i+1)$. Поэтому при переходе от *GLOB(i)* к *GLOB(i+1)* перевычисление значений $Dp_{d, j}(i)$ не требуется.

При подходящем выборе представления множества *U(i, j, s)* эти операции могут быть выполнены за время $O(log|U(i, j, q)|) \le O(log\Delta_1)$, где $\Delta_1 = max\{ k^1_{h+1} - k^1_h \mid h = 0, ..., s_1-1 \}$ [24]. Таким образом, вычисление всех значений $r^i(q_1, j)$, где $q_1 = 0, ..., s_1 ; j = 1, ..., m_2$, может быть выполнено за время $O(m_2 \cdot s_1 \cdot log\Delta_1)$ с использованием памяти $O(m_2 \cdot k^1_{s_1})$.

На практике более эффективным оказывается алгоритм, основанный на использовании леммы 2.1.4 для всех значений q_1 и отказе от поддержания специальных структур для представления множеств *U(i, j, q)*. В худшем случае мы получим для времени оценку $O(m_2 \cdot s_1 \cdot \Delta_1)$, однако, как показывают компьютерные эксперименты, в среднем такой подход дает время $O(m_2 \cdot s_1)$.

Величины $m_{i,j}(q_1, q_2)$ вычисляются по найденным значениям $r^i(q_1, t)$ подобно тому, как значения $r^i(q_1, t)$ вычисляются по значениям $D_{x,y}$. При этом аналогами величин $Dp_{d, j}(i)$ являются величины $rp^i(q, t, j)$, см. определение 2.1.6. В ходе выполнения вызова *эman(i)* для заданного *j* значения $m_{i,j}(q_1, q_2)$ могут быть найдены за время $O(s_1 \cdot s_2 \cdot log\Delta_2)$, где $\Delta_2 = max\{ k^2_{h+1} - k^2_h \mid h = 0, ..., s_2-1 \}$. С практической точки зрения, как и в случае вычисления $r^i(q_1, t)$, можно считать, что верна оценка $O(s_1 \cdot s_2)$. Это дает оценку

$$O(m_2 \cdot s_1 \cdot s_2 \cdot log\Delta_2)$$

для общего времени вычисления значений $m_{i,j}(q_1, q_2)$ при выполнении вызова *эman(i)*.

Как мы увидим ниже, учет построения соответствующих выравниваний не влияет на общую оценку времени выполнения вызова *эman(i)*. Таким образом, для выполнения вызова *эman(i)* верна оценка времени $O(m_2 \cdot (s_1 \cdot log\Delta_1 + s_1 \cdot s_2 \cdot log\Delta_2))$. Что касается используемой для хранения выравниваний памяти, то оценка в худшем случае для нее составляет $O((m_2)^2)$. Однако, в

практически интересных случаях (см. п. 2.1.8) эта память оказывается существенно меньшей и растет как $O(m_2)$. Таким образом, для потребной алгоритму памяти получаем оценку $O((m_2)^2$ в худшем и $O(m_2)$ в практически интересных случаях.

2.1.8. Представление и вычисление выравниваний.

В простейшем варианте алгоритма после выполнения вызова *эtап(i)* мы помним все выравнивания из множества *ALIGNMENT(i)*. В более сложном варианте мы храним только входящие в *ALIGNMENT(i)* т.н. *i*-активные выравнивания, т.е. такие выравнивания, продолжение которых может оказаться искомым оптимальным выравниванием исходных последовательностей v_1 и v_2..

Здесь мы ограничимся рассмотрением простейший вариант. Более сложный вариант имеет существенные преимущества с точки зрения программной реализации (см. [11]), однако приводит к тем же оценкам сложности, что и вариант, описанный ниже.

Пусть S - некоторое множество начальных выравниваний; для удобства далее будем считать, что первой склейкой любого выравнивания является склейка (0, 0), сопоставляющая начальные маркеры сравниваемых слов.

Определение 2.1.8. Пусть S – множество начальных выравниваний. Через $G(S)$ обозначается такой ориентированный помеченный граф, что

(*i*) множество вершин $G(S)$ – это множество склеек, входящих в выравнивания из S;

(*ii*) из вершины (x_1, y_1) ведет ребро в вершину(x_2, y_2) тогда и только тогда, когда $(x_1, y_1) < (x_2, y_2)$ и эти склейки – соседние склейки в некотором выравнивании $\tau \in S$;

(*iii*) каждой вершине $(x, y) \in G(S)$ приписано булево значение ACTIVE: если (x, y) – последняя склейка некоторой связи $\tau \in S$ (такую вершину будем называть *S-активной*), то ACTIVE (x,y)=TRUE, иначе ACTIVE (x,y)=FALSE.

Очевидно, при любом множестве S, граф $G(S)$ – связный граф без циклов, причем вершина (0, 0) – единственная вершина, в которую не входит ни одно ребро и все листья – активные вершины. Если же S – множество выравниваний вида $\tau_{x,y}$ (в частности, множество *ALIGNMENT(i)* или его подмножество), то в

любую вершину, кроме (0, 0), входит ровно одно ребро, то есть $G(S)$ – корневое помеченное ориентированное дерево.

Определение 2.1.9. Вершина в дереве $G(S)$ называется *опорной*, если эта вершина является корнем или листом, или активной вершиной, или из этой вершины выходит более одного ребра. *Простым* путем назовем путь, лежащий между двумя опорными вершинами и не содержащий других опорных вершин. Очевидно, простой путь $(x_1, y_1),..., (x_s, y_s)$ задает выравнивание $\tau = \{(x_1, y_1),...,(x_s, y_s)\}$ между исходными словами и, следовательно, может быть описан концевыми склейками (x_1, y_1), (x_s, y_s) и множеством «дырок» - то есть фрагментов, удаляемых внутри слов v_1, v_2 при выравнивании $< v_1[x_1, x_2]\ v_2[y_1, y_2],\ \tau >$.

Дерево *G(S)* однозначно определяется множеством так называемых *описателей* опорных вершин. Для каждой опорной вершины *V* описатель задает:

1) координаты вершины *V*;

2) координаты «наследников» *V*, то есть опорных вершин, в которые из *V* ведут простые пути (более точно – указатель на начало списка наследников данной опорной вершины, см. ниже);

3) координаты «предшественника» *V*, то есть опорной вершины, из которой в *V* ведут простые пути;

4) указатель на список дырок в простом пути, оканчивающемся в *V*.

Множество выравниваний *ALIGNMENT(i)* описывается деревом G_i .= *G(ALIGNMENT(i)),* которое, в свою очередь, представляется списком описателей опорных вершин. Говоря более точно, представление дерева G_i включает следующие объекты:

1) список описателей опорных вершин;

2) список описателей наследников опорных вершин;

3) списки описателей дырок в простых путях;

Структура описателя опорной вершины описана выше. Описатель наследника опорной вершины – это тройка <*NEXT, LINK* >, где *NEXT* - ссылка на описатель наследника; *LINK* – указатель на описатель следующего наследника той же вершины. Описатель дырки – это четверка <*BEG, END,*

SEQ, LINK>, где *BEG, END* – координаты начала и конца удаляемого фрагмента; *SEQ* – номер последовательности, в которой удаляется фрагмент (1 или 2); *LINK* – указатель на описатель следующей дырки в том же простом пути.

Во время выполнения вызова *эman(i)* над описанным внутренним представлением выполняются операции:

- добавление выравниваний вида $\tau_{i,j}$, где $j = 1, \ldots, m_2$;

- удаление выравниваний вида $\tau_{f,j}$, где $f = i - k^1_{s_1} - 1$ и $f \neq \arg\_ r^i(s_1, j)$.

При включении в *ALIGNMENT(i)* нового выравнивания создается новый описатель опорной вершины, устанавливаются необходимые ссылки и, возможно, добавляется описатель нового удаленного фрагмента. Таким образом, добавление нового выравнивания выполняется за фиксированное время.

Удаление выравнивания, вообще говоря, может потребовать $O(m_2)$ удалений описателей опорных вершин и связанных с ним элементов других перечисленных выше списков, при этом время удаления выравнивания пропорционально количеству удаленных описателей удаленных вершин. Но каждый такой описатель опорной вершины во время работы алгоритма (а не только данного этапа) удаляется только один раз. Поэтому общее время выполнения операций, связанных с поддержкой хранения выравниваний равно $O(m_1 \bullet m_2)$.

Оценим память, необходимую для хранения выравниваний.

Лемма 2.1.5. Дерево $G_i = G(ALIGNMENT(i))$ содержит $O(m_2)$ листьев и $O(m_2)$ опорных вершин.

Доказательство. Количество элементов в множестве *ALIGNMENT(i)* не превосходит $k^1_{s_1} \bullet m_2 = O(m_2)$, т.е. в дереве G_i есть $O(m_2)$ листьев и $O(m_2)$ вершин (некоторые из них – листья), которые являются последними склейками выравниваний из *ALIGNMENT(i)*. Далее, из каждой опорной вершины, не являющейся последней вершиной выравнивания из *ALIGNMENT(i),* выходит не менее двух ребер. Поэтому количество таких вершин пропорционально количеству листьев в дереве *G(ALIGNMENT(i)),* т.е. тоже равно $O(m_2)$. Лемма доказана.

Из леммы следует, что память, необходимая для хранения описателей опорных вершин и их наследников равна $O(m_2)$. Количество удаленных фрагментов в худшем случае равно $O((m_2)^2)$. Однако в практически интересных случаях суммарное количество удаленных фрагментов значительно меньше.

Особенность нашей реализации описанного алгоритма состоит в том, что для хранения дальних склеек выделяется *фиксированная* память $S = c_0 \cdot m_2^*$, где m_2^* - наибольшая (в этой реализации) допустимая длина слова υ_2; c_0 - константа. Когда эта память заполняется, ее содержимое копируется во внешнюю память ЭВМ (на диск) и *больше не модифицируется*. Поэтому мы иногда храним на диске «лишние» описатели вершин.

2.2. Глобальное выравнивание при штрафах за делеции, пропорциональных длине делеции.

2.2.1.Постановка задачи

Фиксируем некоторый алфавит и слова v_1 и v_2 длины m_1 и m_2 соответственно. Для удобства будем считать, что у этих слов есть одинаковый начальный маркер – нулевая буква. Пусть даны неотрицательные числа *c, f* и *d.* Весом выравнивания $G = < v_1, v_2 , S>$ в этом разделе будем называть величину

$$W(G) = c \bullet p - f \bullet r - d \bullet (m_1-p-r) - d \bullet (m_2-p-r) \qquad (2.2.1)$$

Здесь p –количество совпадений; r - количество несовпадений; m_1-p-r - количество символов, удаленных в первой последовательности; m_2-p-r - количество символов, удаленных во второй последовательности.

В разделе 2.2. приведен адаптивный алгоритм выравнивания последовательностей относительно определения веса выравнивания (2.2.1). Частным случаем этого определения (при условии $f \geq 2d$) является хорошо изученная задача о построении наибольшей общей подпоследовательности

двух символьных последовательностей [148]. Поэтому ниже мы будем считать, что

$$f<2d.$$

Неформально говоря, определение (2.2.1) означает, что сопоставление любых двух символов приносит бонус *c*; за сопоставление различных символов дается штраф *f*, а за удаление любого символа – штраф *d*. Такая задача соответствует грубому предварительному анализу последовательностей; ее исследование также представляет интерес с теоретической точки зрения. Учитывая, что нас интересует само оптимальное выравнивание, а не его вес, значения коэффициентов можно нормировать. Поэтому мы будем считать, что в формуле (2.2.1) выполнено:

$$c = 1.$$

Ниже мы продолжаем использовать понятия (склейки, начальные выравнивания и пр.), введенные в разделе 2.1.

2.2.2. Общее описание алгоритма.

2.2.2.1. Опорные множества. Достаточное условие оптимальности.

Лемма 2.2.2. Пусть w_1 и w_2 - слова w_1 и w_2 длины l_1 и l_2 соответственно и $l_1 \geq l_2$. Тогда

(i) вес любого выравнивания слов w_1 и w_2 не превосходит $c\bullet l_1-d\bullet(l_2- l_1)$

(ii) существует выравнивание слов w_1 и w_2, имеющее вес не менее $-f\bullet l_1-d\bullet(l_2- l_1)$

Доказательство очевидно.

Определение 2.2.1. *Специальным весом* (x_1, x_2)-выравнивания G слов v_1 и v_2 длины m_1 и m_2 называется величина

$$SP(G) = P(G)+ c\bullet min(m_1-x_1, m_2-x_2) - d\bullet|(m_1-x_1)-(m_2-x_2) |$$

Лемма 2.2.1. Пусть *B, B'* – начальные выравнивания и *B* - префикс *B'*, Тогда

$$SP(B)\geq SP(B');$$

Доказательство несложно и здесь не приводится.

Следствие. Пусть *G* - выравнивание слов v_1 и v_2 и - *B* - префикс выравнивания *G* . Тогда

$$P(G)\leq SP(B)$$

Определение 2.2.2. Множество S начальных выравниваний слов v_1 и v_2 называется *опорным* множеством, если в S найдется выравнивание, которое является префиксом некоторого оптимального выравнивания слов v_1 и v_2.

Выравнивание G слов v_1 и v_2 называется *граничным,* если $Ub_1(G) = |v_1|$ или $Ub_2(G) = |v_2|$.

Лемма 2.2.3. Пусть S – опорное множество для слов v_1 и v_2 и начальное выравнивание $B \in S$. Пусть B – граничное выравнивание и в S нет выравнивания, которое имеет специальный вес больше, чем $SP(B)$. Тогда выравнивание B' слов v_1 и v_2, имеющее тот же набор сопоставлений, что и выравнивание B – оптимальное выравнивание слов v_1 и v_2.

Доказательство. Очевидно,

$$P(B') = P(B) - d\bullet(|v_1| - Ub_1(B)) - d\bullet(|v_2| - Ub_2(B)) = SP(B)$$

По определению 2.2.2, существует оптимальное выравнивание G слов v_1 и v_2 такое, что некоторый префикс A выравнивания G лежит в S. По лемме 2.2.2 и по условию доказываемой леммы имеем:

$$P(G) \le SP(A) \le SP(B)=P(B'),$$

т.е. B' – тоже оптимальное выравнивание. Лемма доказана.

Определение 2.2.3. Пусть S – опорное множество. *Максимальный* элемент S – это его элемент, имеющий максимально возможный начальный вес.

По лемме 2.2.3, для построения оптимального выравнивания достаточно построить опорное множество, в котором есть максимальный элемент, который является граничным выравниванием. Такое опорное множество будем называть *терминальным.*

2.2.2.2. Каноническая последовательность опорных множеств.

Построение терминального опорного множества и граничного максимального выравнивания в нем может быть выполнено с помощью описанной ниже процедуры построения т.н. *канонической последовательности* опорных множеств $\{S_i\}$. Множество S_0 состоит из одного выравнивания $<v_1, v_2, \{(0,0)\}>$. Пусть S_i построено. Если S_i – терминальное, то процесс построения канонической последовательности закончен. В противном случае мы выбираем некоторое начальное выравнивание $L_i \in S_i$ (*лидер* множества S_i) и заменяем его таким набором продолжений D_i, что любое

оптимальное выравнивание, продолжающее выравнивание L_i, является продолжением одного из выравниваний множества D_i. Это гарантирует то, что полученное множество будет опорным. Далее из построенного множества, возможно, удаляются некоторые избыточные выравнивания. Таким образом,

$S_{i+1} \subseteq (S_i - \{ L_i \}) \cup D_i$

Для завершения описания алгоритма остается уточнить процедуры:

1) выбора лидера L_i;
2) построения замещающего множества D_i;
3) выделения и удаления избыточных выравниваний.

Центральная идея алгоритма в том, чтобы в качестве лидера выбирать максимальное выравнивание. Это позволяет избежать рассмотрения «неперспективных» начальных выравниваний. В качестве замещающих множеств будут использованы т.н. доминаторы, описанные в следующем подразделе. Работа с избыточными выравниваниями описана в п.2.2.4.

2.2.3. Построение замещающего множества D_i.

Определение 2.2.4. Пусть A – выравнивание слов v_1 и v_2 длины соответственно m_1 и m_2; $Ub(A) = (x_1, x_2)$. *Доминатор* выравнивания A – это такой набор D расширений выравнивания A, что множество $\{C \mid A*C \in D\}$ является опорным множеством для выравниваний слов $v_1[x_1+1, \_m_1]$ и $v_2[x_2+1, \_m_2]$. Доминатор D называется *нетривиальным*, если $A \notin D$ и не один элемент D не является продолжением другого.

Лемма 2.2.4. Пусть S – опорное множество; $A \in S$ и D – нетривиальный доминатор выравнивания A. Тогда $S-\{A\} \cup D$ – опорное множество.

Доказательство. Если существует оптимальное выравнивание слов v_1 и v_2, которое продолжает отличное от A начальное выравнивание $C \in S$, то утверждение очевидно. В противном случае существует оптимальное выравнивание $B = A*B'$ слов v_1 и v_2, продолжающее выравнивание A. Очевидно, B'- оптимальное выравнивание слов $v_1[x_1+1, \_m_1]$ и $v_2[x_2+1, \_m_2]$. По определениям 2.2.2 и 2.2.4, существует оптимальное выравнивание C' слов $v_1[x_1+1, \_m_1]$ и $v_2[x_2+1, \_m_2]$, такое, что $A*C'$ продолжает некоторое выравнивание из D. Т.к. $P(A*C') = P(A)+P(C') = P(A)+P(B') = P(A*B') = P(B)$ и B - оптимальное выравнивание B' слов v_1 и v_2, то $A*C'$ – искомое

оптимальное выравнивание слов v_1 и v_2, продолжающее выравнивание из S-$\{A\} \cup D$. Лемма доказана.

Введем дополнительные обозначения.

$\mathrm{Next}_1(v_1, v_2, x_1, x_2) = min\{j \mid j > x_2 \,\&\, v_1[x_1+1] = v_2[j]\}$

$\mathrm{Next}_2(v_1, v_2, x_1, x_2) = min\{j \mid j > x_1 \,\&\, v_1[j] = v_2[x_2+1]\}$

$C(x_1, x_2) = (x_1+1, x_2+1)$

$H((x_1, x_2) = (x_1+1, \mathrm{Next}_1(v_1, v_2, x_1, x_2))$

$V((x_1, x_2) = (\mathrm{Next}_2(v_1, v_2, x_1, x_2), x_2+1)$

Говоря неформально, $H((x_1, x_2)$ и $V((x_1, x_2)$ – ближайшие (по строке и по столбцу) к (x_1, x_2) точки совпадений. Очевидно, если $v_1[x_1+1] = v_2[x_2+1]$, то $C(x_1, x_2) = H((x_1, x_2) = V((x_1, x_2)$.

Обозначение. Пусть A – не граничное начальное выравнивание слов v_1 и v_2, $Ub(A) = (x_1, x_2)$. Через $D(A)$ обозначим набор множество $\{A*C(x_1, x_2), A*H(x_1, x_2), A*V(x_1, x_2)\}$. Если одна из величин $H(x_1, x_2), V(x_1, x_2)$ не определена, то соответствующее выравнивание не включается в $D(A)$.

Лемма 2.2.5. Пусть A – не граничное начальное выравнивание слов v_1 и v_2. и для весовых коэффициентов выполнено $f < 2d$ (см. п.2.2.1). Тогда $D(A)$ является доминатором выравнивания A.

Доказательство. Учитывая определение 2.2.4, лемму достаточно доказать для случая, когда A содержит единственную склейку $(0,0)$. Рассмотрим такое оптимальное выравнивание $B = <v_1, v_2, S>$ слов v_1 и v_2, где m_i – длина слова v_i $(i=1, 2)$; $S = \{x_o=(0,0), x_1, \ldots, x_k\}$; $x_i=(a_1, b_i)$, $i=0,\ldots, k$, что склейка x_1 – минимально возможная. Т.е. не существует оптимального выравнивания $C = <v_1, v_2, T>$ такого, что $T = \{y_o=(0,0), y_1=(r_1,s_1), \ldots, y_n\}$ и $r_1 \le a_1$; $s_1 \le b_1$ и хотя бы одно из этих неравенств – строгое.

Докажем, что $x_1 \in \{C(0,0), H(0,0), V(0,0)\}$, откуда непосредственно следует утверждение леммы. Действительно, либо $a_1 = 1$, либо $b_1 = 1$ (в противном случае, т.к. $f < 2d$, можно добавить в B склейку $(1, 1)$ и увеличить вес выравнивания). Пусть, например, $a_1 = 1$ (случай $b_1 = 1$ рассматривается аналогично). Если $b_1 = 1$, то $x_1 = C(0,0)$ и утверждение доказано. Если $b_1 > 1$, то $v_1[1] \neq v_2[1]$ (в противном случае, не изменяя веса, можно в выравнивании B заменить склейку $x_1=(a_1,b_1)$ меньшей склейкой $(1, 1)$). По аналогичным

соображениям, если $v_1[1] = v_2[1]$, то $b_1 = \text{Next}_1(v_1, v_2, 0, 0)$, т.е. $x_1 = H((x_1, x_2)$. Доказательство завершено.

2.2.4. Избыточные элементы опорных множеств.

Будем называть элемент B опорного множества S *избыточным,* если множество S-$\{B\}$ тоже является опорным. Удаление избыточных элементов опорных множеств основано на понятии мажорирования.

Определение 2.2.5. Начальное выравнивание A слов v_1 и v_2 *мажорирует (*соответственно, *строго мажорирует)* начальное выравнивание B слов v_1 и v_2, если для любого выравнивания B' слов v_1 и v_2 , которое является продолжением выравнивания B, существует выравнивание A' слов v_1 и v_2 такое что A' является продолжением выравнивания A и $P(A) > P(B)$ *(*соответственно, $P(A) \geq P(B)$ *)*

Лемма 2.2.6. Пусть S – опорное множество и $B \in S$. Если существует (1) начальное выравнивание $A \in S$, которое мажорирует B или (2) произвольное (не обязательно лежащее в S) выравнивание C, которое строго мажорирует B, то выравнивание $B \in S$ является избыточным.

Доказательство – очевидно.

Леммы 2.2.7, 2.2.8, 2.2.9 дают достаточные условия мажорирования. Во всех леммах, заменив «больше или равно» на «больше» мы получим условия строгого мажорирования.

Лемма 2.2.7. Пусть A и B – начальные выравнивания слов v_1 и v_2; m_1 и m_2 - длины этих слов. Положим

$$T(A) = f\bullet \min\{ m_1 - Ub_1(A) , m_2 - Ub_2(A) \} + d\bullet \mid (m_1 - Ub_1(A)) - (m_2 - Ub_2(A)) \mid$$

и пусть

$$P(A) - T(A) \geq SP(B)$$

Тогда A мажорирует B.

Доказательство непосредственно следует из леммы 2.2.1.

Лемма 2.2.8. Пусть A и B –начальные выравнивания слов v_1 и v_2; $Ub(A) = (x_1 , x_2)$, $Ub(B) = (y_1 , y_2)$ и $x_1 \leq y_1$; $x_2 \leq y_2$. Пусть для некоторого выравнивания C фрагментов $v_1[x_1 +1, y_1]$ и $v_1[x_1 +1, y_1]$ выполнено:

$$P(A*C) \geq P(B)$$

Тогда А мажорирует В.

Доказательство очевидно.

Следствие. Если $x_1 = y_1$ или $x_2 = y_2$ и $P(A) - d\bullet(y_1 - x_1) - d\bullet(y_2 - x_2) \geq P(B)$, то A мажорирует B.

Лемма 2.2.9 дает достаточные условия для элиминирования элементов доминатора выравнивания, описанного в лемме 2.2.5.

Лемма 2.2.9.

1. Пусть A и B – выравнивания такие, что $Ub_1(B) = Ub_1(A)$, $Ub_2(B) < Ub_2(A)$ и и $P(A) > P(B) - d\bullet(Ub_2(A) - Ub_2(B))$. Пусть далее, B' – такое продолжение B, что $Ub_1(B') = Ub_1(B)+1$ и $Ub_2(B') > Ub_2(A)$. Тогда A строго мажорирует B'.

2. Пусть A и B – выравнивания такие, что $Ub_2(B) = Ub_2(A)$, $Ub_1(B) < Ub_1(A)$ и и $P(A) > P(B) - d\bullet(Ub_1(A) - Ub_1(B))$. Пусть далее, B' – такое продолжение B, что $Ub_1(B') = Ub_1(B)+1$ и $Ub_2(B') > Ub_2(A)$. Тогда A строго мажорирует B'.

Замечание. Условия $P(A) > P(B) - d\bullet(Ub_2(A) - Ub_2(B))$ и $P(A) > P(B) - d\bullet(Ub_1(A) - Ub_1(B))$ аналогичны условиям леммы 2.2.8.

Доказательство. Здесь приведено доказательство для п.1. Доказательство для п.2 аналогично. Пусть s – вес склейки $v_1[Ub_1(B')] = v_2[Ub_2(B')]$. Имеем: $P(A*Ub(B')) = P(A) - d\bullet(Ub_2(B')- Ub_2(A)-1 >$

. $> P(B) - d\bullet(Ub_2(A) - Ub_2(B)) - d\bullet(Ub_2(B')- Ub_2(A)) -1 =$

$= P(B) - d\bullet(Ub_2(B')- Ub_2(B))-1=P(B')$

Отсюда непосредственно следует утверждение леммы.

Определение 2.2.6. Пусть A – начальное выравнивание слов v_1 и v_2,; m_1 и m_2 - длины слов v_1 и v_2. *Диагональ* выравнивания A - это число $d(A)=(m_1-Ub_1(A))–(m_2-Ub_1(A))$. Будем говорить, что выравнивание A заканчивается *выше главной диагонали*, если $d(A) \geq 0$ и *выше главной диагонали,* если $d(A) \leq 0$.

Лемма 2.2.10. Пусть A и B – начальные выравнивания слов v_1 и v_2, длины m_1 и m_2 соответственно, причем A и B заканчиваются по одну сторону от главной диагонали. Пусть далее

$|d(A)| \geq |d(B|)$;

$SP(A) \geq SP(B)$

$min(m_1 – Ub_1(A), m_2 – Ub_2(A)) \leq min(m_1 – Ub_1(B), m_2 – Ub_2(B))$

Тогда A мажорирует B.

Доказательство. Пусть $Ub(A) =(a_1, a_2)$, $Ub(B) =(b_1, b_2)$, см. рис. 2.2.1. Пусть, для определенности, выравнивания A и B заканчиваются ниже главной диагонали, т.е. . $min(m_1 – Ub_1(A), m_2 – Ub_2(A)) = m_1 – Ub_1(A)$; $min(m_1 – Ub_1(B), m_2 – Ub_2(B)) = m_1 – Ub_1(B)$ и, следовательно, $m_1 – Ub_1(A) < m_1 – Ub_1(B)$. Соответственно, условие $|d(A)| \geq |d(B|)$ означает, что

$$(a_1 - a_2) >(b_1 - b_2) \qquad (2.2.2)$$

Пусть C - произвольное выравнивание слов v_1 и v_2, которое является продолжением выравнивания A. Докажем, что у начального выравнивания A есть продолжение, чей вес не менее веса выравнивания C. Пусть $y =(y_1, y_2)$ – первая склейка выравнивания C такая, что $z > Ub(A)$ и s – вес сопоставления. Если такой склейки нет, возьмем в качестве z фиктивную склейку $(m_1 +1, m_2 +1)$ и положим $s =0$. Пусть $A' = A*z$; C' – это начальное выравнивание, которое получается из выравнивания C ограничением на префиксы $v_1[1, y_1]$ и $v_2[1, y_2]$. Для доказательства леммы достаточно показать, что $P(A') \geq P(C')$. Пусть r – количество склеек выравнивания C между концом начального выравнивания B и склейкой y. Очевидно, (см. (2.2.2),

$$r \leq max(a_1 - b_1, a_2 - b_2) = a_1 - b_1 \qquad (2.2.3)$$

Имеем:

$P(A') = P(A)+s - d\bullet(y_1- a_1-1) - d\bullet(y_2- a_2-1)$;

$P(C') \leq P(B) +s + c\bullet r - d\bullet(y_1- b_1-1-r) - d\bullet(y_2- b_2-1-r)$;

$P(A') – P(C') \geq P(A) – P(B) - c\bullet r + d\bullet(a_1- b_1-r) + d\bullet(a_2- b_2-r)$

(2.2.4)

Ввиду (2.2.3),

$$c\bullet r \geq a_1- b_1 \qquad (2.2.5)$$

$$a_1- b_1-r \geq 0 \qquad (2.2.6)$$

$$a_2- b_2-r \geq a_2- b_2- a_1- b_1 =(a_1 - a_2) - (b_1 - b_2) \qquad (2.2.7)$$

Из *(2.2.4), (2.2.5), (2.2.6)* и *(2.2.7)* получаем:

$$P(A') – P(C') \geq P(A) – P(B) - c\bullet(a_1 - b_1) - d\bullet((a_1 - a_2) - (b_1 - b_2)) \qquad (2.2.8)$$

С другой стороны, по условию леммы, $SP(A) \geq SP(B)$. Из этого имеем:

$SP(A) = P(A)+c\bullet(m_1 - a_1) - d\bullet((m_2 - a_2) - (m_1 - a_1))=$

$= P(A) - c\bullet a_1 - d\bullet(a_1 - a_2) + c\bullet m_1 - d\bullet m_2) + d\bullet m_1$

$SP(B) = P(B)+c\bullet(m_1 - b_1) - d\bullet((m_2 - b_2) - (m_1 - b_1)) =$

$= P(B) - c \bullet b_1 - d \bullet (b_1 - b_2) + c \bullet m_1 - d \bullet m_2) + d \bullet m_1$

$SP(A) – SP(B) = P(A) – P(B) - c \bullet (a_1 - b_1) - d \bullet ((a_1 - a_2) - (b_1 - b_2))$

Таким образом,

$P(A) – P(B) - c \bullet (a_1 - b_1) - d \bullet ((a_1 - a_2) - (b_1 - b_2)) \geq 0$

и, ввиду *(2.2.8), $P(A') – P(C') \geq 0$,* что и требовалось доказать.

Определение 2.2.7. Пусть дана каноническая последовательность опорных множеств $E = \{S_0, S_1, ..., S_n\}$. *Ретро-множество RET(E)* – это множество всех максимальных элементов опорных множеств S_i, $i = 1, ..., n$.

Лемма 2.2.11. Пусть $E = \{S_0, S_1, ..., S_n\}$ – каноническая последовательность опорных множеств для v_1 и v_2 такая, что для каждого $i = 1, ..., n$ лидер L_i – максимальный элемент в соответствующем опорном множестве S_i. Пусть $B \in S_n$ – такое начальное выравнивание, что для некоторого $A \in RET(E)$, которое заканчивается по ту же сторону от главной диагонали и не является префиксом B выполнено:

$|d(A)| \geq |d(B|)$;

$SP(A) \geq SP(B)$

$min(m_1 – Ub_1(A), m_2 – Ub_2(A)) \leq min(m_1 – Ub_1(B), m_2 – Ub_2(B))$

Тогда B избыточное выравнивание в S_n.

Доказательство. По лемме 2.2.10, начальное выравнивание A мажорирует начальное выравнивание B. Если $A \in S_n$, то лемма доказана. Пусть $A \in S_i$, где $i < n$ и A F - оптимальное выравнивания слов v_1 и v_2 , которое является продолжением B. Для доказательства достаточно доказать, что найдется оптимальное выравнивание слов v_1 и v_2, которое является продолжением некоторого начального из S_n-$\{B\}$. По определению 2.2.5, существует являющееся продолжением A выравнивание C слов v_1 и v_2 , которое имеет вес не менее $P(F)$ и, следовательно, оптимальное. По определению канонической последовательности, оптимальное выравнивание C является продолжением некоторого начального выравнивания $D \in S_n$, такого, что A – префикс D. По условию, A является префиксом B. Поэтому выравнивания D и B различны и, следовательно, выравнивание D – искомое. Лемма доказана.

Лемма 2.2.12. Пусть $d(A) = d(B)$, $SP(A) \geq SP(B)$ и $Ub_1(A) \geq Ub_1(B)$.

Доказательство аналогично леммам 2.2.8 и 2.2.10.

2.2.5. Описание алгоритма.

2.2.5.1.Структуры данных.

Введем дополнительные обозначения.

Определение 2.2.8.

Пусть S - опорное множество. Через $SP(S)$ будем обозначать величину $SP(S) = max\{SP(A)|A \in S\}$

Пусть $E = \{S_0, ..., S_n\}$- каноническая последовательность опорных множеств. Положим $DRET(i) = min\{ min\{m_1\text{-}Ub_1(A), m_2\text{-}Ub_2(A)\} \mid A \in RET(E) \ \& \ d(A) = i\}$. Алгоритм хранит таблицу значения $DRET(i)$. Таблица инициализируется значениями $+\infty$ и изменяется, когда (1) порождается новое начальное выравнивание с начальным весом $SP(S)$, где S – текущее опорное множество или (2) при переходе к новому опорному множеству значение $SP(S)$ уменьшилось.

Пусть A – начальное выравнивание слов v_1 и v_2 длин соответственно m_1 и m_2. Через $TP(A)$ обозначим величину

$TP(A) = P(A) - f\bullet \min\{ m_1 - Ub_1(A) , m_2 - Ub_2(A) \} + d\bullet \mid (m_1 - Ub_1(A)) - (m_2 - Ub_2(A)) \mid$

Очевидно, существует выравнивание G слов v_1 и v_2, которое продолжает A и имеет вес не менее $TP(A)$.

Глобальные объекты нашего алгоритма включают:

1) текущее опорное множество S, включая значение $SP(S)$;

2) значение $TPMAX$ – максимум значений $TP(A)$ по всем начальным выравниваниям A, порожденным во время работы алгоритма;

3) таблицу значений $DRET(i)$, $i = -m_2, ..., m_1$.

Опорное множество S представлено двумя множествами:

1) множеством $ALIGN$ описателей всех выравниваний $A \in S$;

2) множеством $MATCH$ всех склеек, принадлежащих выравниваниям $A \in S$.

Каждый элемент – это запись, имеющая следующие поля:

- $SP(A)$;
- $Ub(A)$;
- ссылка на элемент $Ub(A)$ в $MATCH$.

Отметим, что, по построению, в S нет двух таких выравниваний A и B, что $Ub(A) = Ub(B)$, поэтому описатель однозначно задает элемент опорного множества S.

Элементы *ALIGN* упорядочены по убыванию $SP(A)$, а при данном $SP(A)$ – по возрастанию $|d(A)|$. Кроме того, на *ALIGN* определена система двусвязных списков, облегчающих поиск избыточных выравниваний (см. леммы 2.2.8, 2.2.9 и 2.2.12). Каждое выравнивание $A \in S$ принадлежит трем спискам:

1) «горизонтальному» списку, включающему все такие выравнивания $B \in S$, что $Ub_1(B) = Ub_1(A)$; элементы списка упорядочены по возрастанию $Ub_2(B)$;

2) «вертикальному» списку, включающему все такие выравнивания $B \in S$, что $Ub_2(B) = Ub_2(A)$; элементы списка упорядочены по возрастанию $Ub_1(B)$;

3) «диагональному» списку, включающему все такие выравнивания $B \in S$, что $Ub_2(B) - Ub_1(B) = Ub_2(A) - Ub_1(A)$; элементы списка упорядочены по возрастанию $Ub_1(B)$.

По построению опорного множества, если склейка z принадлежит двум выравниваниям $A, B \in S$, то склейки, предшествующие склейке z в выравниваниях A и B совпадают. Поэтому множество *MATCH* представляет собой дерево. Для каждой склейки $z \in MATCH$ мы по техническим причинам помним количество ее наследников в дереве *MATCH*, а также признак – существует ли такое выравнивание $A \in S$, что $z = Ub(A)$.

2.2.5.2. Построение нового опорного множества.

Пусть $S = S_i$ – текущее опорное множество. В качестве лидера множества S мы выбираем такое S-максимальное начальное выравнивание A, для которого величина $|d(A)|$ минимальна. По построению, (см. ниже) в S не может быть более одного максимального выравнивания A с данным диагональным числом $d(A)$. Если в S есть два максимальных выравнивания с данным значением $|d(A)|$, то в качестве лидера берем то, у которого $D(A)>0$. Из п.2.2.5.1 следует, что выбор лидера может быть выполнен за фиксированное время.

Если $SP(S_i) \neq SP(S_{i-1})$, то до выбора лидера $L(S)$ следует модифицировать таблицу $DRET(i)$ и выполнить удаление лишних по лемме 2..2.11.

Построение доминатора $D(A)$ для лидера $A = L(S)$ описано в п. 2.2.3. При этом проверяется, не мажорируются ли выравнивания из $D(A)$ другими элементами опорного множества S по лемме 2.2.8, а также не мажорируются ли выравнивания $A*H(Ub(A))$ и $A*V(Ub(A))$ другими элементами опорного множества S по лемме 2.2.9. В последнем случае избыточные выравнивания не включаются в новое опорное множество. После того, как новые выравнивания включены в опорное множество проверяется, не мажорируют ли они некоторые «старые» элементы опорного множества по леммам 2.2.8 и 2.2.12. При необходимости обнаруженные избыточные выравнивания удаляются из опорного множества.

Отметим, что продолжение $B \in D(A)$ лидера A может иметь специальный вес $SP(B)=SP(A)$ только если $v_1[Ub_1(B)] = v_2[Ub_2(B)]$ и при этом (1) $Ub_1(B) = Ub_1(A)+1$; $Ub_2(B) = Ub_2(A)+1$ и, следовательно, $d(B)=d(A)$; или *(2)* склейка $|d(B)| < |d(A)|$ и выравнивания A, B лежат по одну сторону от главной диагонали. В обоих случаях, если такое выравнивание не элиминируется по лемме 2.2.11, оно немедленно становится лидером.

2.2.6. Сложность алгоритма.

Из п.2.2.5 тривиально следует квадратичная оценка потребных времени и памяти в худшем случае. Однако, а отличие от известных алгоритмов построения оптимального выравнивания (см. обзор в [312]) и алгоритма п. 2.1, представленный в настоящем разделе алгоритм является адаптивным. Т.е. время его работы и потребная память в «простых» случаях, существенно меньше, чем в худших. Ниже мы уточним это утверждение.

Лемма 2.2.12. Пусть S – опорное множество, принадлежащее канонической последовательности опорных множеств, построенных по алгоритму п.2.2.5. Тогда в S нет выравниваний, которые могут быть удалены по критериям лемм 2.2.9, 2.2.11 и 2.2.12.

Доказательство – очевидно.

Следствие 1. В S нет двух выравниваний A и B таких, что $Ub(A) = Ub(B)$.

Следствие 2. Пусть $A, B \in S$, причем $d(A) = d(B)$ и $Ub(A) < Ub(B)$. Тогда $SP(A) > SP(B)$.

Следствие 3. Каждый из списков, описанных в п. 2.2.5.1, содержит не более $max(m_1, m_2)$ элементов.

Следствие 4. Количество максимальных элементов в опорном множестве S не превышает $m_1 + m_2$. (это следует из Следствия 2).

Пусть $\{S_0, S_1, \ldots, S_K\}$ – каноническая последовательность опорных множеств, причем множество S_K – терминальное; N_i - количество максимальных элементов в S_i; $m = max(m_1, m_2)$. Т.к. на каждом шаге к опорному множеству добавляется не более трех выравниваний и не менее одного выравнивания удаляется, то для нужной памяти получаем оценку $c \bullet K$. Компьютерные эксперименты показывают, что в среднем $K = O(m)$, что дает для памяти оценку $O(m)$.

Рассмотрим оценки для временной сложности. Построение доминатора лидера текущего опорного множества занимает фиксированное время (см. п.2.2.3). Удаление каждого избыточного выравнивания и включение нового выравнивания в множество *ALIGN* занимает время $\sim log(m)$. Таким образом, обработка лидера очередного опорного множества S_i, включая (1, если нужно) модификацию таблицы *DRET, (2)* построение доминатора, (3) проверку избыточности каждого доминатора по условиям лемм, (4) включение в структуры множеств *ALIGN* и *MATCH* занимает время $O(log(m))$. Поэтому общее время на обработку лидеров равно $O(Klog(m))$. Далее, удаление одного элемента текущего опорного множества из структур *ALIGN* может быть выполнено за постоянное время. Поэтому общее на обнаружение «старых» избыточных элементов и их удаление равно $O(K)$, а общее время работы алгоритма равно $O(K)log(m))$.

Оценим величину K. Пусть для слов v_1 и v_2 длин соответственно m_1 и m_2 существует оптимальное выравнивание B, содержащее t несовпадений и s удаленных символов. Тогда количество p совпадений в выравнивании B равно $(m_1 + m_2 - 2t-2s)/2 = (m_1 + m_2)/2 - t-s$. Обозначим $(m_1 + m_2)/2$ через Q, а $|m_1 - m_2|/2$ через R. Тогда несложные вычисления приводят к следующим формулам:

$$min(m_1, m_2) = Q-R; \quad max(m_1, m_2) = Q+R;$$

$$P(B) = SP(B) = Q - (f+1) \bullet t - (d+1) \bullet s$$

Следовательно,

$$SP(S_K) = Q - (f+1)\bullet t - (d+1)\bullet s.$$

При этом

$$SP(S_0) = Q-R - 2fR = Q- (2f+1)R$$

Поэтому среди весов $SP(S_i)$ есть

$$SP(S_0) - SP(S_K) = R+ (f+1)\bullet(t-2R) + (d+1)s \qquad (2.2.9)$$

По построению канонической последовательности (см. п.2.2.5 и лемму 2.2.12), для произвольного веса P среди множеств S_i есть не более $O(m_1+m_2)$ таких, у которых $SP(S_i)= P$. Вместе с (2.2.9.) это дает оценку

$$K = O((m_1)^2- (m_2)^2) + O((m_1+m_2) \bullet(t+s) \qquad (2.2.10)$$

Сходную оценку можно получить из несколько иных соображений.

Пусть далее, для определенности, $m_1 \geq m_2$.

Лемма 2.2.13. Рассмотрим выравнивание A такое, что для $d(A) = (m_1 - Ub_1(A)) - (m_2 - Ub_1(A))$ выполнено:

$$d(A) > m_1 - m_2 \text{ или } d(A) < 0 \qquad (2.2.11)$$

Тогда $SP(A) \leq min(m_1, m_2) - d\bullet(m_1 - m_2) - (d+1) \bullet d^*(A)$,

где

$d^*(A) = d(A)- (m_1 - m_2)$, если $d(A) > m_1 - m_2$

$d^*(A) = |d(A)|$, если $d(A) < 0$

$d^*(A) = 0$ в остальных случаях

Доказательство – очевидно.

Следствие 1. Пусть $A = L(S_i)$ для некоторого i. Тогда

$$d^*(A) \leq 2t +s + 1.5\bullet(m_1 - m_2)$$

Доказательство.

Имеем:

$SP(A) \geq SP(B);$

$min(m_1, m_2) - d\bullet(m_1 - m_2) - (d+1) \bullet d^*(A) \geq Q - (f+1)\bullet t - (d+1)\bullet s$

$Q-R - d\bullet(m_1 - m_2) - (d+1) \bullet d^*(A) \geq Q - (f+1)\bullet t - (d+1)\bullet s$

$(d+1) \bullet d^*(A) \leq (f+1)\bullet t + (d+1)\bullet s + d\bullet(m_1 - m_2)+ R$

$d^*(A) \leq ((f+1)/ (d+1)) \bullet t + \bullet s +(d/(d+1)) \bullet(m_1 - m_2)+ (m_1 - m_2)/2$

Отсюда, поскольку $f < 2d$, получаем:

$$d^*(A) \leq 2t +s + 1.5\bullet(m_1 - m_2) \qquad (2.2.12)$$

Следствие 2.

$$K \leq (2t + s + 2.5 \bullet |m_1 - m_2|) \bullet \min(m_1, m_2)$$

Доказательство. Следует из (1) того, что все лидеры опорных множеств S_i имеют разные верхние грани и (2) подсчета количества таких склеек z, что выравнивание A с $Ub(A)=z$ удовлетворяет условию *(2.2.12)*. Искомая оценка для величины K, а вместе с ней и оценки сложности алгоритма доказаны.

2.3. Векторные веса сопоставления символов и Парето-оптимальные выравнивания

2.3.1. Общее описание подхода.

В биоинформатике задача выравнивания последовательностей традиционно ставится, как задача построения оптимального выравнивания относительно некоторой весовой функции. При этом выбор значения параметров этой функции, особенно, относящиеся к весам делеций, недостаточно обоснован с точки зрения биологических приложений. Например, недостаточно обосновано то, что штрафы за делеции описываются линейными функциями.

В этом разделе описан многокритериальный подход к задаче выравнивания, позволяющий преодолеть указанный недостаток. При таком подходе весом выравнивания является не число, а k-мерный числовой вектор, обычно k равно 2 или 3. Вместо алгоритма, который находит максимально возможный вес выравнивания и имеющее этот вес оптимальное выравнивание, будет предложен алгоритм, которой строит Парето-оптимальное множество [247] весов и соответствующие этим весам «Парето-оптимальные» выравнивания. Говоря неформально, выравнивание является Парето-оптимальным, если оно является оптимальным при некоторой монотонной (не обязательно линейной!) весовой функции от элементарных весовых параметров – компонентов векторного веса. В качестве таких параметров могут выступать, например, количество совпадений, количество удаленных фрагментов, их суммарная длина и т.п.

Этот подход перекликается с «параметрическими выравниваниями, предложенными в работе М.Уотермана и соавт. [321] и позднее развитого в [141]. Отправной точкой для обоих подходов является то, что традиционный

вес выравнивания является линейной комбинацией «элементарных» весовых функций – суммарного веса сопоставлений, количества удаленных фрагментов и их суммарной длины. При этом в цитированных работах изучается пространство весовых коэффициентов. В то же время, при предлагаемом подходе объектом анализа являются вектора, образованные значениями элементарных весовых функций («векторные веса»), а весовые коэффициенты не используются вовсе. Таким образом, можно сказать, что параметрический подход [321] и представленный ниже многокритериальный подход являются двойственными.

2.3.2. Определения.

Определение 2.3.1. Пусть $k \geq 2$ – целое число. *Векторная весовая функция* – это функция, сопоставляющая каждому выравниванию A k –мерный вектор $V(A)$, называемый (векторным) весом выравнивания A.

Пример. Положим $k = 2$. Рассмотрим векторный вес

$V(A) = (NumMatch(A), -NumDel(A))$,

где $NumMatch(A)$ и $NumDel(A)$ – это соответственно число совпадений и число удаленных символов в выравнивании A. Здесь элементарными весовыми функциями являются $NumMatch(A)$ и $-NumDel(A)$.

Перечислим еще несколько элементарных весовых функций, которые (наряду с другими) могут использоваться при определении векторных весов.

$NumGap(A)$ – количество удаленных сегментов последовательностей («делеций», “gaps”);

$WeightMatch(A)$ – суммарный вес сопоставлений символов относительно выбранной матрицы замен;

$MisMatch(A)$ – количество несовпадений (соответствует использованию единичной матрицы замен).

Определение 2.3.2. Вектор V_1 *мажорирует* вектор V_2 если каждая компонента V_1 больше или равна соответствующей компоненты V_2 и хотя бы в одном случае неравенство является строгим. Вектора V_1 и V_2 называются *несравнимыми*, если ни $V2$ не мажорирует V_1, ни V_1 не мажорирует V_2 .

Определение 2.3.3.[247] Пусть M – множество k-мерных векторов. Вектор $\mathbf{v} \in M$ называется Парето-оптимальным в M, если ни один вектор $\mathbf{u} \in M$ не мажорирует $\mathbf{v}$. Парето-подмножество в M – это множество всех

Парето-оптимальных векторов в M. Множество называется Парето-оптимальным, если оно совпадает со своим Парето-подмножеством.

Определение 2.3.4. Пусть S_1 и S_2 – последовательности; V – векторная весовая функция. Выравнивание A последовательностей S_1 и S_2 называется Парето-оптимальным относительно весовой функции V, если $V(A)$ является Парето-оптимальным вектором в множестве векторных весов всех возможных выравниваний последовательностей S_1 и S_2.

Пример. Пусть S_1 = ATGACG, S_2 = AAGTAGC. Положим

$$V(A) = (NumMatch(A), –NumDel(A).$$

На рис.2.3.1 изображены 6 выравниваний последовательностей S_1 и S_2. Выравнивания (1), (2), (4), (5) – Парето оптимальные относительно весовой функции V. Векторные веса выравниваний (1) и (2) равны (4, –3), векторные веса выравниваний (4) и (5) равны (3, –1). Векторный вес выравнивания (1) мажорирует вес выравнивания (3) и несравним с весом выравнивания (4). Множество Парето-оптимальных весов включает два вектора: (4, –3) and (3, –1).

```
(1)       (2)       (3)       (4)       (5)       (6)
ATG_ACG_  ATG_A_CG  A__T_GACG ATGACG_    ATG_ACG   _ATGACG
| | | |   | | | |   |  | | |  | |  |    | | | |    |
AAGTA_GC  AAGTAGC_  AAGTAG_C_ AAGTAGC     AAGTAGC   AAGTAGC
```

Рис. 2.3.1. Различные выравнивания последовательностей S_1 = ATGACG и S_2 = AAGTAGC.

Лемма 2.3.1. Пусть $V(A) = \{x_1(A), ..., x_n(A)\}$ – векторный вес выравнивания A и $W(A)$ – скалярный вес вида

$$W(A) = W(x_1(A), ..., x_n(A)),$$

где $W(x_1, ..., x_n)$ монотонно возрастает по всем своим аргументам.

Пусть далее A – оптимальное выравнивание последовательностей S_1 и S_2 относительно весовой функции $W(A)$. Тогда A – Парето-оптимальное выравнивание относительно векторной весовой функции $V(A)$.

Доказательство несложно и здесь не приводится.

Лемма 2.3.2. Существуют последовательности U_1, U_2 и выравнивание A этих последовательностей такое, что

i) выравнивание A оптимально относительно векторной весовой функции $V(A) = (NumMatch(A), –NumDel(A))$;

(ii) выравнивание A не является оптимальным относительно скалярной весовой фуцкции $W(A) = k_1 \bullet NumMatch(A) – k_2 \bullet NumDel(A)$ при любом выборе коэффициентов k_1 и k_2.

Доказательство. Рассмотрим, например, последовательности U_1 = XXAAGT and U_2 = AGTYYY (см. Рис. 2.3.2).

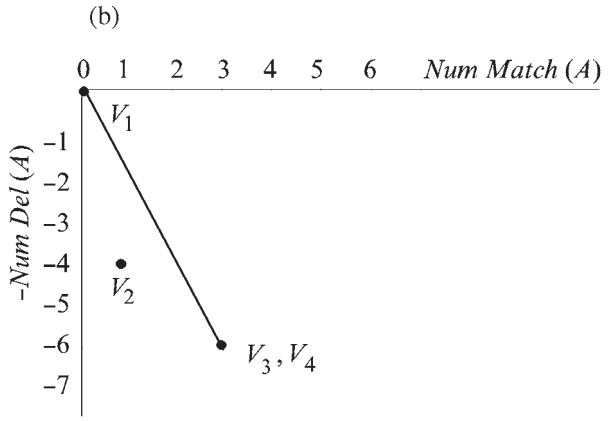


Рис.2.3.2. Графическое изображение множества Парето-оптимальных весов последовательностей XXAAGT и AGTYY.

Используя технику из раздела 2.2, легко показать, что множество Парето-оптимальных весов для U_1 и U_2 состоит из трех векторов $V_1 = (0, 0)$; $V_2 = (1, –4)$; $V_3 = (3, –6)$. Точке V_1 соответствует тривиальное безделеционное выравнивание, обозначим его AL_1.

Точке V_2 соответствует выравнивание AL_2:

AL_2:

```
XXAAGT--
--AGTYYY
```

Точке V_3 соответствуют два выравнивания:

AL_3

```
XXAAGT---
---AGTYYY
```

AL_4

```
XXAAGT---
--A-GTYYY
```

Точка V_2 лежит ниже прямой, проходящей через точки V_1 and V_3. Поэтому соответствующее весу V_2 выравнивание AL_2 является искомым. Лемма доказана.

В рассмотренном примере (см. рис.2.3.2) выравнивания AL_3 and AL_4, имеют одинаковый векторный вес V_3. Для этих выравниваний выполнено:

$$NumMatch(AL_3) = NumMatch(AL_4) = 3;$$
$$NumDel(AL_3) = NumDel(AL_4) = 6.$$

В то же время у этих выравниваний различно количество удаленных сегментов: $NumGap(AL_3) = 2$; $NumGap(AL_4) = 3$. Поэтому для весовой функции

$$VG(A) = (NumMatch(A), -NumDel(A), -NumGap(A)),$$

выравнивание AL_4 не является Парето-оптимальным.

2.3.2. Базовый алгоритм.

В этом и следующих разделах мы приводим алгоритмы, которые для двух данных последовательностей S_1 и S_2 строят все Парето-оптимальные выравнивания и их веса относительно следующих весовых функций:

(*a*) $VD(A) = (NumMatch(A), -NumDel(A))$;

(*b*) $VG(A) = (NumMatch(A), -NumGap(A))$;

(*c*) $VGD(A) = (NumMatch(A), -NumDel(A), -NumGap(A))$;

(*d*) $WD(A) = (WeightMatch(A), -NumDel(A))$;

(*e*) $WG(A) = (WeightMatch(A), -NumGap(A))$;

(*f*) $WGD(A) = (WeightMatch(A), -NumDel(A), -NumGap(A))$.

Отметим, что функции (a)–(c) являются частными случаями функций (d)–(f) для единичной весовой матрицы замен.

Мы начнем с алгоритма Pareto_Align_Del (см. Рис. 2.3.3), который решает задачу для весовой функции (a).

Входными данными для алгоритма являются две последовательности S_1 и S_2 и их длины n и m. Кроме того, в качестве входа алгоритма используется еще один параметр - число Dmax, максимально допустимое число удаленных символов. Введение этого параметра носит технический характер и обсуждается ниже. Отметим, что в любом выравнивании $D\max \le n + m$.

Алгоритм Pareto_Align_Del (см. рис. 2.3.3) – это алгоритм динамического программирования над полукольцом Парето-оптимальных множеств (см. главу 1).

Каждое выравнивание представляется списком «событий», причем возможны три вида событий: удаление последнего символа в первой последовательности (обозначение: del_1), удаление последнего символа во второй последовательности (обозначение: del_2), сопоставление последних символов из сравниваемых последовательностей, эти символы могут не совпадать (обозначение: no_del). Алгоритм основан на следующем утверждении.

Лемма 2.3.3. Пусть S_1 и S_2 – последовательности; их длины равны соответственно n и m и A – Парето-оптимальное выравнивание этих последовательностей. Пусть B – выравнивание последовательностей T_1 и T_2, которое получается из A удалением последнего события e (если $e = del\_1$, то $T_1 = S_1$[1..n-1] и $T_2 = S_2$; если $e = del\_2$, то $T_1 = S_1$ и $T_2 = S_2$[1..m-1]; если $e = nodel$, то $T_1 = S_1$[1..n-1] и $T_2 = S_2$[1..m-1]. Тогда B - Парето-оптимальное выравнивание последовательностей T_1 и T_2.

Доказательство несложно и здесь не приводится.

В процессе работы алгоритм Pareto_Align_Del для каждой пары *(i, j)* строит множество Парето-оптимальных выравниваний префиксов S_1[1..i] и S_2[1..j], ($1 \le i \le n$; $1 \le j \le m$; эти множества хранятся в матрице *ParetoMatrix[1..n, 1..m]*. В матрице *ParetoMatrix* множество выравниваний представлено множеством весов, причем для каждого веса указано, каким может быть последнее событие, у имеющих данный вес Парето-оптимальных выравниваний соответствующих префиксов (см. тип Alignment и поле LINK в нем). Такая структура (фактически – граф, вершинами которого являются тройки вида *(i, j, w), 1≤ i≤ n; 1≤j≤m, w* – вес выравнивания, а ребра задаются полями LINK)

```
algorithm Pareto_Align_Del(S1,S2: array of char; n,m: integer; Dmax: integer)
typedef Alignment=record begin
//   Описание выравнивания: вес и ссылка на предшественника в матрице
// динамического программирования
MATCHES: integer;        // количество совпадений
DELETIONS: integer;  // количество удаленных символов
LINK:  non-empty subset of set {no_del, del_1, del_2}
                         // событие в конце выравнивания */
end
var
ParetoMatrix: array[1..n, 1..m] of set of Alignment;
P, P0, P1, P2: set of Alignment;
begin
/* А. Инициализация – заполнение 1-й строки в ParetoMatrix */
```

```
if (S1[1]=S2[1])
then ParetoMatrix[1, 1]:={(1, 0, nil)}
else ParetoMatrix[1, 1]:={(0, 0, nil)};

for j:=2 to n do begin
        if (j ≥D max+1)
        then ParetoMatrix[1, j]:= NULL;
        else begin
                if ((S1[j]=S2[1])||(ParetoMatrix[1, j–1][1].MATCH>0))
                then ParetoMatrix[1, j]:={(1, j–1, nil)};
                else ParetoMatrix[1, j]:={(0, j–1, nil)};
        end
end
/* B. Основной цикл построчное заполнение ParetoMatrix */
for i:=2 to n do begin
                /* B1. Заполнить 1-й элемент строки */
if (i ≥D max+1)
then    ParetoMatrix[1, j]:= NULL;
else begin
        if ((S1[1]=S2[i]) || (ParetoMatrix[i–1,1][1].MATCH>0))
        then ParetoMatrix[i, 1]:={(1, i–1, nil)};
        else  ParetoMatrix[i, 1]:={(0, i–1, nil)};
        end
end
                /* B3. Цикл заполнения очередной строки */
for j:=2 to m do begin
/* Разбор случаев, связанных с удалением последнего символа */
        P1:={(m, d, {del_1})| d≤D max &&( ∃ x: (m, d–1, x) ∈ ParetoMatrix[i–1, j]) } ;
        P2:={(m, d, {del_2})| d≤D max && ( ∃ x: (m, d–1, x) ∈ ParetoMatrix[i, j–1]) };
                /* Разбор сопоставления последних символов */
        if (S1[i]=S2[j])
        then  P0:={(m, d, {no_del}) | ∃ x: (m–1, d, x) ∈ ParetoMatrix[i–1, j–1] };
        else  P0:={(m, d, {no_del}) | ∃ x: (m, d, x) ∈ ParetoMatrix[i–1, j–1] };
                /* construction of the Pareto subset of the union of P1, P2, P0 */
        ParetoMatrix[i,j] = PARETO (P0 ∪P1 ∪P2)
end
return (ParetoMatrix[n, m]);
end
```

Рис. 2.3.3**.** Алгоритм Pareto_Align_Del.

позволяет после завершения работы алгоритма восстановить все Парето-оптимальные выравнивания.

Множества P_0, P_1, P_2 содержат все Парето-оптимальные выравнивания текущих префиксов $S_1[1..i]$ и $S_2[1..j]$ (см. лемму 2.3.3), но, возможно, и некоторые выравнивания, которые не являются Парето-оптимальными. Корректность работы алгоритма может быть легко доказана по индукции.

2.3.3. Алгоритм построения Парето-оптимальных выравниваний для весовой функции $VD(A) = (Match(A), –NumDel(A))$.

Рассмотрим следующий алгоритм Pareto_Align_Del_Dmax. Входом алгоритма Pareto_Align_Del_Dmax являются две символьные

последовательности S_1 и S_2, а также их длины, соответственно, n и m. Результатом работы алгоритма Pareto_Align_Del_Dmax являются:

1) множество Парето-оптимальных весов последовательностей S_1 и S_2 относительно весовой функции $VD(A) = (Match(A), -NumDel(A))$;
2) граф, представляющий множество всех Парето-оптимальных выравниваний последовательностей S_1 и S_2 (см. выше).

Алгоритм Pareto_Align_Del_Dmax работает следующим образом. Сначала вычисляется длина p наибольшей общей подпоследовательности последовательностей S_1 и S_2, например, с помощью алгоритма [148]. Затем полагается $D\max = m + n - 2p$ и вызывается алгоритм для получения искомых Парето-оптимальных весов и выравниваний.

Лемма 2.3.4. Пусть S_1 и S_2 – последовательности; их длины равны соответственно n и m. Пусть p – длина наибольшей общей подпоследовательности S_1 и S_2; $d = m + n - 2p$ и $r = \min(p, d)$.

Алгоритм Pareto_Align_Del_Dmax строит множество Парето-оптимальных весов и Парето-оптимальных выравниваний для последовательностей S_1 и S_2 относительно весовой функции $VD(A) = (Match(A), -NumDel(A))$ за время $O(\min(n, 2d) \bullet m \bullet r \bullet \log(r))$.

Доказательство. Мы сначала оценим время работы алгоритма Pareto_Align_Del_Dmax, а затем – корректность его работы.

1) Инициализация занимает время $O(n)$. Пусть K – наибольшее количество элементов в множествах $ParetoMatrix[i, j]$ $(1 \le i \le n;\ 1 \le j \le m)$. Построение множеств $P1$, $P2$, $P0$ и объедения этих множеств требует времени $O(K)$. Построение Парето-подмножества объединения $P = P1 \cup P2 \cup P0$ может быть выполнено за время $K\log(K)$ [189]. Далее, т.к. для любой пары i, j $(1 \le i \le n;\ 1 \le j \le m)$, такой, что $|i - j| > D\max$, множество $ParetoMatrix[i, j]$ пусто, то каждая строка содержит не более $2 \bullet D\max$ непустых клеток. Таким образом, мы получаем оценку (c – константа)

$$T \le c \min(n, 2D\max)\ m \bullet K \bullet \log(K) \qquad (2.3.1)$$

для времени работы алгоритма Pareto_Align_Del.

Чтобы получить оценку времени через длины последовательностей m, n, оценим величину K – наибольший размер промежуточных Парето-множеств $ParetoMatrix[i, j]$. Пусть $p = \mathrm{LCS}(S_1, S_2)$ – длина наибольшей общей

подпоследовательности последовательностей S_1 и S_2. Как следует из [148], время вычисления величины p мало по сравнению с оценкой (2.3.1). Любые два элемента множества *ParetoMatrix*[i, j] ($1 \le i \le n$, $1 \le j \le m$) имеют различные значения, как в поле MATCHES, так и в поле DELETIONS. Значения поля MATCHES – это целые числа от 0 до p. В поле DELETIONS могут находиться только целые числа от 0 до Dmax. Отсюда получаем $K \le min(p, D\text{max})$, что, в силу выбора значения Dmax и дает искомую оценку времени работы.

2) Для доказательства корректности работы алгоритма Pareto_Align_Del_DMax (учитывая проведенный выше анализ работы алгоритма Pareto_Align_Del) достаточно показать, что корректным является значение $D\text{max} = m + n - 2p$, т.е. это ограничение не приводит к потере Парето-оптимальных выравниваний. Пусть A – некоторое Парето-оптимальное выравнивание S_1 и S_2; $VD(A)$ = (s, -d). Рассмотрим выравнивание H, задаваемое наибольшей общей подпоследовательностью S_1 и S_2. Очевидно, $VD(H)$ = (p, -(m+n-2p)) = (p, -Dmax). По определению наибольшей общей подпоследовательности, s ≤ p. Поэтому d ≤ Dmax = $m + n - 2p$ (в противном случае выравнивание H мажорировало бы выравнивание A) и, следовательно, все выравнивания префиксов S_1[1..i]и S_2[1..j] индуцированные Парето-оптимальным выравниванием A, содержат не более x = $m + n - 2p$ удаленных символов. Лемма 2.3.4 доказана.

2.3.4. Учет количества удаленных сегментов.

С биологической точки зрения (см. Введение) необходимо учитывать не только количество удаленных символов (см. п. 2.3.3), но и количество удаленных сегментов (делеций). При традиционном подходе для этого вводится штраф за «инициализацию делеции» (“gap opening penalty” [138]). В рамках нашего подхода количество делеций учитывается явно с помощью элементарной весовой функции *NumGap*(A) (см. п.2.3.2) и соответствующих векторных функций, например,

$$VG(A) = (NumMatch(A), -NumGap(A));$$

$$VGD(A) = (NumMatch(A), -NumDel(A), -NumGap(A)).$$

Алгоритмы, которые строят множество Парето-оптимальных выравниваний для этих весовых функций (они будут обозначаться соответственно Pareto_Align_Gap_DMax и Pareto_Align_DelGap_DMax) аналогичны приведенному в п. 2.3.3 алгоритму Pareto_Align_Del_DMax; они получаются модификацией базового алгоритма Pareto_Align_Del. Эти изменения аналогичны тем, которые используются в алгоритме Смита-Уотермана [295]). Во-первых, структура Alignment должна содержать еще одно поле GAPS, предназначенное для хранения количества удаленных сегментов. Во-вторых, для каждой пары префиксов $S_1[1..i]$ and $S_2[1..j]$ ($1 \le i \le n$, $1 \le j \le m$) необходимо наряду с множеством Парето-оптимальных выравниваний *ParetoMatrix[i, j]* строить отдельно Парето-оптимальные подмножества среди трех классов выравниваний этих префиксов: выравниваний, которые заканчиваются (1) удалением символа в 1-й последовательности, (2) удалением символа во 2-й последовательности и (3) сопоставлением символов. Рекурсивные соотношения при вычислении этих множеств аналогичны соответствующим соотношениям алгоритма Pareto_Align_Del; чтобы избежать погружения в несущественные детали, мы их не приводим.

Следующая лемма является аналогом леммы 2.3.4.

Лемма 2.3.5. Рассмотрим последовательности S_1 and S_2 длин n и m соответственно. Пусть p – длина наибольшей общей подпоследовательности S_1 и S_2; $d = m + n - 2p$ и $r = \min(p, d)$.

Алгоритм Pareto_Align_Gap_Dmax строит множество Парето-оптимальных весов и Парето-оптимальных выравниваний для последовательностей S_1 и S_2 относительно весовой функции $VG(A) = (Match(A), -NumGap(A))$ за время $O(\min(n, 2d)\, mr\log(r))$.

Доказательство дословно повторяет доказательство утверждения 2.3.4.

Лемма 2.3.6. Рассмотрим последовательности S_1 and S_2 длин n и m соответственно. Пусть p – длина наибольшей общей подпоследовательности S_1 и S_2; $d = m + n - 2p$ и $r = \min(p, d)$.

Алгоритм Pareto_Align_DelGap_Dmax строит множество Парето-оптимальных весов и Парето-оптимальных выравниваний для

последовательностей S_1 и S_2 относительно весовой функции $VDG(A) = (Match(A), -NumDel(A), - NumGap(A))$ за время $O(\min(n, 2d)mrd\log(rd))$.

Доказательство. Доказательство корректности работы алгоритмов следует из приведенных выше рассуждений.

Оценка времени работы алгоритмов Pareto_Align_Gap_DMax и Pareto_Align_DelGap_DMax получается так же, как и для алгоритма Pareto_Align_Del_DMax. Изменения относятся только к оценке величины K – максимальному количеству Парето-оптимальных весов в промежуточных множествах *ParetoMatrix[i,,j]*. Отметим, что для размерности 3 построение Парето-подмножества так же, как и для случая размерности 2, выполняется за время *O(KlogK).*Для функции $VG(A) = (NumMatch(A), -NumGap(A))$ верна та же оценка, что и для функции $VD(A) = (NumMatch(A), -NumDel(A))$, доказательство может быть повторено дословно.

Для функции $VGD(A) = (NumMatch(A), -NumDel(A), -NumGap(A)) \le m)$ рассмотрим Парето-оптимальное подмножество *PS* весов выравниваний $S_1[1..i]$ и $S_2[1..j]$, в которых сопоставлены символы $S_1[i]$ и $S_2[j]$. Для каждой пары значений GAPS and DELETION, в *PS* присутствует не более одного веса. Поэтому, $K \le (D\max)^2 \le d^2$. Аналогично, рассматривая пару полей MATCHES, DELETIONS, получаем $K \le pd$. Это завершает доказательство утверждения 2.3.6.

2.3.5. Другие варианты весовых функций.

2.3.5.1. Произвольные матрицы замен..

Подобно алгоритмам выравнивания со скалярными весами, алгоритмы Pareto_Align_Del_DMax, Pareto_Align_Gap_DMax и Pareto_Align_DelGap_DMax могут быть модифицированы для работы с произвольными весовыми матрицами замен. Для этого нужно заменить поле MATCHES (количество совпадений в выравнивании) в структуре Alignment полем SUBST_SCORE, в котором будет хранится суммарный вес сопоставлений при выбранной весовой матрице замен. Оценки времени для алгоритма построения множества Парето-оптимальных выравниваний строятся аналогично.

Учитывая то, что теория весовых матриц замен хорошо разработана [26], использование весовых матриц, по нашему мнению, не является критичным. Предлагаемый подход может быть применен и для учета сходства

между различными буквами на основе отношения эквивалентности на множестве используемых символов (см. раздел 4.3). Например, в случае ДНК, можно считать эквивалентными пурины и пиримидины соответственно. В этом случае можно рассмотреть, например, 3-мерную весовую функцию *Class(A) = (NumMatch(A), NumClass(A)* и *–NumDel(A),* где *NumClass(A)* – количество сопоставлений эквивалентных символов в выравнивании *A*.

2.3.5.2. Локальные выравнивания.

Традиционно задача локального выравнивания сводится к задаче глобального выравнивания с помощью введения нулевых штрафов за делецию на концах последовательностей. Соответствующий алгоритм отличается от алгоритма глобального выравнивания только в двух аспектах: (a) при заполнении очередной клетки матрицы динамического программирования отрицательные веса следует заменить нулевыми; (b) итоговый вес (и соответствующее выравнивание) соответствует не последней клетке последней строки, а клетке, содержащей максимальное значение веса. Близкий подход применим и для Парето-оптимальных выравниваний. Отметим, что, в отличие от глобальных выравниваний, для локальных выравниваний векторная весовая функция

$$VD(A) = (NumMatch(A), -NumDel(A))$$

не является адекватным аналогом скалярной весовой функции

$$V(A) = k_1 NumMatch(A) - k_2 NumDel(A) - k_3 NumMisMatch(A),$$

поскольку количество несовпадений не выражается однозначно через количество совпадений и количество удаленных символов. Из этой ситуации можно выйти двумя путями: (1) явно ввести в векторную весовую функцию еще одну компоненту–*NumMisMatch*(*A*) или (2) заменить компоненту *NumMatch*(*A*) компонентой $Weight(A) = k_1 NumMatch(A) - k_2 NumDel(A)$, что соответствует использованию весовой матрице замен с k_1 на диагонали и $-k_2$ вне ее.

2.4. Построение биологически-корректного выравнивания без явного задания штрафов за делеции.

2.4.1. Общее описание подхода. Критическое выравнивание.

Рассмотри две последовательности u и v и фиксируем некоторую весовую матрицу замен. Для произвольного целого g, через $S(g)$ обозначим наибольший суммарный вес сопоставлений среди выравниваний, содержащих не более g удаленных фрагментов. Через $D(g)$ обозначим «производную» $S(g)$:

$$D(g) = S(g+1) - S(g)$$

Значения $S(g)$ могут быть найдены, например, путем построения Парето-оптимальных выравниваний (см. раздел 2.3) относительно векторной весовой

$$WG(A) = (WeightMatch(A), -NumGap(A))$$

Компьютерные эксперименты со случайными и реальными последовательностями (см. ниже) показали следующее.

1. Пусть u – случайная последовательность и последовательность v получена из u с помощью случайных замен, но без удаления и вставок. Тогда для всех g значения $S(g)$ возрастают медленно, т.е. значения D(g) малы (Рис.. 2.4.1, кривая *1*).

2. Пусть u – случайная последовательность и последовательность v получена из u с помощью случайных замен, а также некоторого количества удаления и/или вставок. Тогда сначала $S(g)$ возрастает достаточно быстро, соответственно, значения D(g) велики. Начиная с некоторого значения g (назовем его «критическим»), $S(g)$ начинает расти медленно, т.е. D(g) мало (Рис. 2.4.1, кривые *2* и *3*). Оптимальное выравнивание, соответствующее критическому значению g, будем называть критическим.

3. Пусть u и v– гомологичные аминокислотные последовательности, для которых известно биологически корректное выравнивание (например, полученное, как выравнивание пространственных структур). Тогда поведение $S(g)$ аналогично одному из двух описанных выше случаев, в зависимости от того, есть ли делеции в биологически корректном выравнивании (см. Рис.2.4.2).

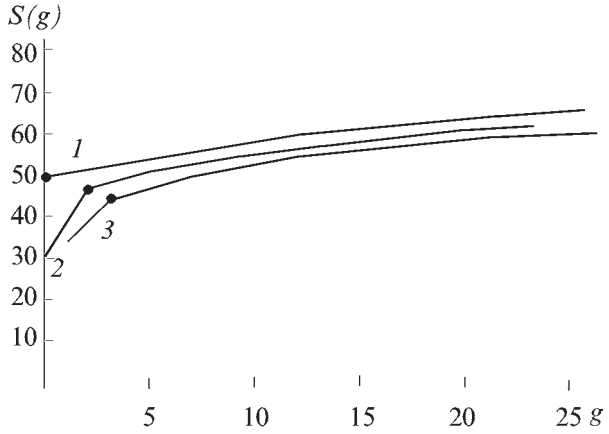


Рис. 2.4.1. Максимальный суммарный вес сопоставлений $S(g)$ в выравниваниях, содержащих не более g удаленных фрагментов. Кривые *1*, *2*, *3* демонстрируют результаты, полученные для трех пар случайных нуклеотидных последовательностей с единичной матрицей весов сопоставлений. U_i и V_i $(i = 1,2,3)$. Кривая *1*: Первая последовательность U_1 – это случайная бернуллиевская нуклеотидная последовательность длиной 100. Вторая последовательность V_1 получена из U заменами в 50 случайных позициях. Кривая *2*: Последовательности U_2 and V_2 получены из последовательностей U_1 and V_1 удалением фрагмента длины 5 в каждой из последовательностей. Кривая *3*: Последовательности U_3 and V_3 получены из последовательностей U_1 and V_1 удалением двух фрагментов длины 5 в U_1 и одного фрагмента длины 5 в V_1 .
Точками отмечены критические значения g_{cr} аргумента g. Для кривой 1: $g_{cr}=0;\ S(g_{cr})=50;$ максимальное значение S_{max} величины S достигается при $g=26:\ S_{max}=66.$ Для кривой 2: $g_{cr}=2;\ S(g_{cr})=47;$ максимальное значение S_{max} величины S достигается при $g=23:\ S_{max}=62.$ Для кривой 3: $g_{cr}=3;\ S(g_{cr})=45;$ максимальное значение S_{max} величины S достигается при $g=28:\ S_{max}=62.$

Величина скачка, которое испытывает значение D(g) при критическом значении g зависит от степени сходства между последовательностями; при этом в «регулярных» случаях этот скачок хорошо различим. Например, в примере Рис.2.4.1 значения D(g) не менее 30 при до-критических значениях g и не более 10 после этого значения.

Это наблюдение может быть интерпретировано следующим образом. Допустим, удалением некоторого количества фрагментов из последовательностей u и v можно получить высокогомологичные последовательности равной длины, эти фрагменты назовем правильным. Удаление «правильного» фрагмента восстанавливает соответствие между гомологичными фрагментами исходных последовательностей и, тем самым, ведет к существенному увеличению веса $S(g)$ (значение $D(g)$ велико). Когда все «правильные» фрагменты удалены (это соответствует критическому

значению *g),* новые удаления уже не мотивированы биологически и, следовательно, не могут привести к существенному увеличению *S*(*g*) (малые значения *D(g))*.

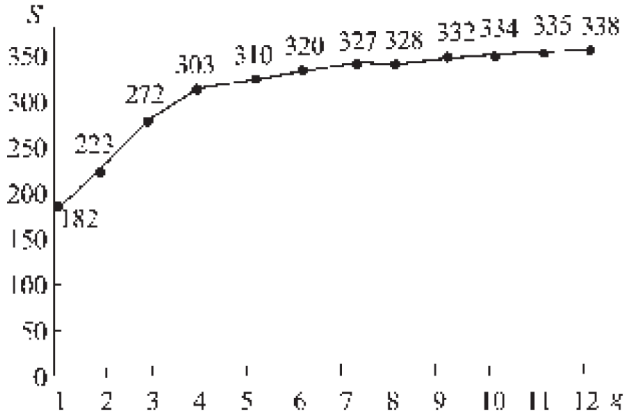


Рис.2.4.2. Значения величины *S*(*g*) при сравнении α и β цепей гемоглобина лошади (PDB код 2mhb). Критическое значение количества удаленных фрагментов $g_{cr}=4$; $S(g_{cr}) = 303$.

Таким образом, критическое значение *g* соответствует биологически корректному выравниванию данных последовательностей. Это приводит к следующему методу выравнивания последовательностей.

1. Построить множество Парето-оптимальных выравниваний относительно векторной весовой функции $WG(A) = (WeightMatch(A), –NumGap(A))$.
2. Определить критическое значение количества удаленных фрагментов *g* и взять критическое выравнивание в качестве результата.

Успешность применения этого метода зависит от алгоритма, который определяет порог D_{cr}, отделяющий «малые» и «большие» значения *D*(g). Оставшаяся часть настоящего параграфа посвящена этому вопросу. В разделе 2.4.2 мы опишем компьютерные эксперименты со случайными последовательностями; в разделе 2.4.3 – эксперименты с аминокислотными последовательностями реальных белков. Теоретический анализ дан в разделе 2.4.4.

2.4.2. Вычислительные эксперименты.

Чтобы проверить сформулированную выше гипотезу, было проведено четыре серии вычислительных экспериментов – три серии с нуклеотидными последовательностями и одна серия с аминокислотными

последовательностями. Целью экспериментов было изучение зависимости поведения величины $D(g)$ от длин сравниваемых последовательностей и степени их сходства. В каждом эксперименте мы вычисляли два значения: D_{cr} (максимальное значение D(g) в закритическом интервале, т.е. для малых значений D(g)) и D_{pr} (минимальное значение D(g) в докритическом интервале, т.е. при больших значениях D(g)).

Сравниваемые последовательности были получены с помощью следующей процедуры. Первая последовательность из пары была получена как случайная Бернуллиевская последовательность с равномерным распределением вероятностей. Вторая последовательность была получена из первой путем случайных замен (все замены равновероятны) в случайных местах. При этом количество замен было фиксировано. Далее производились делеции; процедура внесения делеций была различна в различных экспериментах и описана отдельно для каждого эксперимента.

В качестве меры сходства во всех экспериментах используется величина *SAVE* – отношение числа консервативных (неизмененных) позиций к исходной длине последовательностей. В качестве матрицы замен для нуклеотидных последовательностей мы использовали единичную матрицу (вес совпадения 1, вес несовпадения 0). Таким образом, величина *WeightMatch* в этом случае равна количеству совпадений. Для каждой пары последовательностей было построено множество Парето-оптимальных выравниваний относительно весовой функции *WG(A) = (WeightMatch(A), -NumGap(A))*.

А. Эксперименты с нуклеотидными последовательностями. В каждом тесте для каждого значения длины последовательности *SeqLen* = 200, 300, 700 and 1000, были сгенерированы по 140 случайных нуклеотидных последовательностей; в этих последовательностях были проведены случайные замены с коэффициентами консервативности *SAVE* = 0.3, 0.4, 0.5, 0,6, 0.7, 0.8, 0.9 (по 20 последовательностей для каждого значения величины *SAVE)*.

Тест 1. Только замены. Делеции не производились, так что сравниваемые последовательности были одинаковой длины.

Результат (см.таблицу 2.4.1) . Функция $S(g)$ не имеет скачков. Критическое значение *g=0.*

Таблица 2.4.1.

Длина	SAVE						
	0.3	0.4	0.5	0.6	0.7	0.8	0.9
200	7.3	5.8	4.1	3.1	2.7	1.1	1
300	7.3	5.9	4.15	3.5	3.1	1.6	1.1
700	7.8	6.15	4.25	3.6	2.7	1.75	1.2
1000	8.05	6.1	4.35	3.25	2.9	2	1.45

Результаты теста 1. Зависимость среднего значения величины D_{cr} (максимальное значение $D(g)$ в закритической области) от длины сравниваемых последовательностей и степени их сходства *SAVE*.

Тест 2. Один удаленный фрагмент в одной последовательности.

В случайном месте первой последовательности был удален фрагмент длиной 10. Таким образом, сравнивались последовательности разной длины.

Результат. Критическим значением является значение $g=1$. В большинстве случаев в критическом выравнивании положение удаленного сегмента восстановлено точно (см. таблицу 2.4.2).

Тест 3. Удаление двух сегментов (по одному в каждой из последовательностей). В каждой из сравниваемых последовательностей был удален фрагмент длиной 5 нуклеотидов; расстояние между местами удаления во всех случаях составляло 120 нуклеотидов. Таким образом, сравниваемые последовательности были одной длины.

Таблица 2.4.2.

Длина	SAVE						
	0.3	0.4	0.5	0.6	0.7	0.8	0.9
200	7.25	5.2	4.15	3.15	2.8	1.6	1
300	7.4	5.1	4.3	3.5	3.4	1.7	1
700	7.8	5.85	4.5	3.55	2.65	1.7	1.1
1000	8.1	6.15	4.8	3.6	2.8	1.9	1.3

Результаты теста 2. Зависимость среднего значения величины D_{cr} (максимальное значение $D(g)$ в закритической области) от длины сравниваемых последовательностей и степени их сходства *SAVE*.

Результат. Во множестве Парето-ортимальных выравниваний есть безделеционное выравнивание (*NumGap* = 0). Критическим выравниванием

является выравнивание, соответствующее значению $g = NumGap = 2$ (см. таблицы 2.4.3, 2.4.4). При $SAVE > 0.4$ положение удаленных символов восстановлено точно.

Б. Эксперимент для аминокислотных последовательностей *(Тест 4)*. Был проведен один тест (тест 4) – аналог теста 3, проведенного для нуклеотидных последовательностей. Для каждой длины последовательности *SeqLen* = 100, 200, 300 были сгенерировано по 160 случайных бернуллиевских аминокислотных последовательностей. Вероятности замен соответствовали матрице замен BLOSUM62 [145]. Для параметра *SAVE* были взяты значения от 0.2 до 0.9 с шагом 0.1.

Результат. Так же, как и в случае теста 3, критическим значением является значение *NumGap* = 2 (см. таблицы 2.4.5, 2.4.6). Положение удаленных сегментов установлено со сдвигом от 0 до 5 нуклеотидов.

Таблица 2.4.3.

Длина	**SAVE**						
	0.3	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***
200	**11.45**	**18.2**	**29.6**	**41.7**	**54.1**	**65.6**	**77.9**
300	**12.1**	**19.3**	**33.1**	**40.8**	**56**	**64.7**	**77**
700	**9.5**	**19.8**	**30.9**	**41.7**	**53.3**	**66.2**	**75.45**
1000	**10.1**	**20.6**	**30.5**	**43.3**	**54.2**	**67.85**	**78.25**

Результаты теста 3. Зависимость среднего значения величины D_{pr} (минимальное значение $D(g)$ в докритической области) от длины сравниваемых последовательностей и степени их сходства *SAVE*.

Результаты экспериментов показывают, что докритические значения величины $D(g)$ не зависят от величины критического значения $g_{ск}$ и отделены от закритических значений величины $D(g)$. В частности, пороги 8 и 9 может использоваться для сравнения нуклеотидных последовательностей длины до 1000 нуклеотидов.

Таблица 2.4.4.

Длина	**SAVE**						
	0.3	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***
200	**7.35**	**5.5**	**3.95**	**3.1**	**1.9**	**1.1**	**1.1**

300	**7.3**	**5.85**	**3.65**	**3.1**	**2.25**	**1.6**	**1**
700	**7.45**	**5.95**	**4**	**3.55**	**2.15**	**1.55**	**1**
1000	**7.65**	**6.15**	**4.1**	**3.4**	**2.4**	**1.7**	**1.4**

Результаты теста 4. Зависимость среднего значения величины D_{cr} (максимальное значение $D(g)$ в закритической области) от длины сравниваемых последовательностей и степени их сходства *SAVE*.

Таблица 2.4.5.

Длина	**SAVE**							
	0.2	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***
100	**74.8**	**115.9**	**157.2**	**198**	**227.1**	**271.7**	**306.6**	**335.4**
200	**69.6**	**123.9**	**161.3**	**179.5**	**235.9**	**280.2**	**300.4**	**332.4**
300	**71.3**	**121.7**	**165.1**	**177.8**	**222.7**	**269.3**	**321.1**	**328.9**

Результаты теста 4. Зависимость среднего значения величины D_{pr} (минимальное значение $D(g)$ в докритической области) от длины сравниваемых последовательностей и степени их сходства *SAVE*.

Таблица 2.4.6.

Длина	**SAVE**							
	0.2	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***
100	**22.95**	**18.1**	**15.4**	**12.6**	**10.5**	**8.8**	**7.75**	**6.4**
200	**24.8**	**20**	**15.75**	**13.9**	**10.9**	**9.85**	**8.4**	**7.35**
300	**25.9**	**21.1**	**16**	**14.1**	**11.15**	**10.2**	**8.8**	**7.9**

Результаты теста 4. Зависимость среднего значения величины D_{cr} (максимальное значение $D(g)$ в закритической области) от длины сравниваемых последовательностей и степени их сходства *SAVE*.

2.4.3. Сравнение глобинов.

Предложенный подход был протестирован на сравнении последовательностей глобинов. Эталонные выравнивания были взяты из базы структурных выравнивания 3DALI [61] . В таблице 2.4.7 приведены сводные данные для четырех пар выравниваний с различной степенью сходства. Критическое Парето-оптимальное выравнивание было получено на основе весовой матрицы BLOSUM62 и весовой функции $<WeightMatch(A), -NumGap(A)>$. Во всех случаях множество Парето-оптимальных весов имеет структуру, описанную в п. 3.1 и полученные значения величин *D*cr и *D*pr хорошо согласуются с данными таблиц 2.4.5 и 2.4.6. Критические выравнивания достаточно близки к выравниваниям, построенным по пространственной структуре.

Отметим, что в тех случаях, когда критическое выравнивание не совпадает со структурным, последнее вообще не является Парето-оптимальным, даже по отношению к весовой функции *WGD(A)* = (*WeightMatch*(*A*), –*NumGap*(*A*), –*NumDel*(*A*)). Следовательно (см. п.2.3), при использовании стандартной весовой матрицы сопоставлений BLOSUM62 это выравнивание не будет оптимальным ни для какой весовой функции вида

$$WeightMatch(A) - k_1 NumGap(A) - k_2 NumDel(A)$$

Вопрос о соответствии между структурными и алгоритмически оптимальными выравниваниями подробно рассмотрен в разделе 3.1.

2.4.4. Теоретические оценки для D_{cr} и D_{pr}.

Напомним, что D_{cr} – это максимальное значение величины *D(g)* для закритической области значение *g* ($g \geq g_{cr}$), а D_{pr} – это минимальное значение величины *D(g)* для докритической области значение *g* ($g < g_{cr}$). Мы исследуем поведение этих величин для единичной весовой матрицы сопоставлений и для векторной весовой функции *WG(A)* = <*WeightMatch*(*A*), –*NumGap*(*A*)>.

2.4.4.1. Поведение D_{cr}.

Рассмотрим статистическую модель, близкую к модели теста 1 (см. п.2.4.4.2). А именно, рассмотрим случайную бернуллиевскую последовательность *X* длины *L* в *K*-буквенном алфавите (все буквы равновероятны). Последовательность *Y* получается из *X* случайными заменами символов (вероятность замены символа *X[i]* равна $p < 1/K$; замена на все допустимые символы равновероятна). Нас будет интересовать максимальная величина увеличения суммарного веса сопоставлений *dS,* который может быть получен при удалении из каждой из последовательностей по одному фрагменту произвольной длины (так как исходные последовательности имели одну и ту же длину, то и удаляемые фрагменты должны быть одной длины).

Таблица 2.4.7

№	Белки			Критическое выравнивание					3D-ALI		Различия	
	Код	Длина	%ID	Dcr	Dpr	WM	NG	ND	NG	ND	N	%
1	2mhb-B	146	0.7	6	н/о	146	0	0	0	0	0	0

	ifdh	146									0	
2	4hhb-A	141	0.5	10	31	325	4	9	4	9	7	4.5
	4hhb-B	146									6	
3	ifdh	146	0.2	11	35	119	3	9	7	13	11	7.5
	1mbd-G	153									12	
4	1mbs	153	0.2	6	40	146	4	28	5	28	21	12
	1lhb	149									17	

Сравнение критических выравниваний (группа столбцов (“Критическое выравнивание”) и структурных выравниваний (группа столбцов (“3D-ALI ”) для четырех пар аминокислотных последовательностей белков семейства глобинов. Для каждого пары указана доля совпадений в структурном выравнивании (%ID). Для критического выравнивания указаны значения величин *D*cr и *D*pr (см. обозначения в п.2.4.2), суммарный вес сопоставлений по матрице BLOSUM62 (WM), количество удаленных фрагментов (NG), суммарное количество удаленных символов (ND). В группе столбцов «Различия» указаны количество позиций белка, по-разному сопоставленных в критическом и структурном выравниваниях в абсолютном и процентном (среднее по двум последовательностям отнесенное к средней длине последовательностей) выражении.

В итоге удаления двух сегментов происходит сдвиг «внутренних» частей последовательностей друг относительно друга. Наложим дополнительное условие: при сдвиге не должны «разрушаться» уже существующие совпадения (это условие тем менее обременительно, чем выше исходная доля совпадений, т.е. чем меньше доля замен $p < 1/K$). Для максимального значения $dSMAX_1$ приращения dS при этом дополнительном условии получаем:

$$dSMAX_1 \le R\text{-}1$$

где R – наибольшее количество идущих подряд несовпадений.

Аналогично [35] можно показать, что при больших длинах последовательностей L, значение R с вероятностью близкой к 1 равно $\log(L)/\log(1/(1-p))$. Это дает нам оценку для величины D_{cr} :

$$D_{\text{cr}} \le \log(L)/\log(1/(1-p)) - 1 \qquad (2.4.1)$$

В таблице 2.4.8 показано что формула (2.4.1) находится в хорошем согласии с вычислительным экспериментом.

Таблица 2.4.8

Length	SAVE						
	0.3	0.4	0.5	0.6	0.7	0.8	0.9

200	**FORM**	**13.8**	**9.4**	**6.6**	**4.8**	**3.4**	**2.3**	**1.3**
	EXP	**10**	**8**	**6**	**4**	**3**	**2**	**1**
300	**FORM**	**15**	**10.2**	**7.2**	**5.2**	**3.7**	**2.5**	**1.5**
	EXP	**10**	**8**	**6**	**5**	**3**	**2**	**1**
700	**FORM**	**17.3**	**11.8**	**8.5**	**6.1**	**4.4**	**3.07**	**1.8**
	EXP	**11**	**9**	**7**	**5**	**3**	**3**	**1**
1000	**FORM**	**18.3**	**12.5**	**9**	**6.6**	**4.7**	**3.3**	**2**
	EXP	**11**	**9**	**7**	**6**	**4**	**3**	**2**

Верхняя оценка величины *D*cr, вычисленная по формуле (2.4.1) (строки FORM) и максимальные значения, полученные в тестах 1-3 из раздела 2.4.3 (строки EXP) для различных значений длины и параметра *SAVE.*

2.4.4.2. Поведение D_{pr}.

Рассмотрим статистическую модель, соответствующую тесту 3. Пусть X - случайная Бернуллиевская последовательность в k-буквенном алфавите; все k символов равновероятны. Пусть последовательность X' получается из последовательности X случайными заменами, причем вероятность того, что буква в последовательности X заменена, равна $p < 1/k$. Удалим из каждой из последовательностей фрагмент длины d, расстояние между удаленными фрагментами равно L. Тогда

$$X = UDVEW;\ X' = U'D'V'E'W'$$

где $|D| = |D'| = |E| = |E'| = d;\ |V| = |V'| = L;$ из слова X удален фрагмент D, из слова X' удален фрагмент E'и, следовательно, после удалений из X и X' получаются соответственно слова $Y = UVEW$ и $Y' = X' = U'D'V'W'$

В безделиционном выравнивании последовательностей X' и Y' друг другу сопоставляются «независимые» фрагменты VE и $D'V'$ длины $L+d;$ ожидаемое число совпадений при их сопоставлении равно

$$M1 = (L+d)/k.$$

В выравнивании с делециями восстанавливается правильное соответствие между фрагментами V и V' длины $L;$ при этом ожидаемое количество совпадений становится равным

$$M2 = pL$$

Таким образом, ожидаемый прирост числа совпадений равен $L(p - 1/k) - d/k.$

Глава 3. Специализированные алгоритмы выравнивания биологических последовательностей.

3.0. Введение.

В главе 1 были рассмотрены общие подходы к задаче построения оптимального выравнивания символьных последовательностей, в частности, - проблема выбора штрафов за удаление фрагментов. В этой главе будет рассмотрена более специальная задача - выравнивание биологических последовательностей. Центральная тема этой главы – учет пространственной структуры сравниваемых молекул при сравнении их первичных структур (последовательностей). Имея это ввиду, мы ограничимся рассмотрением наиболее распространенного в современной биоинформатике класса весовых функций удаления фрагментов – аффинными функциями [141, 139]. Это позволит существенно упростить изложение, хотя приведенные в разделах 3.2 и 3.3 алгоритмы могут быть обобщены на случай рассмотренных в главе 1 кусочно-линейных функций. В разделе 3.4. представлен оригинальный алгоритм предсказания вторичной структуры РНК.

3.1. Алгоритмические и структурные выравнивания аминокислотных последовательностей белков.

3.1.1. Постановка задачи.

В заключительном разделе предыдущей главы мы указали ограничения, которые препятствуют алгоритмическому восстановлению биологически корректных выравниваний нуклеотидных последовательностей. Природа этих ограничений в том, что при выравнивании последовательностей с малым уровнем сходства невозможно по величине прироста веса (т.е. путем выбора штрафа за удаление фрагмента) отделить удаления биологически мотивированные, от удалений, не имеющих биологического обоснования (см. п. 2.4.4). В этом разделе мы изучим связь между биологически корректными выравниваниями аминокислотных последовательностей и их выравниваниями, полученными с помощью наиболее распространенного в настоящее время

алгоритма построения оптимального выравнивания последовательностей – алгоритма Смита-Уотермана [294].

Для оценки качества выравнивания, построенного каким-либо алгоритмом, нужно иметь в распоряжении эталонное выравнивание этих же последовательностей. В качестве эталонных мы насматриваем выравнивания, основанные на наложении пространственной структуры белков (см. раздел 1.2). Таким образом, наша задача состояла в том, чтобы проанализировать степень сходства между алгоритмическими и структурными выравниваниями в зависимости от степени сходства между сравниваемыми последовательностями и выявить причины расхождений между этими выравниваниями. Отметим, что между задачами выравнивания нуклеотидных и аминокислотных последовательностей есть два существенных отличия. Во-первых, для аминокислотных последовательностей есть эталонные выравнивания, с которыми можно сравнивать качество алгоритмически построенных выравниваний. Во-вторых, из-за разнообразия свойств аминокислот при выравнивании аминокислотных последовательностей необходимо использовать нетривиальные весовые матрицы сопоставление [147].

Прежде, чем перейти к изложению результатов, уточним методику исследования.

В качестве количественной оценки качества алгоритмически полученных выравниваний мы использовали две взаимодополняющих меры (см. [313]; [103]):

Точность выравнивания (обозначение: *Ali_Acc)* равна отношению количества одинаковых сопоставлений (*I*) в обоих выравниваниях к общему количеству сопоставлений в эталонном выравнивании (*G*)

$$Ali\_Acc = I/G*100\ \%.$$

Достоверность выравнивания (обозначение: *Ali_Conf)*, равна отношению количества одинаковых сопоставлений в обоих выравниваниях (*I*) к общему количеству сопоставлений в алгоритмически построенном выравнивании (*A*):

$$Ali\_Conf = I/A*100\ \%.$$

Неформально говоря, точность *Ali_Acc* показывает, какую долю эталонного выравнивания удалось восстановить, а достоверность *Ali_Conf* – насколько можно доверять построенному выравниванию.

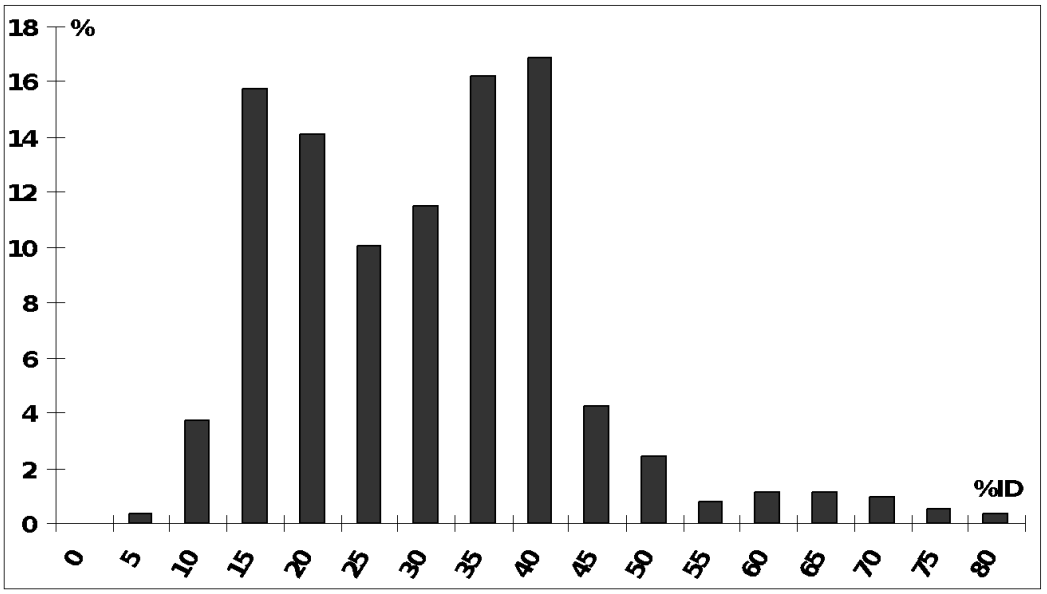


Рис. 3.1.1. Распределение пар белков по уровню гомологии в используемой части базы данных BaliBase.

В качестве меры сходства белков использовался ***%ID*** (identity – идентичность, уровень гомологии) равен отношению числа совпадений к числу сопоставлений эталонном выравнивании (см. ниже).

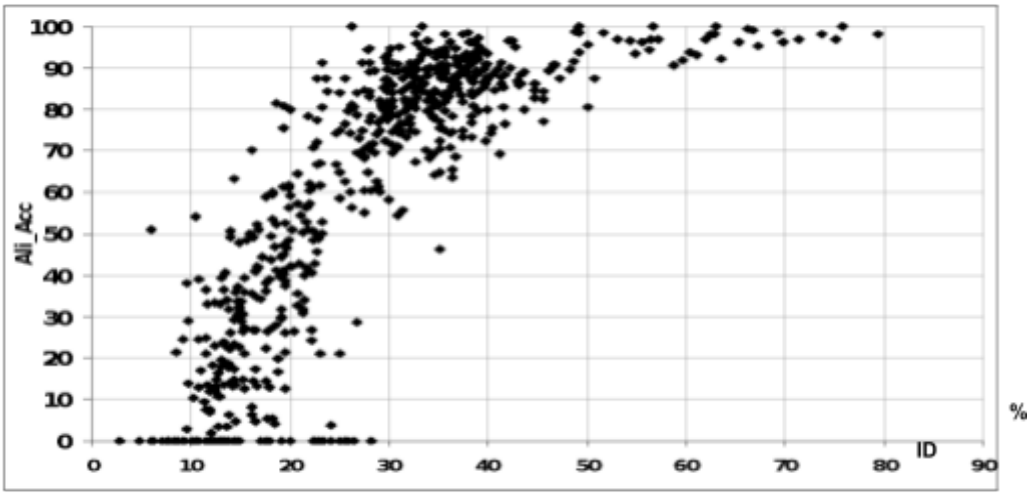
В качестве источника эталонных выравниваний использовалась база данных BaliBase [46]. Эта база содержит множественные выравнивания для нескольких десятков семейств белков, причем биологическая адекватность выравниваний проверена создателями базы. База создавалась для тестирования программ множественного выравнивания, поэтому не все, находящиеся в ней данные могли быть использованы для наших целей. В частности, не рассматривались последовательности, для которых неизвестна трехмерная структура, а также те белки, которые присутствовали одновременно в двух семействах. В результате, в качестве эталонных

тестировалось 583 парных выравнивания. На рис. 3.1.1 приведена гистограмма распределения величины %ID в множестве эталонных выравниваний.

Выравнивания по методу Смита-Уотермана были получены с помощью стандартного алгоритма, реализованного в программном пакете FASTA (модуль dropnsw.c) [253] и откомпилировано для платформы Windows.

Для построения выравниваний методом Смита-Уотермана мы использовали матрицу замен Gonnet250 [138] т.к. для этой матрицы среднее значение точности выравниваний, построенных по банку BaliBase, получается максимальным. Аналогичные результаты были получены ранее в работах [313] и [20]. Однако использование матрицы BLOSUM62 [145] или Gonnet (целочисленная) [138] лишь незначительно уменьшает среднее значение точности.

а)



б)

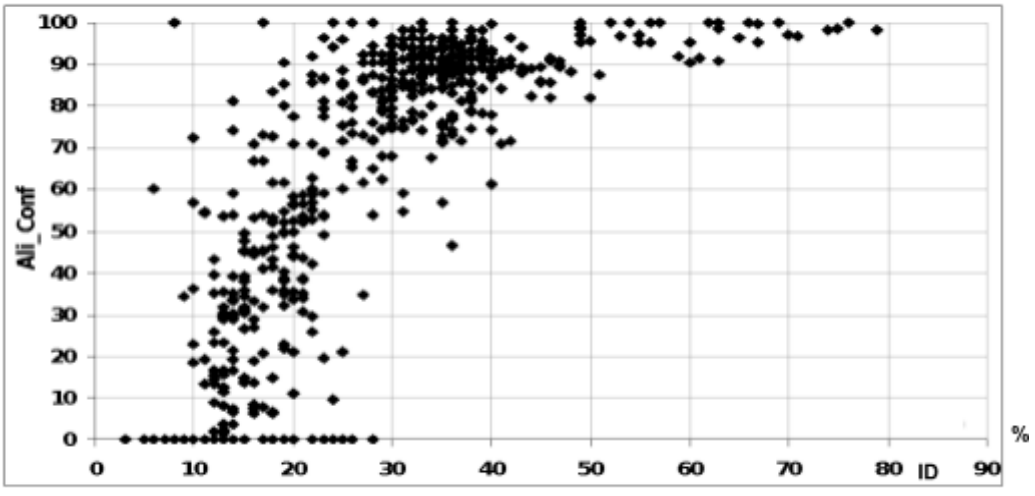


Рис. 3.1.2. Зависимость точности (а) и достоверности (б) восстановления эталонных выравниваний методом SW. Каждая точка соответствует паре эталонных белков. X-координата точки равна %*ID* пары, а *Y*-равный координата – значению точности (а) или достоверности (б) соответствующего эталонного выравнивания. Данные приведены для линейного набора параметров штрафов за делеции. Для логарифмических параметров картина, в целом, такая же с некоторым уменьшением точности и увеличением достоверности.

Мы применяли два набора штрафов за делеции. Наибольшая точность восстановления эталонного выравнивания достигается при значениях 10 за открытие делеции (*GOP*) и 0.5 за её удлинение (*GEP*) (аналогичные параметры указываются как оптимальные в работах [313] и [27]).

Однако стоит заметить, что эти параметры относятся к линейному домену параметров [320], т.е. соответствуют линейному росту длины оптимального выравнивания относительно длины случайной последовательности, и, следовательно, не могут быть использованы для работы с многодоменными белками и для поиска гомологов по банкам данных. Чтобы учесть это обстоятельство, мы также использовали параметры из логарифмического домена [320] (длина оптимального выравнивания растет как логарифм от длины случайной последовательности): штраф за открытие делеции (*GOP*) = 15, штраф за удлинение делеции (*GEP*) = 1. Эти параметры обычно используются для поиска гомологов по банкам данных.

3.1.2. Сравнение эталонных выравниваний и выравниваний Смита-Уотермана.

На рисунке 3.1.2 представлены точечные диаграммы зависимости точности и достоверности. Зависимость, аналогичная представленной на рис. 3.1.2 (а), была получена ранее в работе [313]). Виды диаграмм для точности и достоверности очень похожи.

Как видно из рис. 3.1.2, алгоритм SW может строить выравнивания, хорошо совпадающие с эталонными, только при уровне сходства сравниваемых белков более 30-40%. Этот диапазон уровня гомологии (*ID* > 30%) примерно совпадает с известным порогом *%ID*, выше которого можно

достоверно восстановить структурное выравнивание, зная только последовательности [281, 20, 66, 269].

При уровне гомологии меньше 10% метод SW не может восстановить правильное выравнивание даже частично. Для диапазона уровня гомологии от 10% до 30% выравнивания Смита-Уотермана показывают очень широкий разброс точности и достоверности. Для разных пар последовательностей с одинаковым уровнем сходства построенные SW-выравнивания могут иметь очень различные значения точности и достоверности. Это означает, что в этом диапазоне *%ID*, качество алгоритмических выравниваний определяется не только уровнем сходства сравниваемых белков, но и «внутренними свойствами» их эталонных выравниваний.

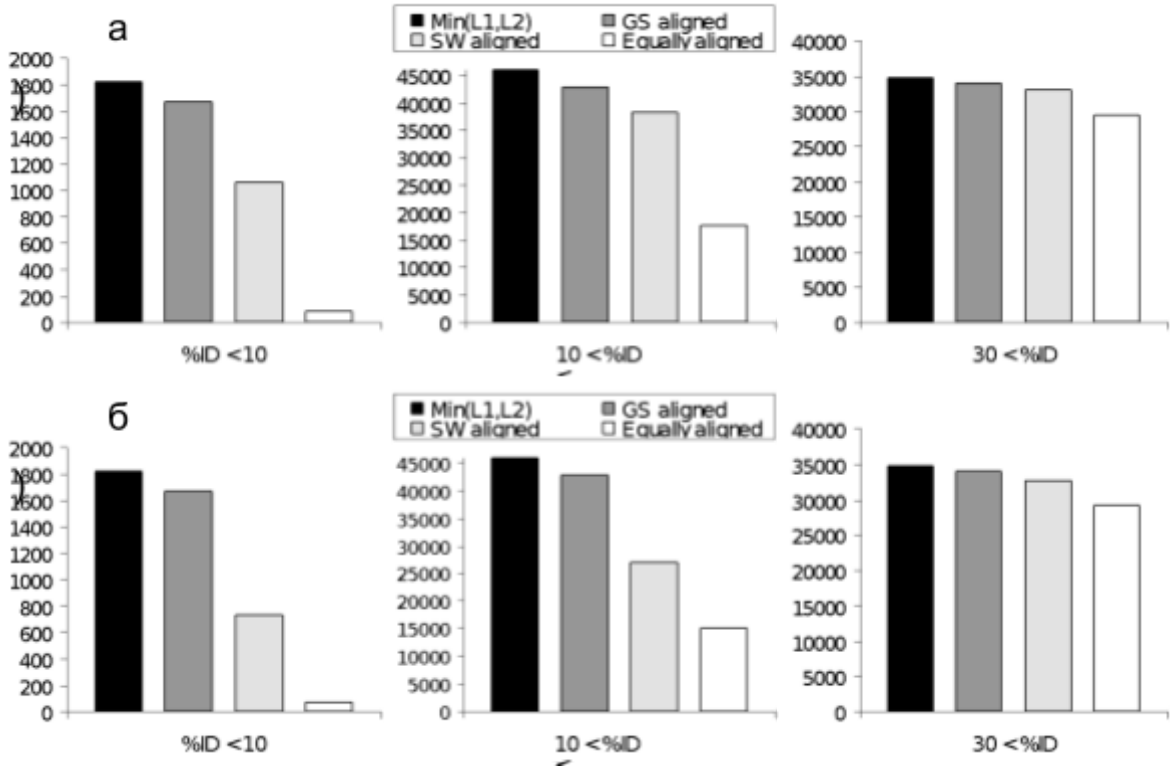


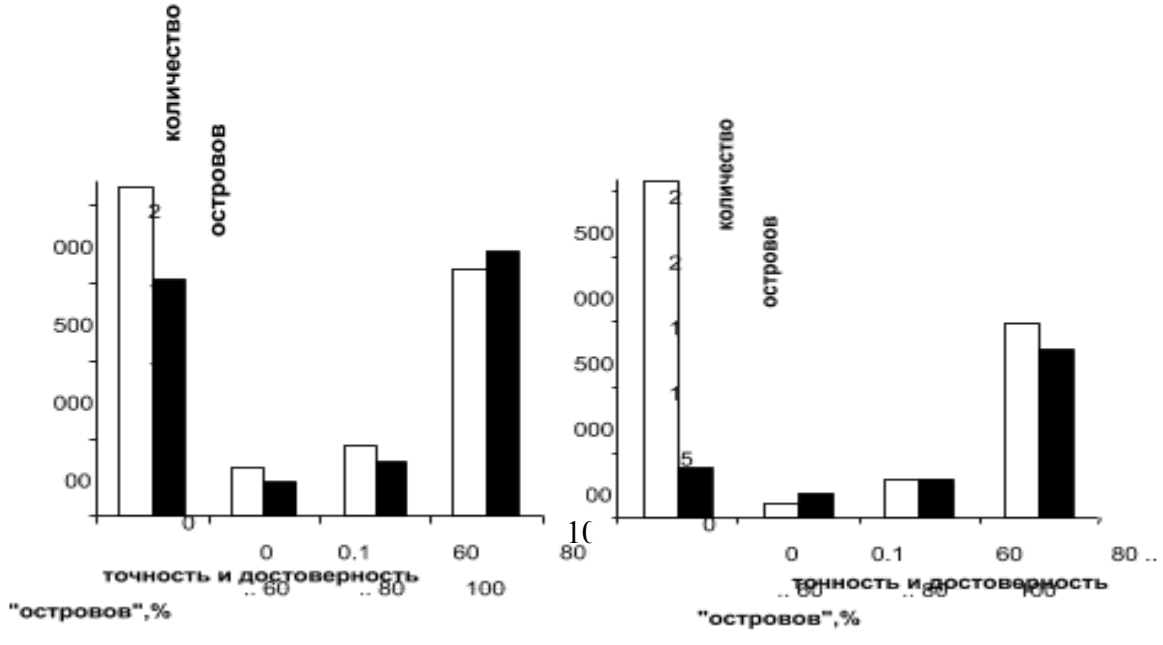
Рис. 3.1.3. Соответствие между эталонными выравниваниями и выравниваниями Смита-Уотермана для 3-х диапазонов уровня гомологии: (а) линейные параметры; (б) логарифмические параметры. Столбцы отображают 4 характеристики выравниваний. Черный столбец – максимально возможное количество позиций, которые могут быть выровнены, т.е. сумма по всем парам величины $L = \min(L_1, L_2)$, где L_1, L_2 – длины последовательностей в паре. Темно-серый – суммарное количество

выровненных позиций в эталонных выравниваниях. Светло-серый – суммарное количество выровненных позиций в выравниваниях SW. Белый – суммарное количество позиций, выровненных одинаково в эталонном выравнивании и выравнивании SW.

Рисунок 3.1.3 дает представление о различиях в длинах между эталонными и алгоритмическими выравниваниями для двух указанных выше наборов параметров. В частности, из рисунка видны причины увеличения достоверности при одновременном уменьшении точности выравниваний при переходе от линейных параметров к логарифмическим. Увеличение штрафов за удаление несколько уменьшает суммарную длину правильно сопоставленных участков, значит, точность выравниваний понижается. В то же время существенно уменьшается длина неправильных сопоставлений, что значительно увеличивает достоверность выравнивания в целом. На диаграммах также видно, что с уменьшением *%ID* количество выровненных позиций в процентах от максимально возможной длины выравнивания ($\min(L_1, L_2)$) существенно уменьшается для эталонных выравниваний, в то время как для алгоритмических практически не изменяется. Даже только из этого следует, что качество построенных выравниваний будет ухудшаться с уменьшением *%ID*.

3.1.3. Острова в эталонных и алгоритмических выравниваний

Определение 3.1.1. Островом в выравнивании $A = \langle u, v, Q \rangle$ называется непродолжаемая последовательность сопоставлений, не разделенных удалениями фрагментов. Весом острова называется сумма весов входящих в остров сопоставлений.



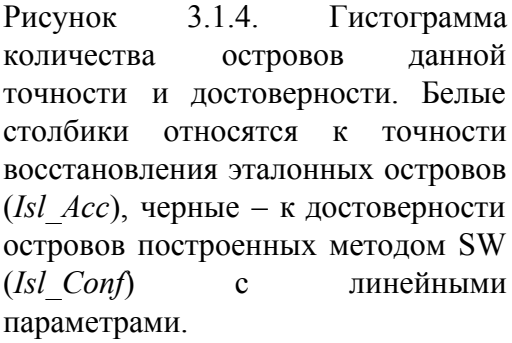


Рисунок 3.1.4. Гистограмма количества островов данной точности и достоверности. Белые столбики относятся к точности восстановления эталонных островов (*Isl_Acc*), черные – к достоверности островов построенных методом SW (*Isl_Conf*) с линейными параметрами.

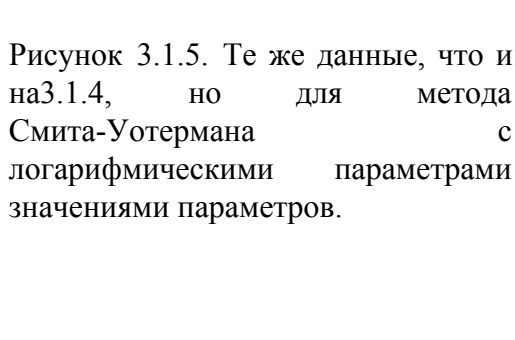


Рисунок 3.1.5. Те же данные, что и на3.1.4, но для метода Смита-Уотермана с логарифмическими параметрами значениями параметров.

Выравнивание можно представить как цепочку островов, соединенных делециями. Мы покажем, что различие в структуре эталонных и алгоритмических выравниваний проявляется именно на уровне островов.

Определение 3.1.2. Точностью восстановления острова эталонного выравнивания (обозначение: *Isl_Acc*) называется отношение количества позиций, одинаково выровненных в построенном и эталонном выравниваниях (*I_isl*), к полной длине эталонного острова (*G_isl*):

Isl_Acc = I_isl / G_isl *100 %.

Достоверностью острова в алгоритмическом выравнивании (обозначение: *Isl _Conf*) называется отношение количества правильных сопоставлений (*I_isl*) к полной длине построенного острова (*A_isl*):

Isl_Conf = I_isl / A_isl *100 %.

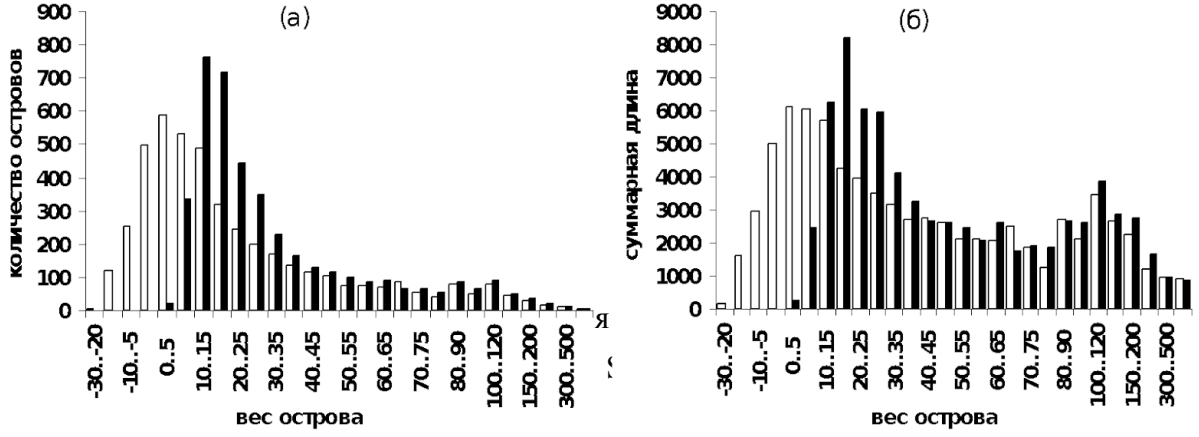
Как видно из рисунков 3.1.4, 3.1.5 остров эталонного выравнивания или достаточно точно восстановлен, или полностью потерян. Частичное восстановление острова случается достаточно редко. Точно также частично достоверные острова в выравниваниях Смита-Уотермана составляют небольшую долю. Дальнейший анализ показал, что это в целом верно для всех диапазонов *%ID* (данные не представлены). Таким образом, острова условно можно поделить на «потерянные» (не имеющие ничего общего с эталонными выравниванием) и «найденные» (содержащие хотя бы одно правильное сопоставление). Это ситуация «все или ничего» не имеет места для точности и достоверности по выравниванию в целом. Например, при линейных параметрах (см. рис. 3.1.2), получилось 69 выравниваний SW с *Ali_Acc* = 0; *Ali_Acc* больше либо равное 80% наблюдается у 249 выравниваний (229 из них

соответствует ID > 30%), в то время как 264 выравнивания имеют *Ali_Acc* в диапазоне 0 – 80%.

Эталонные выравнивания и SW-выравнивания имеют существенно различную структуру островов с точки зрения веса островов (см. рис. 3.1.6). Неожиданно много эталонных островов имеют очень низкий или даже отрицательный вес, в то время как алгоритмические выравнивания совсем не содержат островов малого веса. Стоит отметить, что суммарная длина таких «слабых» островов в эталонных выравниваниях достаточно велика (см. рис. 3.1.6 б).

Эталонные острова веса меньше 5 составляют 32% от всех островов и покрывают 20% всей длины эталонных выравниваний, острова веса < 5 оставляют 65% от всех потерянных островов и покрывают 63% суммарной длины потерянных островов. Только 5% островов такого малого веса были восстановлены алгоритмом. Для выравниваний из серой зоны ($10 < \%ID \leq 30$) картина еще более критическая – восстановлено всего 2.5% островов веса меньше 5. Эти «слабые» острова обычно не имеют шансов быть восстановленными любым алгоритмом, использующим данную матрицу замен. В частности, наличие таких эталонных островов малого веса говорит о неадекватности применения одной и той же матрицы замен для всей длины выравнивания.

Эталонные острова высокого веса (больше 25) почти всегда восстановлены, в то время как острова с весом от 10 до 25 составляют зону неопределенности. Вероятность восстановления острова из этого весового диапазона сильно зависит от наличия альтернативных выравниваний, имеющих более высокие веса.



Ри я
вы

Суммарная длина островов, имеющих вес в пределах диапазонов. Эталонные острова – белый, SW – черный. Выравнивания SW получены с линейными параметрами. Данные для логарифмического домена параметров такие же.

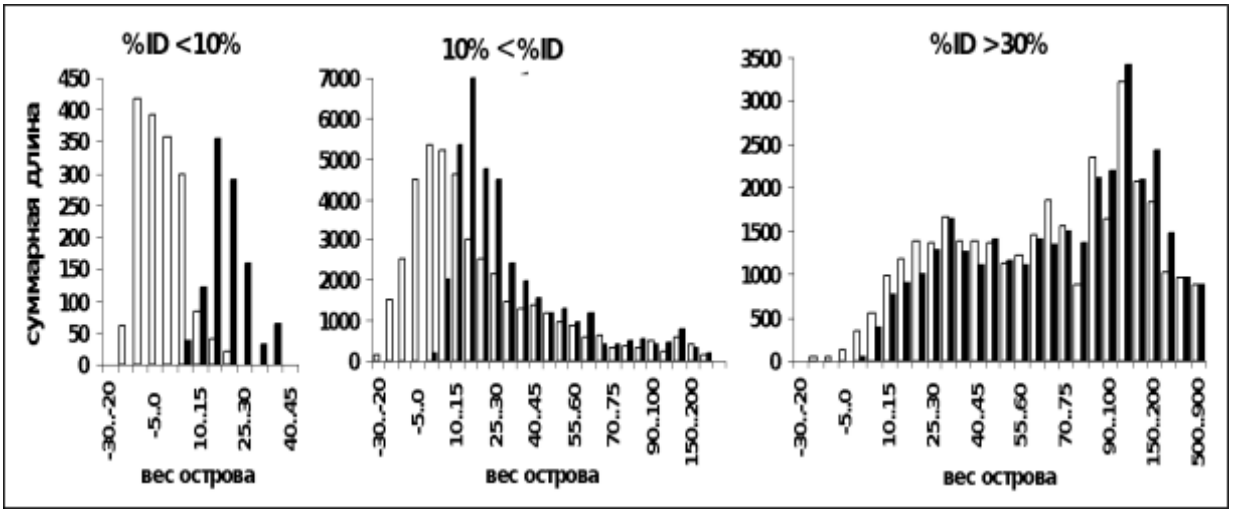


Рисунок 3.1.7. Более детальное представление данных рис. 3.1.6а. Гистограммы суммарных длин островов с весом в пределах указанных по оси X диапазонов отдельно для каждой их 3х областей *%ID*. Данные по эталонным островам показаны белым, SW – черным. Выравнивания SW получены с «линейными» параметрами».

С увеличением степени сходства сравниваемых белков (см. рис. 3.1.6) различие в весе эталонных и построенных островов уменьшается. Однако даже для высокого уровня сходства белков (*ID* > 30%) встречаются эталонные острова отрицательного веса.

Острова малого веса присутствуют в эталонных выравниваниях и при использовании весовых матриц сопоставлений, отличных от матрицы Gonnet250. В Таблице 3.1.1 представлены количество и суммарная длина отрицательно весящих эталонных островов для разных матриц замен. Первые две колонки этой таблицы относятся к часто используемым матрицам замен Blosum62 и Gonnet250. Данные третьей колонки получены при использовании матриц замен, вычисленных по множественным выравниваниям базы данных BaliBase в соответствии с формулами указанными в [146]. Эти матрицы были вычислены отдельно для трех категорий выравниваний: *ID* ≤ 25%, 25%< *ID* ≤ 50% и *ID* больше 50%. Вес островов для колонки «BaliBase» вычислялся по

матрице, соответствующей уровню гомологии сравниваемых белков в эталонном выравнивании.

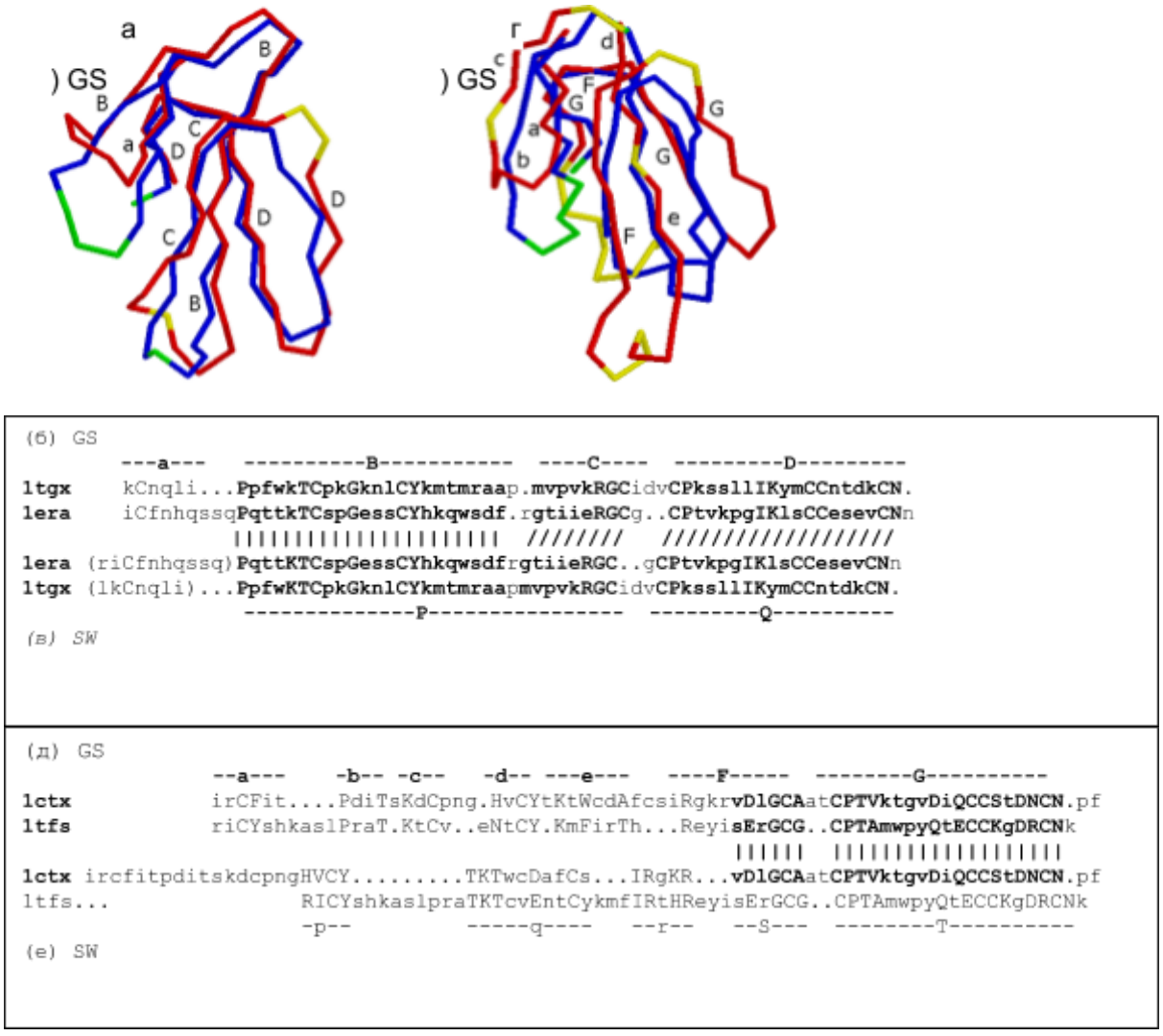


Рис. 3.1.8**.** Сравнение эталонных структурных выравниваний (GS) и выравниваний Смита-Уотермана (SW). (а), (г) – структурные выравнивания (GS) виде совмещения структур в пространстве; (б), (д) – структурные выравнивания (GS) в буквенном представлении; (в), (е) – выравнивания последовательностей соответствующих белков, построенные методом Смита-Уотермана (SW). Выравнивания строились для 2-х пар белков: (а-в) кардиотоксин (код PDB [55]: 1tgx) с эрабутоксином b (код PDB: 1era); (г-е) кобратоксин (код PDB: 1ctx) с токсином FS2 (код PDB: 1tfs).

Таблица 3.1.1.

Матрица замен	Blosum62	Gonnet250	BaliBase

Количество островов отрицательного веса	1374	936	799
Суммарная длина островов отрицательного веса	16833	10600	8674

Количество и суммарная длина островов эталонных выравниваний, имеющих отрицательный вес, для разных матриц замен.

Даже для матрицы замен, специально вычисленной по исследуемому банку данных с учетом уровня гомологии сравниваемых белков, количество отрицательно весящих эталонных островов весьма велико. Таблица 3.1.1 показывает, что далеко не всегда можно применять единую матрицу замен ко всем локусам выравнивания. Уровень гомологии зачастую распределен неравномерно вдоль выравнивания, т.е. существуют участки большего и меньшего сходства, и они требуют индивидуального подхода.

Подведем итоги. Алгоритм SW не может восстановить остров с невысоким весом по двум причинам: (1) если штраф за открытие делеции относительно большой, то алгоритму не выгодно вставлять новую делецию (или делеции), чтобы построить этот остров (см. остров «-a-» на Рис. 3.1.7); (2) если штрафы низкие – найдется альтернативный путь более высокого веса по чисто статистическим причинам (острова выравниваний на Рис. 3.1.7 (д) и (е)). Со стандартными параметрами первый случай типичен для пар белков с высокой степенью гомологии, второй – для дальних гомологов.

3.2. Выравнивания аминокислотных последовательностей с учетом вторичной структуры.

3.2.1. Постановка задачи.

Как мы показали выше, даже наиболее точный из используемых в настоящее время алгоритмов, SW-алгоритм, не способен надежно воссоздать выравнивание пространственных структур, если идентичность последовательностей ниже 30%. В этом разделе мы укажем путь, позволяющий преодолеть эти ограничения. Он состоит в явном учете физико-химических свойств сравниваемых белков, точнее - сведений о их

вторичной структуре. При этом примерно с равным успехом может использоваться как экспериментально полученная вторичная структура, так и теоретически предсказанная. Таким образом, метод применим для выравнивания белков с неизвестной пространственной структурой.

Использование именно вторичной структуры как одного их возможных типов данных о физико-химических свойствах белков, объясняется несколькими причинами. Во-первых, вторичная структура (как часть пространственной структуры белка) существенно более консервативна, чем его первичная структура. Во-вторых, для теоретического предсказания вторичной структуры разработаны весьма эффективные методы. Идея использования данных о вторичной структуре при выравнивании аминокислотных последовательностей рассматривается в работах по биоинформатике с конца 80-х годов XX века. При этом, авторы работ основное внимание уделяли использованию этих выравниваний при установлении дальних гомологий между белками, сведения же о качестве алгоритмически полученных выравниваний белков носили лишь отрывочный характер (см. подробнее гл. 1).

3.2.2. Использованные данные и методы.

3.2.2.1. Вторичная структура: экспериментальные данные.

Источником данных об экспериментально определенных вторичных структурах белков служила база данных DSSP [165], полученная при обработке пространственных координат белков. Восемь типов вторичной структуры, используемые в DSSP, были приведены к трем общепринятым типам H (спираль), E (бета-участок), L (петля) по следующей схеме: (H,G,I) -> H; (E,B) -> E; (T,S,пробел) -> L, где H – альфа-спираль; G – 3/10-спираль в DSSP; I – π-спираль в DSSP; E – элемент бета-структуры; B – одиночный бета-мостик в DSSP; T - резкий поворот цепи в DSSP; S – поворот цепи, не стабилизированный водородными связями в DSSP; пробел – структура не определена.

3.2.2.2. Вторичная структура: предсказания.

Для предсказания вторичной структуры использовалась программа PSIPRED [162] в двух режимах: совместного предсказания структуры для

группы гомологичных белков (“full version”) и предсказание структуры только по аминокислотной последовательности (“single version”). При этом точность предсказания для использованного набора белков (см. ниже) составила 82 и 65%, соответственно, что согласуется с результатами, приведенными на сервере EVA (http://cubic.bioc.columbia.edu/eva/). Для каждой из этих версий использовалось два способа представления предсказанной вторичной структуры. В первом случае («тип_структуры»), каждому остатку аминокислотной последовательности приписывается определенный символ вторичной структуры (H – спираль; E – бета-структура; L – петля). Во втором («вероятность_структуры»), каждому остатку приписываются вероятности принадлежности остатка к каждому из трех типов вторичной структуры, которые также рассчитываются программой PSIPRED. Ниже перечислены аббревиатуры используемых методов:Exp – экспериментально известная структура по DSSP; PSI_S – предсказание структуры по гомологии (“full version” PSIPRED); представление предсказания – «тип_структуры»; PSI_% – предсказание структуры по гомологии (“full version” PSIPRED); представление предсказания – «вероятность_структуры»; SIN_S – предсказание структуры только по аминокислотной последовательности (“single version” PSIPRED); представление предсказания – «тип_структуры»; SIN_% – предсказание структуры только по аминокислотной последовательности (“single version” PSIPRED); представление предсказания – «вероятность_структуры».

*3.2.2.3. Эталонные структурные выравнивания***.**

В качестве эталонных выравниваний, мы использовали те же пространственные выравнивания белков из базы данных BAliBASE [46], что и в разделе 3.1. BAliBASE содержит множественные выравнивания белковых доменов, построенные на основе выравнивания пространственных структур и проверенные экспертами. Использование BAliBASE в качестве тестового множества объясняется постановкой задачи – улучшением качества выравнивания заведомо гомологичных белков. Полученная база эталонных парных выравниваний для корректного сравнения методов была разделена на тренировочный и тестовый наборы. В тренировочный набор вошли все четные (в нашем списке) эталонные пары белков, в тестовый – все нечетные.

3.2.2.4. Оценка качества выравнивания.

Для сравнения алгоритмических и эталонных выравниваний использовались те же меры (точность и достоверность), что и в разделе 3.1.

3.2.3. Алгоритм выравнивания с использованием сведений о вторичной структуре.

3.2.3.1 Общее описание.

Наш алгоритм является модификацией алгоритма Смита-Уотермана. Единственное отличие состоит в том, что при сопоставлении i-го аминокислотного остатка одной последовательности и j-го остатка другой добавляется бонус – коэффициент, определяющий вклад вторичной структуры (SBON), умноженный на величину сходства вторичной структуры. Полный вид рекурсивных уравнений приведен ниже:

$$W(i,j)=\max\begin{cases} W(i-1,j-1)+M(a_i,b_j)+SBON\times Q(i,j) \\ W(i-1,j)-GOP-GEP \\ WA(i-1,j)-GEP \\ W(i,j-1)-GOP-GEP \\ WB(i,j-1)-GEP \\ 0 \end{cases},$$

$$WA(i,j)=\max\begin{cases} W(i-1,j)-GOP-GEP \\ WA(i-1,j)-GEP \end{cases},$$

$$WB(i,j)=\max\begin{cases} W(i,j-1)-GOP-GEP \\ WB(i,j-1)-GEP \end{cases}.$$

Здесь: a и b – первая и вторая сравниваемые белковые цепи; a_i и b_j – i-й и j-й аминокислотные остатки в цепях a и b; W(i,j) – вес наилучшего выравнивания начального фрагмента a[1...i], включающего остатки 1 i последовательности a, и начального фрагмента b[1...j] , включающего остатки 1 j последовательности b; WA(i,j) – вес наилучшего выравнивания фрагментов a[1..i] и b[1..j], в котором последний остаток i в a[1..i] не сопоставлен никакому остатку в b[1..j]; WB(i,j) – вес наилучшего выравнивания фрагментов a[1..i] и b[1..j], в котором последний остаток j в b[1..i] не сопоставлен никакому остатку в a[1..j]; $M(a_i, b_j)$ – вес сопоставления

аминокислотных остатков по выбранной матрице замен (в данной работе используется BLOSUM62 [146]); SBON – коэффициент, определяющий вклад вторичной структуры в выравнивание, Q(i,j) – функция, определяющая сходство вторичной структуры аминокислотных остатков i и j в цепях a и b:

если аминокислотным остаткам i и j приписаны «типы структуры» $T_a(i)$, и $T_b(j)$, то

$$Q(i,j) = \begin{cases} 1, \text{если } \{[\, T_a(i) = T_b(j) = \text{'H'}\,] \text{ ИЛИ } [\, T_a(i) = T_b(j) = \text{'E'}\,]\} \\ 0 \text{ - в остальных случаях} \end{cases};$$

если аминокислотным остаткам i и j приписаны «вероятности структур» $Hp_a(i)$, $Ep_a(i)$, $Lp_a(i)$ и $Hp_b(j)$, $Ep_b(j)$, $Lp_b(j)$, то

$$Q(i,j) = Hp_a(i) \times Hp_b(j) + Ep_a(i) \times Ep_b(j) .$$

«Структурный» вес выравнивания, при данных штрафах за делеции, отличается от веса Смита-Уотермана только тем, что использует вес $M(a_i,b_j)$+SBON*Q(i,j) там, где алгоритм Смита-Уотермана использует вес сопоставления по матрице замен $M(a_i, b_j)$.

3.2.3.2. Другие программы выравнивания, использующие сведения о вторичной структуре

Идейно наиболее близким к нашему методу является метод WFMFL, представленный в работе Валлквиста, Фукуниши, Мерфи, Фадела и Леви [316]. Основные отличия нашего метода от WFMFL состоят в том, что: а) WFMFL использует специальную матрицу весов сходства вторичных структур, полученную из анализа пространственных выравниваний, в то время как в предлагаемом методе STRUSWER учет вторичной структуры определяется всего одним параметром (бонусом за сопоставление вторичной структуры); б) STRUSWER использует предсказание вторичной структуры не в абсолютной, а в относительной форме (для каждого остатка указывается его «склонность» к каждому из возможных видов структуры). Ниже мы приводим сравнение результатов нашей программы STRUSWER и программы, описанной в работе [316].

3.2.3.4. Оптимизация параметров программ.

Мы строили девять выравниваний для каждой пары белков из тренировочного набора и каждого набора параметров:

1) SW-выравнивание (без вторичной структуры);

2) STRUSWER_SIN_S: STRUSWER-выравнивание с вторичной структурой, предсказанной программой PSIPRRED в режиме “single” с выбором мажоритарной вторичной структуры;

3) STRUSWER_SIN_%: STRUSWER-выравнивание с вторичной структурой, предсказанной программой PSIPRRED в режиме “single” с использованием вероятностей вторичных структур;

4) WFMFL_SIN: WFMFL-выравнивание с вторичной структурой, предсказанной программой PSIPRRED в режиме “single” с выбором мажоритарной вторичной структурой;

5) STRUSWER_PSI_S: STRUSWER-выравнивание с вторичной структурой, предсказанной программой PSIPRRED в режиме “full” (см. «Материалы и методы») с выбором мажоритарной вторичной структуры;

6) STRUSWER_PSI_%: STRUSWER-выравнивание с вторичной структурой, предсказанной программой PSIPRRED в режиме “full” с использованием вероятностей вторичных структур;

7) WFMFL_PSI: WFMFL-выравнивание с вторичной структурой, предсказанной программой PSIPRRED в режиме “full” с мажоритарной системой представления структуры;

8) STRUSWER_Exp: STRUSWER-выравнивание с использованием экспериментально определенной вторичной структуры;

9) WFMFL_Exp: WFMFL-выравнивание с использованием экспериментально определенной вторичной структуры.

Полученное каждым из методов алгоритмическое выравнивание сравнивали с эталонным выравниванием с помощью описанных выше мер точности и достоверности. После получения результатов для всех пар белков, входящих в тренировочный набор (включающий лишь четные пары белков), величины точности и достоверности усредняли по всем этим парам, с получением <Acc> = <I/G> и <Conf> = <I/A>, соответственно. Оптимизацию параметров производили путем перебора, следующим образом: SBON менялся от 1 до 30; GOP от 4 до 20; GEP от 1 до 7. Все перечисленные параметры меняли с шагом 1. Таким образом, тренировочный набор обрабатывался 30 х

17 x 7 = 3570 раз. Параметры, при которых точность или достоверность, в зависимости от цели оптимизации, достигали максимального значения, использовали для тестирования каждого метода на тестовом наборе. Следует отметить, что все три параметра (SBON; GOP, GEP) оптимизируются только в программе STRUSWER (методы 2-3, 4-5, 8). В остальных случаях оптимизируются лишь GOP и GEP, так как метод WFMFL использует вместо параметра SBON фиксированную матрицу сопоставления элементов вторичной структуры (см. табл. 3.2.1), а метод SW основан только на сравнении аминокислотных последовательностей.

3.2.4. Тестирование методов.

Каждый метод тестировался на тестовом наборе (включающем лишь нечетные пары белков) с параметрами, найденными при оптимизации программ. Кроме значений точности и достоверности по всему набору, также отдельно вычислялись значения точности и достоверности для низкогомологичных (ID < 30%) пар.

Таблица 3.2.1.

	H	E	L
H	2	-15	-4
E	-15	4	-4
L	-4	-4	2

Матрица сопоставления вторичной структуры используемая в алгоритме WFMFL. Обозначения: H –α-спираль; E – β-лист; L – петля.

В табл. 3.2.2 и 3.2.3 приведены данные о точности Acc и достоверности Conf, а также параметры SBON, GOP, GEP при которых достигается максимум точности (табл. 3.2.2) или достоверности (табл. 3.2.3), соответственно. Табл. 3.2.4, в дополнение к табл. 3.2.2 и 3.2.3 содержит сведения о точности и достоверности методов WFMFL и SW, полученных при стандартных для этих методов параметрах.

Данные во всех таблицах показаны как для всей тестовой выборки, так и отдельно для пар белков, входящих в «серую зону», т.е. для пар белков, гомологичность которых составляет менее 30%. Хотя отдельную оптимизацию для низкогомологичной части не производили, учет таких

белковых пар отдельно от остальной базы полезен для оценки работы методов при важном для практического применения случае низкой гомологии сравниваемых белков. Оптимизация проводилась отдельно для трех способов разметки вторичной структуры: а) путем предсказания вторичной структуры, исходя только из самой последовательности; б) путем предсказания вторичной структуры с привлечением данных о гомологичных белках; в) на основе экспериментальных данных; подробнее, см. п.3.2.3.4.

Таблица 3.2.2.

Метод	bonus	GOP	GEP	Acc	Conf	Acc, ID < 30%	Conf, ID < 30%
SW	-	7	1	0.525	0.585	0.353	0.429
а) предсказание вторичной структуры по последовательности							
STRUSWER_SIN_S	2	10	1	0.578	**0.622**	0.428	**0.482**
STRUSWER_SIN_%	7	8	2	**0.602**	0.618	**0.461**	0.477
WFMFL_SIN	матрица	13	1	0.399	0.488	0.263	0.346
б) предсказание вторичной структуры с привлечением данных о гомологичных белках							
STRUSWER_PSI_S	8	9	1	0.659	0.683	0.546	0.573
STRUSWER_PSI_%	17	6	2	**0.683**	**0.695**	**0.579**	**0.589**
WFMFL_PSI	матрица	16	1	0.631	0.672	0.503	0.56
в) экспериментально известная структура							
STRUSWER_EXP	8	10	1	**0.677**	**0.7**	**0.577**	0.601
WFMFL_EXP	матрица	15	1	0.638	0.698	0.527	**0.602**

Точность (Acc) и достоверность (Conf) различных методов выравнивания при тестировании на тестовой выборке. Параметры (bonus, GOP, GEP) были подобраны на тренировочном наборе для получения максимальной точности (Acc) каждым из методов. Представлены данные как для всей выборки (288 пар белков) так и для «серой зоны» (белки с гомологичностью ниже 30%, 182 пары). Обозначения методов – см. раздел «Вторичная структура: предсказания». Матрица, использовавшаяся в методах WFMFL описана в разделе «Алгоритм выравнивания с использованием сведений о вторичной структуре».

Наилучшие результаты, как по точности (табл. 3.2.2), так и по достоверности (табл. 3.2.3), показали методы, основанные на экспериментально известной вторичной структуре (в) и методы, использующие предсказание вторичной структуры с привлечением данных о

таковой в гомологичных белках (б). Однако эти результаты представляют, скорее, методический интерес. Как правило, экспериментально определенная вторичная структура предполагает наличие известной пространственной структуры, и тогда уж имеет смысл воспользоваться одной из программ выравнивания белков по их пространственной структуре. Вместе с тем, тесты на экспериментально известной вторичной структуре показывают примерный предел, которого можно достичь в данном методе, используя вторичную структуру совместно с последовательностью. Во-вторых, привлечение гомологов и их вторичных структур для выравнивания пары белков, пусть и в форме предсказаний вторичной структуры, противоречит смыслу парного выравнивания. Результаты выравнивания двух белков с использованием таких «предсказаний по гомологии» следует, скорее, сравнивать с результатами множественного выравниваниями использованной группы белков.

Таблица 3.2.3.

Метод	bonus	GOP	GEP	Acc	Conf	Acc, ID < 30%	Conf, ID < 30%
SW	-	20	6	0.380	0.706	0.189	0.607
а) предсказание вторичной структуры по последовательности							
STRUSWER_SIN_S	1	15	7	0.433	**0.707**	0.246	**0.620**
STRUSWER_SIN_%	1	10	6	**0.458**	0.700	**0.262**	0.596
WFMFL_SIN	матрица	19	7	0.314	0.646	0.158	0.535
б) предсказание вторичной структуры с привлечением данных о гомологичных белках							
STRUSWER_PSI_S	1	17	6	0.468	0.715	0.286	0.630
STRUSWER_PSI_%	1	14	6	0.465	**0.717**	0.282	**0.631**
WFMFL_PSI	матрица	12	4	**0.606**	0.694	**0.467**	0.596
в) экспериментально известная структура							
STRUSWER_EXP	1	13	6	0.483	0.710	0.303	0.615
WFMFL_EXP	матрица	15	7	**0.553**	**0.748**	**0.400**	**0.676**

Точность (Acc) и достоверность (Conf) различных методов выравнивания при тестировании на тестовой выборке. Параметры (bonus, GOP, GEP) были подобраны на тренировочном наборе для получения максимальной достоверности (Conf) каждым из методов. Представлены данные как для всей выборки (288 пар белков) так и для «серой зоны» (белки с гомологичностью ниже 30%, 182 пары). Обозначения методов – см. раздел «Вторичная структура: предсказания». Матрица,

использовавшаяся в методах WFMFL, описана в разделе «Алгоритм выравнивания с использованием сведений о вторичной структуре».

Таким образом, в качестве основного метода предсказания вторичной структуры мы рассматриваем метод, делающий предсказание вторичной структуры по отдельной аминокислотной последовательности, в данной работе – программой PSIPREDsingle. Обладая меньшей, чем полный вариант PSIPREDfull, точностью предсказаний, он, тем не менее, имеет ряд преимуществ. Первое преимущество состоит в том, что PSIPREDsingle не основан на поиске гомологов, и поэтому STRUSWER в соответствующих режимах (STRUSWER_SIN_S и STRUSWER_SIN_%) можно использовать в случаях, когда произвести поиск гомологов по тем или иным причинам невозможно. Второе преимущество вытекает из первого, и заключается в существенно меньших затратах времени, необходимого для предсказания вторичной структуры. Подобный факт может оказаться решающим в больших вычислительных проектах. Выравнивания, сделанные алгоритмом STRUSWER_SIN, с использованием вторичной структуры, предсказанной по аминокислотной последовательности, превосходят аналогичные выравнивания, полученные алгоритмом SW и алгоритмом WFMFL_SIN, как по точности (табл. 3.2.2), так и по достоверности (табл. 3.2.3). При этом метод WFMFL_SIN показал наименьшую точность, даже ниже, чем алгоритм Смита-Уотермана. Это может быть вызвано тем, что в методе WFMFL для сравнения вторичных структур используется матрица, построенная на основе экспериментальных данных, и поэтому она более чувствительна к качеству предсказания.

Методы WFMFL_EXP и WFMFL_PSI, хотя и показали выигрыш в точности по сравнению с алгоритмом SW (на 0.113 и 0.106 соответственно), но уступили методам STRUSWER_EXP и STRUSWER_PSI. Любопытно, что при использовании вторичной структуры, предсказанной по гомологии, точность выравнивания становится сопоставимой с точностью, полученной на экспериментально известной структуре, и даже превосходит ее. Так, метод STRUSWER_PSI_%, использующий «вероятностное» представление вторичной структуры, является абсолютным лидером по точности среди всех

методов. Если сопоставить методы, использующие состояния вторичной структуры, и методы, использующие вероятности вторичной структуры, то «вероятностные» методы имеют примерно на 2% большую точность. При оптимизации по достоверности (Conf) метод WFMFL_EXP превосходит метод STRUSWER_Exp, хотя метод WFMFL_PSI уступает методам STRUSWER_PSI_S и STRUSWER_PSI_%, а метод WFMFL_SIN_S уступает методам STRUSWER_SIN_S и STRUSWER_SIN_%. Все указанные соотношения остаются верными, если мы ограничимся только слабогомологичными парами белков. При этом относительный выигрыш от использования вторичной структуры существенно возрастает (особенно при использовании экспериментально определенной структуры, хотя последний случай, вряд ли представляет практический интерес).

Данные в табл. 3.2.4 показывают, что при стандартных параметрах, программы WFMFL и SW работают не намного хуже. Однако мы провели оптимизацию их параметров, чтобы сделать сравнение корректным.

3.2.5. Выводы.

Использование вторичной структуры позволяет существенно повысить качество выравнивания аминокислотных последовательностей. При этом примерно с равным успехом может использоваться как экспериментально полученная вторичная структура, так и теоретически предсказанная с помощью сервера PSIPRED.

Таблица 3.2.4.

Метод	bonus	GOP	GEP	Acc	Conf	Acc, ID < 30%	Conf, ID < 30%
SW	-	7	1	0.525	0.585	0.353	0.429
WFMFL_SIN	матрица	12	2	0.386	0.517	0.234	0.381
WFMFL_PSI	матрица	12	2	0.62	0.664	0.49	0.551
WFMFL_EXP	матрица	12	2	**0.632**	**0.697**	**0.52**	**0.603**

Точность и достоверность методов WFMFL и SW при стандартных для каждого из методов параметрах. Тестовый набор такой же, как и в первых двух таблицах. Обозначения методов – см. раздел «Вторичная структура: предсказания». Матрица, использовавшаяся в методах WFMFL описана в разделе «Алгоритм выравнивания с использованием сведений о вторичной структуре».

Таким образом, метод применим при выравнивании белков с неизвестной пространственной структурой. Метод STRUSWER превосходит близкий по подходу метод WFMFL по качеству выравниваний, прежде всего – по точности выравниваний. Преимущество по достоверности выравниваний не столь значительно. Однако вопрос о преобразовании этих преимуществ в лучшее качество поиска по базе данных требует отдельного изучения.

3.3. Выравнивание последовательностей РНК с заданной вторичной структурой

3.3.1. Постановка задачи.

В этом разделе мы продолжаем тему построения оптимального выравнивания биологических последовательностей, обогащенных сведениями о структуре соответствующих макромолекул, на примере молекул РНК. Важное отличие РНК от белков состоит в том, вторичная структура РНК описывается не словом, а деревом [45].

Определение 3.3.1.

Пусть ***S*** **-** последовательность длины ***L*** в РНК-алфавите {A, C, G, U}. Вторичная структура РНК (или просто структура) над ***S*** - это пара ***<S, P>,*** где ***P*** – это такое множество пар целых чисел, что:

1) $\langle i, j\rangle \in P \Rightarrow 1 \le i < j \le L;$

2) $\langle i_1, j_1\rangle, \langle i_2, j_2\rangle \in P \Rightarrow (i_1 \neq i_2) \ \&\ (j_1 \neq j_2) \ \&\ (i_1 \neq j_2) \ \&\ (j_1 \neq i_2)$

Структура ***<S, P>*** не содержит псевдоузлов, если дополнительно выполнено

3) $(\langle i_1, j_1\rangle, \langle i_2, j_2\rangle \in P) \ \&\ (i_1 < i_2) \Rightarrow ((j_1 < i_2) \ \textit{ИЛИ}\ (j_1 > j_2)$

Говоря неформально, (1) *P* – это множество спариваний между нуклеотидами; $\langle i, j\rangle \in P$ означает, что спарены нуклеотиды в позициях *i* и *j*; (2) каждый нуклеотид принадлежит не более, чем одной паре и (3) пары $\langle i_1, j_1\rangle, \langle i_2, j_2\rangle \in P$ не образуют псевдоузлов, т.е. отрезки $[i_1, j_1]$ и $[i_2, j_2]$ либо не пересекаются, либо один их них есть часть другого. Ниже все рассматриваемые структуры РНК будут структурами без псевдоузлов (см. п. 1.2.2).

Спаривания в таких структурах образуют правильную скобочную структуру, которую естественно представлять в виде дерева. Это наблюдение ведет к использованию алгоритмов выравнивания деревьев (см., например, [334], [176]) для построения алгоритмов выравнивания последовательностей РНК, обогащенных сведениями об их вторичной структуре (см. [107]). В то же время, как и в случае сравнения «обычных» биологических последовательностей, необходимо выбрать функцию весов удаления фрагментов последовательности (см. раздел 1.1 и главу 2).

В этом разделе представлен алгоритм выравнивания последовательностей РНК с известными вторичными структурами относительно аффинных весовых функций удаления фрагментов, т.е. функций вида $\phi(l) = g + d \cdot l$, где l – длина удаляемого фрагмента, g и d - коэффициенты (см. раздел 3.1), при этом, как правило, g существенно больше, чем d, см. [27].

3.3.2. Определения и формальная постановка задачи.

3.3.2.1.Выравнивания структур

Определение 3.3.2. Пусть $\langle S_1, P_1 \rangle$ and $\langle S_2, P_2 \rangle$ - две структуры РНК; L_1, L_2 длины последовательностей S_1 и S_2 соответственно. Выравниванием структур $\langle S_1, P_1 \rangle$ и $\langle S_2, P_2 \rangle$ называется произвольное выравнивание последовательностей S_1 и S_2.

Весовая система для выравнивания структур РНК – это пятерка $\langle M, g, d, b, c \rangle$, где M – весовая матрица замен; g и d – это коэффициенты аффинной весовой функции удалений фрагментов; b – бонус за одновременное сопоставление двух пар нуклеотидов (сопоставление спариваний), c – штраф за спаривание в одной из структур, не сопоставленное никакому спариванию в другой структуре.

Пусть задана весовая система $\langle M, g, d, b, c \rangle$,, $\langle S_1, P_1 \rangle$ и $\langle S_2, P_2 \rangle$ структуры и A – их выравнивание с набором склеек $= \{\langle p_k, q_k \rangle \mid k = 1,\ldots, N\}$. Пусть m – общее число удаленных фрагментов в A; l – суммарная длина этих фрагментов, t - количество сопоставлений спариваний; $k = |P_1| + |P_2| - 2k$ – количество спариваний в P_1 and P_2; которые не сопоставлены в A спариваниям в другой структуре. Тогда структурным весом выравнивания A называется величина:

$$\mathbf{Score(A) = \Sigma_{k=1..n} M(}S_1[p_k], S_2[q_k]) - g \cdot m - d \cdot l + b \cdot t - c \cdot k$$

Это определение соответствует определению веса глобального выравнивания последовательностей с аффинными весовыми функциями удаления фрагментов, к которому добавлены бонусы за сопоставление спариваний и штрафы за «потерянные» (не сопоставленные ничему) спаривания.

3.3.2.2. Деревья РНК.

Определение 3.3.3. *РНК-дерево* – это такое корневое упорядоченное дерево *T*, что

(1) листья помечены буквами РНК-алфавита {A, C, G, U};

(2) каждая внутренняя вершина имеет по меньшей мере двух сыновей; причем самый левый и самый правый из этих сыновей – листья (они называются *главными* сыновьями).

Здесь подразумевается обычное представление дерева на плоскости, где корень расположен внизу, а листья – вверху. Таким образом, порядок на ребрах, исходящих из одной вершины, - это порядок слева направо. Этот порядок индуцирует порядок на листьях, а также порядок на множестве всех вершин дерева, соответствующие левому обходу [24]. Имея это ввиду, мы будем считать, что листья перенумерованы: 1, 2, …. , будем говорить, что один лист находится левее (правее) другого и т.п.

Листья ***u*** и ***v*** в дереве ***F*** образуют *пару (*являются *спаренными)* если эти листья являются соответственно самым левым и самым правым сыновьями своего отца. Говоря неформально, листья РНК-дерева соответствуют буквам последовательности РНК, а каждая внутренняя вершина - спариванию своих главных сыновей.

РНК-лесом $\boldsymbol{F = <T_1, \ldots, T_n>}$ называется упорядоченное множество РНК-деревьев.

Если не оговорено противное, в этом разделе под деревом мы понимаем РНК-дерево, а под лесом – РНК-лес. Порядок на деревьях внутри леса и порядок на листьях внутри каждого дерева индуцируют порядок на листьях внутри леса; будем считать листья перенумерованными:1, 2, … .

Определение 3.3.4. Пусть ***F*** – лес. Через ***String*(*F*)** будет обозначаться символьная последовательность, в которой ***i*-**й символ совпадает с пометкой на ***i*-**м листе дерева. Через ***Pairing(F)*** будет обозначаться множество всех таких пар ***<i, j>,*** что ***i*-**й и ***j*-**й лист дерева ***F*** образуют пару (т.е. являются главными сыновьями одной и той же внутренней вершины). Через ***Vertex(F)*** обозначается множество всех вершин леса ***F;*** через ***Leaves(F) -*** множество всех листьев леса ***F.***

Обратно, пусть ***<S, P>*** - РНК-структура без псевдоузлов. Тогда по ней можно однозначно построить РНК-лес ***Forest(S, P).*** Будем говорить, что позиция ***x*** *принадлежит* спариванию ***(i, j),*** если (1) ***i* ≤*x*≤ *j*** и (2) условие ***i' ≤x≤ j'*** не выполнено ни для какого спаривания ***(i', j'),*** удовлетворяющего условию ***i< i' < j'<,j.*** Будем говорить, что спаривание ***(i', j')*** *принадлежит* спариванию ***(i, j),*** если (1) ***i< i' < j'<,j*** и (2) условие ***i"< i' < j'<,j"*** не выполнено ни для какого спаривания ***(i", j"),*** удовлетворяющего условию ***i< i" < j"<,j.***

Определим лес ***Forest(S, P)*** следующим образом. Листья ***Forest(S, P)*** соответствуют позициям последовательности ***S***; а внутренние вершины – спариваниям из ***P***. Из вершины ***(i, j)*** ребро ведет во внутреннюю вершину ***(i', j')*** тогда и только тогда, когда ***(i', j')*** принадлежит ***(i, j)***. Из вершины ***(i, j)*** ребро ведет в лист ***x*** тогда и только тогда, когда ***x*** принадлежит ***(i, j)***. Порядок на множестве ребер, выходящих из вершины ***(i, j)*** определяется естественным образом.

Лемма 3.3.1.

1. Пусть ***F*** – РНК-лес. Тогда ***<String*(*F*), Pairing(F)>** - это РНК-структура без псевдоузлов и . ***Forest(String*(*F*), Pairing(F)) = F.***

2. Пусть ***<S, P>*** - РНК-структура без псевдоузлов. Тогда ***Forest(S, P)*** – РНК-лес, причем ***String*(*Forest(S, P)) = S*** и ***Pairing*(*Forest(S, P)) = P.***

Доказательство – очевидно.

3.3.2.3. Выравнивания РНК-лесов.

Пусть F_1, F_2 - РНК-леса.

Определение 3.3.4. *Выравнивание* РНК-лесов F_1 и F_2 -это взаимно однозначное соответствие $A \subseteq Vertex(F_1) \times Vertex(F_2)$ между некоторыми

подмножествами $Vertex(F_1)$ и $Vertex(F_2)$, для которого выполнены следующие условия:

1) Пусть $(v_1,w_1), (v_2,w_2) \in A$.

(i) Вершина v_1 предшествует вершине v_2 в смысле левого обхода леса F_1 тогда и только тогда, когда вершина w_1 предшествует вершине w_2 в смысле левого обхода леса F_2.

(ii) В F_1 есть путь из вершины v_1 в вершину v_2 тогда и только тогда, когда в F_2 есть путь из вершины w_1 в вершину w_2.

2) Если $(v, w) \in A$, то либо обе вершины v и w – листья, либо обе они – внутренние вершины.

3) Пусть v и w – внутренние вершины; v' и v'' – главные сыновья вершины v; w' и w'' – главные сыновья вершины w, причем $v' < v''$ и $w' < w''$. В этих условиях $(v, w) \in A$ тогда и только тогда, когда $(v',w'), (v'',w'') \in A$.

Первое условие следует [57] и применимо к выравниваниям произвольных упорядоченных лесов. Условия (3) и (4) отражают специфику РНК-лесов.

Пусть A – выравнивание РНК-лесов F_1 и F_2; G_1 – подлес F_1 ;G_2 – подлес F_2. *Ограничением* A на G_1 и G_2 называется выравнивание $A \cap (Vertex(G_1) \times Vertex(G_2))$

Определение 3.3.5. Пусть F_1 and F_2 - РНК-леса и B - выравнивание последовательностей $String(F_1)$ и $String(F_2)$. Выравнивание A лесов F_1 and F_2 называется *расширением* выравнивания B, если ограничение A на $Leaves(F_1)$ и $Leaves(F_2)$ совпадает с выравниванием B.

Лемма 3.3.2.

1. Пусть F_1 и F_2 – РНК-леса и A – выравнивание F_1 и F_2. Тогда ограничение A на $Leaves(F_1)$ и $Leaves(F_2)$ корректно определяет выравнивание последовательностей $String(F_1)$ и $String(F_2)$ и, следовательно, выравнивание РНК-структур $<String(F_1), Pairing(F_1)>$.

2. Пусть F_1 and F_2 - РНК-леса и и B - выравнивание последовательностей $String(F_1)$ и $String(F_2)$. Тогда существует единственное выравнивание лесов F_1 and F_2, которое является расширением выравнивания B.

Доказательство. Утверждение 1 следует из условий 1) и 2) определения 3.3.4. Утверждение 2 следует из условия 3) определения 3.3.4.

Определение 3.3.6. Вершина $v \in Vertex(F_1)$ (соответственно, $w \in Vertex(F_2)$) *удалена* в выравнивании A лесов F_1 и F_2, если A не содержит пары вида (v, w') (соответственно, (v', w)).

В соответствии с леммами 3.3.1 и 3.3.2 задача выравнивания РНК-структур сводится к задаче выравнивания РНК-лесов. В частности, вес выравнивания лесов можно определить, основываясь на лемме 3.3.2.

Определение 3.3.7. Пусть дана весовая система $<M, g, d, b, c>$ (см. Определение 3.3.2.) и A - выравнивание лесов F_1 и F_2,. Тогда весом выравнивания A называется вес индуцированного им выравнивания $Struct(A)$ структур $<String(F_1), Pairing(F_1)>$ и $<String(F_2), Pairing(F_2)>$. Вес выравнивания A будет обозначаться $W(A)$.

Это определение находится в соответствии с определением веса выравнивания деревьев, данного в [57].

3.3.2.4. Формальная постановка задачи.

В соответствии со сказанным выше, проблема построения оптимального глобального выравнивания структур РНК формулируется следующим образом.

Проблема построения оптимального глобального выравнивания структур РНК. Даны РНК-леса F_1 и F_2 и весовая система $<M, g, d, b, c>$. Найти оптимальное, т.е. имеющее наибольший возможный вес, выравнивание лесов F_1 и F_2.

Отметим, что задача выравнивания лесов хорошо изучена. Специфика РНК-лесов заключается в (1) в различной интерпретации листьев и внутренних вершин (нуклеотиды и спаривания), в частности, в понятии главных сыновей и (2) использовании аффинной функции штрафов за удаление символов.

3.3.3. Описание алгоритма.

3.3.3.1. Активные деревья

Алгоритм следует парадигме «левый-правый», предложенной в [176], см. также [98]. Эта парадигма состоит в следующем. При выравнивании лесов

F_1 и F_2 мы в каждый момент выбираем т.н. *активные* деревья $T_1 \in F_1$ и $T_2 \in F_2$., активные деревья являются либо одновременно крайними левыми, либо одновременно крайними правыми деревьями в F_1 и F_2. Элементарный шаг выравнивания состоит в сопоставлении корней активных деревьев, либо в удалении одного из корней. После этого задача сводится к выравниванию лесов F'_1 и F'_2 меньшего размера. Для описания алгоритма, следующего описанной парадигме, следует уточнить: (1) алгоритм выбора активных деревьев; (2) критерий выбора операции с корнями активных деревьев. Далее алгоритм выравнивания может быть построен, как алгоритм динамического программирования (см. [121]).

Для выбора активных деревьев будем использовать следующий алгоритм, являющийся модификацией алгоритма [176].

Алгоритм выбора активного дерева. Если оба леса F_1 и F_2 состоят из одного дерева, то выбор очевиден. Пусть лес F_1 содержит более одного дерева; $LEFT(F_1)$ и $RIGHT(F_1)$ – соответственно самое левое и самое правое деревья леса F_1. В качестве активного дерева в F_1 берем то из них, которое содержит меньшее количество вершин. В качестве активного дерева в F_2 берем крайнее дерево с той же стороны. Если лес F_1 состоит из одного дерева, а лес F_2 - более, чем из одного дерева, то поступаем аналогично.

Конец алгоритма.

3.3.3.2. Идея алгоритма выравнивания.

Лемма 3.3.3. [57] Пусть F_1 и F_2 - РНК-леса; T_1 и T_2 - их активные деревья, R_1 и R_2 - корни соответственно T_1 и T_2. Пусть A - выравнивание F_1 и F_2. Тогда выполнено ровно одно из двух следующих утверждений:

1) R_1 и R_2 сопоставлены в A (это возможно только, если обе вершины R_1 и R_2 - листья либо ни одна из них не является листом);

2) в выравнивании A хотя бы одна из вершин R_1 и R_2 удалена.

Доказательство следует из условий (1) и (2) определения 3.3.4.

Замечание. Согласно (4) определению 3.3.4, если R_1 и R_2 – внутренние вершины, которые не сопоставлены в выравнивании A, то хотя бы один из главных сыновей R_1 не сопоставлен в A с соответствующим сыном R_2. По техническим причинам при описании алгоритма выравнивания РНК-лесов мы

ослабим это условие и будем также рассматривать выравнивания, в которых обе пары главных сыновей некоторых внутренних вершин сравниваемых лесов сопоставлены друг другу, а сами эти вершины – нет. Очевидно, тем не менее, такое выравнивание не может быть оптимальным (добавлением сопоставления несопоставленных отцов можно получить выравнивание большего веса) и расширение множества допустимых выравниваний не повлияет на результат работы алгоритма. Конец замечания.

Определение 3.3.8. Пусть ***T*** – РНК-дерево. *Редуцированный* лес ***Rd(T)*** дерева ***T*** – это лес, полученный из ***T*** удалением корня и выходящих из него ребер. Пусть далее корень дерева ***T*** не является листом (и, следовательно, лес ***Rd(T)*** непуст). *Сильно редуцированный* лес ***Rds(T)*** – это лес, полученный из ***T*** удалением корня, выходящих из него вершин и главных сыновей.

Определение 3.3.9. Пусть ***F*** – РНК-лес и ***T*** – его активное дерево. *Редуцированный* лес ***Rd(F, T)*** – это лес, полученный из ***F*** заменой дерева ***T*** на лес ***Rd(T)***. *Сильно редуцированный* лес ***Rds(F, T)*** – это лес, полученный ***F*** заменой дерева ***T*** на лес ***Rds(T)***. *Обрезанный* лес ***Tr(F, T)*** – это лес, полученный из ***F*** удалением дерева ***T.***

Определение 3.3.10. Пусть F_1 и F_2 - РНК-леса, T_1 и T_2 – их активные деревья. *Наследниками* пары лесов $<F_1, F_2>$ будем называть следующие пары лесов::

1) $< Rd(F_1, T_1), F_2 >$;

2) $< F_1, Rd(F_2, T_2) >$;

3) $< Tr(F_1, T_1), Tr(F_2, T_2) >$ (если оба дерева T_1 и T_2 состоят из одного листа);

4) $< Rds(F_1, T_1), Rds(F_2, T_2) >$ (если оба дерева T_1 и T_2 содержат более одного листа);

Говоря неформально, случай 1) соответствует удалению корня T_1; случай 2) соответствует удалению корня T_2; случаи 3) и 4) соответствуют сопоставлению корней T_1 и T_2.

Определение 3.3.11. Пара лесов $<F'_1, F'_2>$ *выводима* из пары лесов $<F_1, F_2>$ (при заданной процедуре выбора активных деревьев), если существует

такая последовательность пар лесов P_0 ,…, P_n , что $P_0 = <F_1, F_2>$; $P_n = <F'_1, F'_2>$ и P_i является наследником P_{i-1} $(i=1,\ldots, n)$.

Предлагаемый алгоритм (подобно [334, 176, 98]) строит оптимальное выравнивание для всех пар лесов, выводимых из исходной пары $<F_1, F_2>$; при этом пары обрабатываются в порядке возрастания их размера, т.е. общего количества вершин. Обработка пары лесов будет занимать фиксированное время, так что общее время работы алгоритма будет пропорционально количеству пар лесов, выводимых из исходной пары $<F_1, F_2>$.

3.3.3.3. Характеристики пар лесов

Ниже будем считать фиксированной весовую систему $<M, g, d, b, c>$.

Подобно выравниваниям последовательностей, для каждой обрабатываемой промежуточной пары лесов $<F_1, F_2>$ нам потребуется вычислить значения нескольких величин, характеризующих множество выравниваний пары $<F_1, F_2>$. Эти величины аналогичны величинам, используемым при вычислении оптимального выравнивания последовательностей относительно аффинных весовых функций [139]. Более точно смысл этих величин объяснен ниже.

Определение 3.3.12. Пусть F – непустой лес, l – количество листьев в F, k – количество внутренних вершин в F. Через $D(F)$ будет обозначаться стоимость полного удаления F , т.е. величина

$$D(F) = -c \cdot k - g - b \cdot l$$

Назовем выравнивание A лесов F_1 и F_2 *нетривиальным*, если оно содержит хотя бы одно сопоставление и тривиальным в противном случае.

Из определения 3.3.2 и замечания после определения 3.3.6 следует, что вес тривиального выравнивания $\tau(F_1, F_2)$ лесов F_1 и F_2 равен $D(F_1) + D(F_2)$.

Определение 3.3.13. Пусть $L, R \subseteq \{1, 2\}$. Будем говорить, что выравнивание A лесов F_1 и F_2 является L, R-*согласованным,* если из $k \in L$ следует, что A содержит удаленный фрагмент на левом конце $String(F_k)$, а из $k \in R$ следует, что A содержит удаленный фрагмент на правом конце $String(F_k)$.

Рекурсивно вычисляемыми величинами в нашем алгоритме являются 16 величин вида $Best(F_1, F_2, L, R)$, где $L, R \subseteq \{1, 2\}$ (см. определение 3.3.15).

Говоря неформально, ***Best(F_1 ,F_2, L, R)*** – это вес оптимального выравнивания лесов F_1 и F_2 в классе ***L, R-***согласованных выравниваний. Понятие делеционного веса (см. определение 3.3.14) соответствует добавлению (там, где это нужно) штрафа за фиктивное удаление на конце. Как и в случае выравнивания последовательностей [139], это позволяет упростить рекуррентные уравнения (см. ниже).

Определение 3.3.14. Пусть $L, R \subseteq \{1, 2\}$ и A - выравнивание лесов F_1 и F_2. Пусть k_L (соответственно, k_R) - количество таких индексов $i \in L$, что при выравнивании A в F_i нет удаления самой левой (правой) буквы или F_i пусто. *Делеционным* весом ***WD(A, L, R)*** выравнивания A при ограничениях L, R называется величина

$$WD(A, L, R) = W(A) - (k_L + k_R) \cdot g$$

Определение 3.3.15. Пусть $<F_1, F_2>$ - пара лесов, $L, R \subseteq \{1, 2\}$. Через ***Best(F_1 ,F_2, L, R)*** обозначается наибольшее возможное значение делеционного веса ***WD(A, L, R)*** для выравниваний A лесов F_1 и F_2.

Очевидно,

$$WD(A, \varnothing, \varnothing) = W(A)$$

и ***Best(F_1 ,F_2, ∅, ∅)*** – это вес оптимального выравнивания лесов F_1 и F_2.

Для каждой пары лесов $<F_1, F_2>$, выводимой из исходной пары сравниваемых весов, мы будем вычислять шестнадцать значений ***Best(F_1 ,F_2, L, R).*** Пары лесов будут рассматриваться в порядке возрастания общего количества вершин в F_1 и F_2 . Ниже приведены рекурсивные уравнения для величин ***Best.*** Их инициализация для случая, когда один из лесов F_1 и F_2, следует из рассмотрения тривиальных выравниваний в п. 3.3.3.3 и определений 3.3.13, 3.3.14.

Мы приводим уравнения для случая, когда активные деревья расположены слева. Уравнения для активных деревьев, расположенных справа, аналогичны. Пусть F_1 и F_2 - РНК-леса; T_1 и T_2 - их активные деревья, R_1 и R_2 - корни соответственно T_1 и T_2; $L, R \subseteq \{1, 2\}$. Будем считать, что леса F_1 и F_2 непусты (см. выше). Рассмотрим отдельно три случая.

А. Пусть T_1 и T_2 содержат более одной вершины, т.е. R_1 и R_2 - внутренние вершины. Пусть $s_л$ ($s_п$) – вес за сопоставление символов-пометок

левых (правых) главных сыновей R_1 и R_2 . У $<F_1 . F_2>$ есть три возможных наследника:

$< Rd(F_1, T_1) , F_2 >$;

$< F_1 , Rd(F_2, T_2) >$;

$< Rds(F_1, T_1) , Rds(F_2, T_2) >$

Первые два случая соответствуют тому, что спаривание R_1 (в первом случае) или R_2 (во втором случае) не выровнены. Третий случай означает, что спаривания R_1 и R_2 выровнены и, следовательно, их главные сыновья, сопоставлены друг другу. Ни один из этих случаев не соответствует удалению символа. Для любых L и R имеем (ниже $k = |L|+|R|$):

$$Best(F_1 ,F_2,L, R) = max\{$$
$$Best(Rd(F_1, T_1) , F_2, L, R) – c;$$
$$Best (F_1 , Rd(F_2, T_2, L, R))> -c;$$
$$Best (Rds(F_1, T_1) , Rds(F_2, T_2) , \varnothing, \varnothing) +b + s_1+ s_1-k•g,$$
$$\} \qquad (3.3.1)$$

Б. Пусть одна из вершин R_1 и R_2 (например, R_1) – лист, а другая (например, R_2) – внутренняя вершина. У $<F_1 . F_2>$ есть два возможных наследника:

$< Rd(F_1, T_1) , F_2 > = < Tr(F_1, T_1) , F_2 >$

$< F_1 , Rd(F_2, T_2) >$;

Первый случай соответствует удалению левой буквы в символьной последовательности $String(F_1)$, второй «разрыву» связи между главными сыновьями вершины R_2. Имеем такие уравнения:

$$Best(F_1 ,F_2, L, R) = max\{$$
$$Best(Rd(F_1, T_1)), F_2, L \cup\{1\}, R) –d;$$
$$Best (F_1 , Rd(F_2, T_2), L, R) – c;$$
$$\} \qquad (3.3.2)$$

В. Пусть обе вершины R_1 и R_2 – листья. У $<F_1 . F_2>$ есть три возможных наследника:

$< Tr(F_1, T_1) , F_2 >$

$< F_1 , Tr(F_2, T_2) >$;

$< Tr(F_1, T_1) , Tr(F_2, T_2) >$;

Первый случай соответствует удалению левой буквы в символьной последовательности ***String(F_1),*** второй случай - удалению левой буквы в символьной последовательности ***String(F_2),*** третий – сопоставлению левых букв в ***String(F_1)*** и ***String(F_2).*** Имеем уравнение:

$$
\begin{aligned}
&Best(F_1, F_2, L, R) = max\{ \\
&\quad Best(Tr(F_1, T_1)), F_2, L \cup\{1\}, R) - d; \\
&\quad Best(F_1, Tr(F_2, T_2), L \cup\{2\}, R) - d; \\
&\quad Best(Tr(F_1, T_1), Tr(F_2, T_2), \varnothing, R) + s - k \bullet g, \\
&\}
\end{aligned}
\qquad (3.3.3)
$$

где $k = |L|+|R|$, ***s*** - вес сопоставления левых букв ***String(F_1)*** и ***String(F_2).***

Лемма 3.3.4. Рекурсивные уравнения п. 3.3.3.3 правильно вычисляют значения величин ***Best.***

Доказательство проводится стандартным для метода динамического программирования способом. Чтобы не перегружать изложение техническими деталями, ограничимся рассмотрением случая Б.

Пусть ***B*** - выравнивание, которое соответствует максимуму в рекурсивном уравнении (3.3.2) и ***A*** – произвольное выравнивание F_1 и F_2. По определению случая Б, корни R_1 и R_2 в выравнивании ***A*** не могут быть сопоставлены друг другу и, следовательно, один из них удален. Пусть в выравнивании ***A*** удален корень R_1. Рассмотрим выравнивание ***A'*** лесов ***Rd(F_1, T_1))*** и F_2, индуцированное выравниванием ***A.*** Очевидно,

$$WD(A, L, R) = WD(A', L \cup\{1\}, R) - d \qquad (3.3.4)$$

(добавление {1} к множеству ***L*** вызвано необходимостью правильно вычислить вес удаления на левом конце ***String(F_1)).***

Из (3.3.4) следует, что

$$
\begin{aligned}
&WD(A, L, R) = WD(A', L \cup\{1\}, R) - d \le \\
&\quad \le Best(Rd(F_1, T_1)), F_2, L \cup\{1\}, R) - d \le \\
&\quad \le WD(B, L, R)
\end{aligned}
$$

Пусть теперь в выравнивании ***A*** удален корень R_2. Рассмотрим выравнивание ***A"*** лесов F_1 и ***Rd(F_2, T_2),*** индуцированное выравниванием ***A.*** Очевидно,

$$WD(A, L, R) = WD(A'', L, R) - c \qquad (3.3.5)$$

Из (3.3.5) следует, что

$$WD(A, L, R) = WD(A'', L, R) - c \le$$
$$\le Best(F_1, Rd(F_2, T_2), L, R) - c \le$$
$$\le WD(B, L, R)$$

Разбор случая Б закончен. Случаи А и В разбираются аналогично.

3.3.4. Структуры данных и оценки времени работы.

При вычислениях по рекурсивным уравнениям п.3.3.3.3 необходимо решить две взаимосвязанные проблемы: (1) как хранить данные о промежуточных парах лесов и (2) как избежать рассмотрения пар под-лесов, которые не выводимы из исходных лесов F_1 и F_2. Эти задачи могут быть решены с помощью следующей двухэтапной схемы. На первом этапе с помощью рекурсии, задаваемой определением 3.3.10, строится множество ***Descr***, содержащее описатели всех пар $<G_1, G_2>$, выводимых из $<F_1, F_2>$. Это множество представлено структурой, позволяющей выполнить операции ДОБАВИТЬ и НАЙТИ за время $log(Size)$, где ***Size*** – размер множества. При этом для каждой пары $<G_1, G_2>$ запоминаются указатели на дескрипторы ее наследников. После того, как все дескрипторы построены, они упорядочиваются по возрастанию суммы размеров лесов G_1 и G_2.

На втором этапе мы, обходим дескрипторы в указанном порядке и для каждого дескриптора запоминаем его характеристики и для каждой характеристики - ссылку на наследника, которому соответствует значение максимума (см. п.3.2.4).

Значение $Best(F_1, F_2, \varnothing, \varnothing)$ будет искомым весом оптимального выравнивания, а само выравнивание может быть восстановлено по построенным на втором этапе ссылкам.

Оценим сложность работы алгоритма. Пусть K – количество промежуточных пар лесов. Тогда, очевидно, потребная память есть $O(K)$, а время - $O(Klog(K))$. Метод выбора активных деревьев, описанный выше, дает $K = O(m^2 n \ lg \ (n))$ ([см. 176, 57], где $m \le n$ – длины сравниваемых последовательностей. Таким образом, получаем для памяти оценку $O(m^2 n \log (n))$, а для времени - оценку $O(m^2 n \log^2 (n))$.

3.4. Предсказание вторичной структуры РНК.

3.4.1. Постановка задачи.

В этом разделе представлен алгоритм вычисления энергии внутренних петель в структурах РНК в рамках модели Цукера-Мзтьюза-Тернера (Nearest Neighbour Model, NNM, см. [215]) – общепринятой в настоящее время модели строения РНК. Такой алгоритм является неотъемлемой частью общего алгоритма поиска оптимальной (т.е. имеющей минимальную свободную энергию) структуры для данной последовательности РНК в рамках NNM. Наш алгоритм обобщает алгоритм [116] на случай принятых в NNM формул для оценки энергии внутренних петель и имеет временную сложность $O(M*\log^2(L))$, где L - длина последовательности, M - количество теоретически допустимых спариваний. Т.к. $M \le L^2$, эта сложность существенно превосходит сложность $O(L^3)$ лучшего из ранее предложенных алгоритмов [204].

Рассматриваемая задача напрямую не связана с выравниванием последовательностей. Однако, как показано в предыдущих разделах этой главы, использование сведений о вторичной структуре, в том числе – теоретических предсказаний структуры, весьма важно для сравнения биополимеров. Кроме того, как мы увидим ниже, использованная алгоритмическая техника близка к технике выравнивания последовательностей.

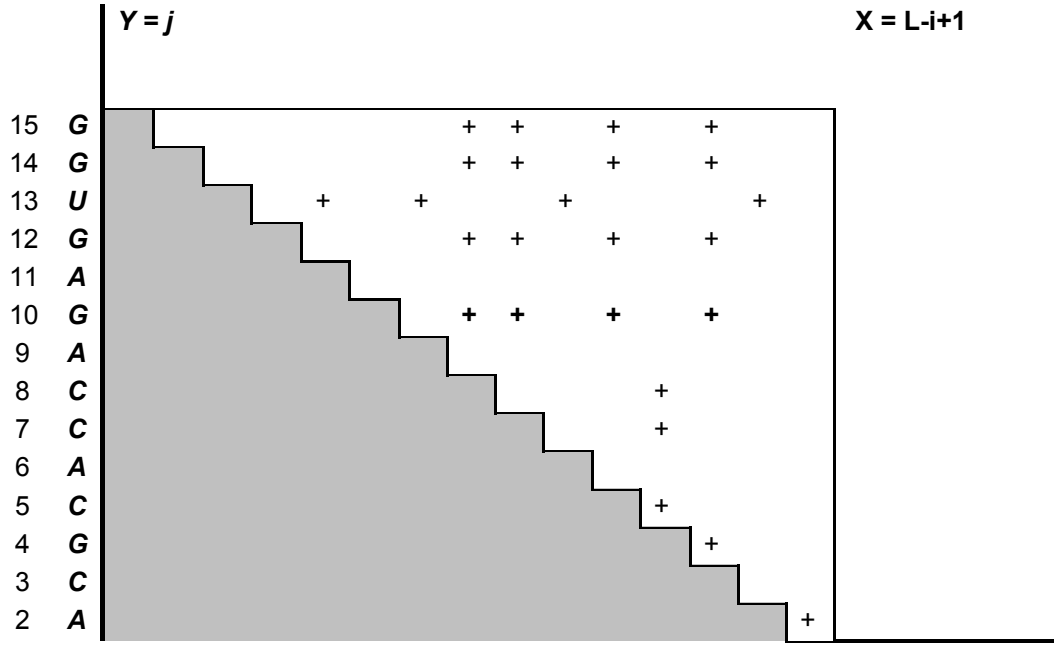


Рис. 3.4.1. Представление допустимых спариваний в виде точечной матрицы. Рассмотрена последовательность РНК UACGCACCAGAGUGG (L=15). Каждое допустимое спаривание *(p, q)* представлено знаком '+' в позиции *(L-p+1, q)*. Это соответствует тому, что одна копия молекулы расположена вдоль оси X справа налево, а другая – вдоль оси Y снизу вверх. Спаривания, принадлежащие строке ROW_{10} выделены жирным.

3.4.2. Терминология и обозначения.

3.4.2.1. Спаривания и структуры.

На протяжении этого раздела мы будем считать фиксированной последовательность РНК длины L и множество U, состоящее из M спариваний позиций этой последовательности. В связи с этим, в отличие от раздела 3.3, под структурой РНК здесь мы будем понимать множество спариваний $P \subseteq U$, удовлетворяющее условию отсутствия псевдоузлов (см. определение 3.3.1). Спаривания из U будут называться *допустимыми*.

Будем считать, что для каждой пары $(p, q) \in U$ выполнено: $p < q$. Нам будет удобно представлять множества спариваний на точечной матрице (dot-matrix): спаривание (p, q) представляется точкой $(L-p+1, q)$. Все эти точки расположены в правом верхнем треугольнике матрицы размером LxL (см. рис. 3.4.1).

Введем дополнительные обозначения. Ниже $U(A, B) \subseteq U$ обозначает множество допустимых спариваний (p, q), для которых выполнено $A+3 \le p < q \le B-3$ (константа 3 следует из теории Цукера-Тернера-Мэтьюза, см. [215]). Для каждого $B \in [1, L]$ через ROW_B обозначается множество всех допустимых спариваний вида (p, B). Через $DIAG_r$ обозначается множество спариваний

$$DIAG_r = \{(p, q) \mid (p, q) \in U,\ p+q = r\},$$

это множество будем называть r-ой *диагональю.* Множество $STRIP_r$, где $width = 6$ – параметр модели NNM, - это множество диагоналей, окружающих r-ю диагональ:

$$STRIP_r = \{DIAG_i \mid r-width < i < r+width\}.$$

Множество $STRIP_r$ будем называть r-й *полосой.*

3.4.2.2. Типы структур и петель.

В рамках модели NNM энергия структуры РНК представляется как сумма энергий т.н. петель (loop). Каждую петлю можно представить себе, как простой цикл в представлении вторичной структуры РНК в виде планарного графа [19]. В этом графе вершинами являются нуклеотиды, а ребра (двух видов) изображают соответственно валентные и водородные связи. Петли делятся на типы в зависимости от количества входящих в них водородных связей и длин неспаренных участков. Ниже приводятся необходимые сведения о петлях и вычислении их энергий. (см. рис. 3.4.2)

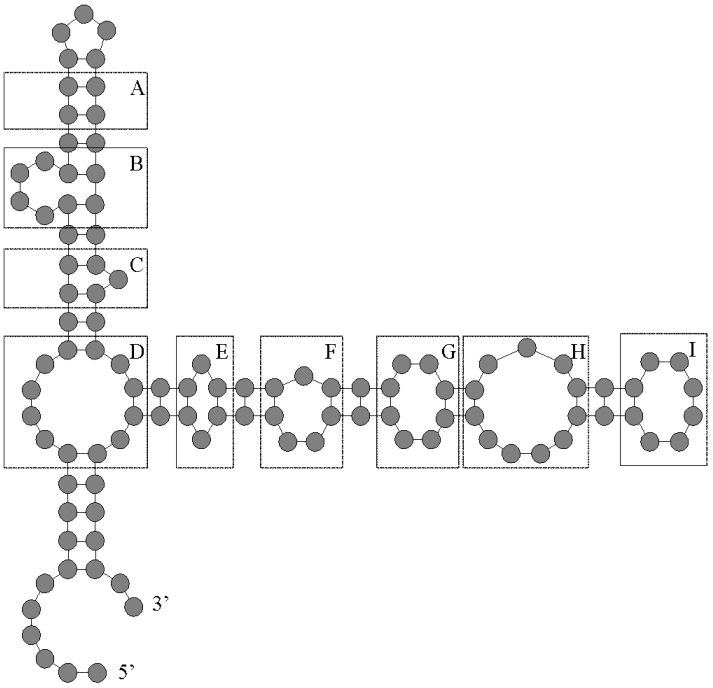


Рис. 3.4.2. Пример вторичной структуры РНК, содержащей разные виды петель: A - стекинг, B - выпячивание (bulge) общего вида, C – специальное выпячивание (длины крыльев 0 и 1), D – ветвящаяся (multi-branched) петля, E – специальная внутренняя (internal) петля (длины крыльев 0 и 1), F – специальная внутренняя петля (длины крыльев 1 и 2), G – специальная внутренняя петля (длины крыльев 2 и 2), H – внутренняя петля, I – шпилька (hairpin loop).

Определение 3.4.1. Пусть $P \subseteq U$ – структура, $(A, B) \in P$. Спаривание *(A, B) замыкает шпильку* (hairpin), если ни одна из позиций $x \in [A+1, B-1]$ не участвует в спариваниях структуры *P*. В рамках модели NNM считается, что для пар (*A, B),* замыкающих шпильки, выполнено: $B-A > 3$.

Определение 3.4.2. Пусть $P \subseteq U$ – структура, $(A, B) \in P$. Спаривание *(A, B) замыкает внутреннюю петлю* (internal loop) в *P*, если существует такое

спаривание *(p, q)* $\in$ *P,* что $A < p < q < B$ и ни один из нуклеотидов *x,* где $A < x < p$ или $q < x < B$, не участвует в спариваниях структуры *P*. Спаривание *(p, q)* называется *внутренним спариванием* петли; фрагменты [*A+1, p-1*] и [q+*1, B-1*] – *крыльями* внутренней петли.

Определение 3.4.3. Пусть $P \subseteq U$ – структура, *(A, B)* $\in$ *P*. Спаривание *(A, B) замыкает ветвящуюся петлю* (multibranch loop), если существуют такие спаривания $(p_1, q_1), \ldots, (p_k, q_k) \in P$ $(k \geq 2)$, что $A < p_1 < q_1 < \ldots < p_k < q_k < B$ и ни один из нуклеотидов *x,* где $A < x < B$ и $x \notin ([p_1, q_1] \cup \ldots \cup [p_k, q_k])$, не участвует в спариваниях структуры *P*.

Определение 3.4.4. Будем говорить, что спаривание *(A, B) – замыкающее спаривание* (closing pairing) в структуре *P,* если *P* не содержит пар *(x, y),* где $x < A$ или $y > B$. Структура *P* называется *(A, B)-структурой,* если в ней нет таких пар *(x, y),* что $x < A$ или $y > B$. Будем говорить, что *(A, B)*-структура *P* имеет тип 0, если *(A, B)* $\notin$ *P*. Будем говорить, что *(A, B)*-структура *P* имеет тип 1 (соответственно, 2, 3), если *(A, B)* $\in$ *P* и *(A, B)* замыкает шпильку (внутреннюю петлю, ветвящуюся петлю).

Определение 3.4.5. *Оптимальная* структура РНК в данном классе структур – это структура, имеющая минимальную свободную энергию среди структур рассматриваемого класса. Через *IStruct(A, B; p, q),* где $A < p < q < B$, обозначается оптимальная структура в классе таких структур *P*, что спаривания *(A, B), (p, q)* образуют в *P* внутреннюю петлю и *(A, B)* - замыкающая пара структуры *P*. Через *IStruct(A, B)* обозначается оптимальная структура среди структур вида *IStruct(A, B; p, q),* где $A < p < q < B$.

3.4.2.3. Энергии.

В этом разделе приведены необходимые сведения о вычислении энергий в рамках модели NNM.

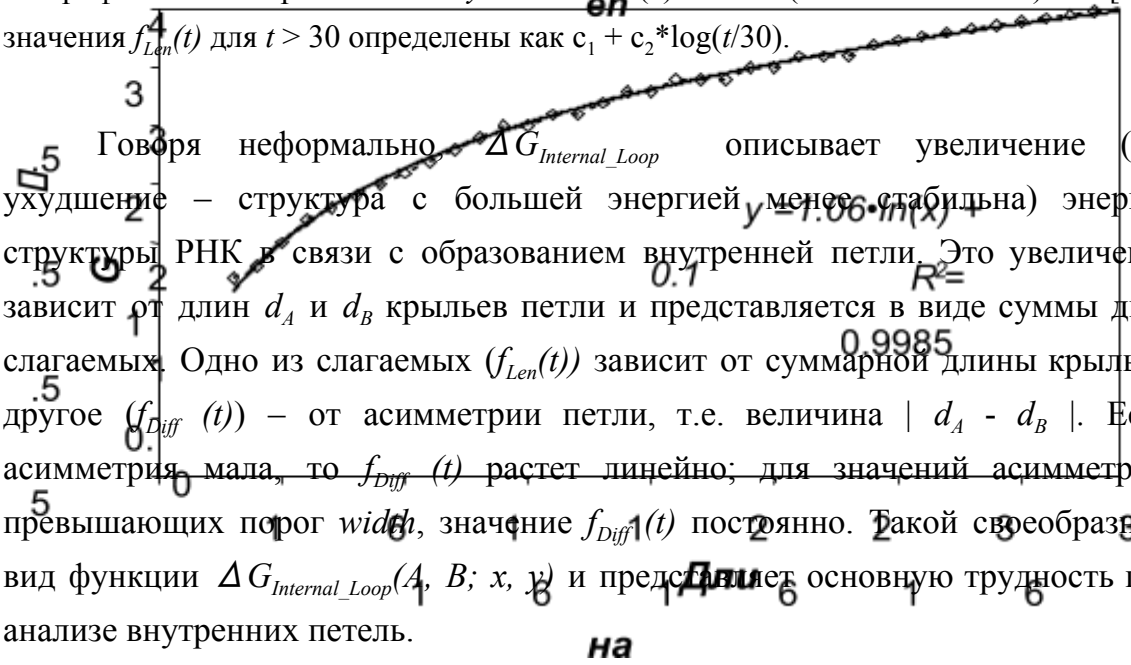
Определение 3.4.6. Пусть спаривания *(A, B), (x, y)* $\in$ *P* , где $A<x<y<B$, образуют внутреннюю петлю; $d_A = x-A-1$ и $d_B = B-y-1$ –длины крыльев этой петли. Энергия петли $\langle (A, B), (x, y) \rangle$ - это величина

$$\Delta G_{Internal\_Loop}(A, B; x, y) =$$
$$= f_{Len}(d_A+d_B) + f_{Diff}(d_A-d_B) =$$
$$= f_{Len}((B-A) - (y-x+2)) + f_{Diff}((B+A) - (y+x)) \qquad (3.4.1)$$

Здесь $f_{Len}(t)$ – выпуклая функция (см. рис. 3.4.3), она хорошо аппроксимируется логарифмической функцией. Функция $f_{Diff}(t)$ определяется соотношениями:

$$f_{Diff}(t) = base\_level, \quad \text{if } |t| \geq width$$
$$f_{Diff}(t) = (base\_level / width)*|t|, \text{ if } |t| < width; \quad (3.4.2)$$

Рис. 3.4.3. График функции $f_{Len}(t)$ по работе [215] (пунктирная линия) и ее логарифмическое приближение $y = 1.06*\ln(x) + 0.1$ (сплошная линия). В [215] значения $f_{Len}(t)$ для $t > 30$ определены как $c_1 + c_2*\log(t/30)$.



Говоря неформально $\Delta G_{Internal\_Loop}$ описывает увеличение (т.е. ухудшение – структура с большей энергией менее стабильна) энергии структуры РНК в связи с образованием внутренней петли. Это увеличение зависит от длин d_A и d_B крыльев петли и представляется в виде суммы двух слагаемых. Одно из слагаемых ($f_{Len}(t)$) зависит от суммарной длины крыльев, другое ($f_{Diff}(t)$) – от асимметрии петли, т.е. величина $|\ d_A - d_B\ |$. Если асимметрия мала, то $f_{Diff}(t)$ растет линейно; для значений асимметрии, превышающих порог *width*, значение $f_{Diff}(t)$ постоянно. Такой своеобразный вид функции $\Delta G_{Internal\_Loop}(A, B; x, y)$ и представляет основную трудность при анализе внутренних петель.

Наш алгоритм опирается на то, что значение параметра *width* невелико (*base_level* = +3.00, *width* = 6). Таким образом, значения $f_{Diff}(t)$ отличаются от константы лишь в *2width-1* =11 точках.

Замечание. С точки зрения модели NNM особо трактуются следующие случаи:

1) $d_A = d_B = 0$ (стекинг);

2) $min(d_A, d_B)=0$; $max(d_A, d_B)>0$; (выпячивание, bulge);

3) $0< d_A , d_B <4$ (внутренние петли малой длины).

Чтобы не загромождать изложение, мы здесь пренебрегаем этими деталями. Алгоритм, полностью учитывающий все особенности теории NNM, реализован в программе AFOLD [242] и имеет ту же сложность, что и описанный ниже.

Определение 3.4.7. Пусть $(A, B) \in U$. Через $\Delta G(A, B)$ обозначается минимальная возможная энергия среди структур с замыкающим спариванием (A, B). Через $\Delta G_K(A, B)$ $(K = 0, 1, 2, 3)$ обозначается наименьшая возможная энергия среди структур типа K с замыкающим спариванием (A, B) (см. Определение 3.4.4).

Напомним, что по определению,

$$\Delta G(A, B) = min\{ \Delta G_0(A, B),\ \Delta G_1(A, B),\ \Delta G_2(A, B),\ \Delta G_3(A, B) \}.$$

Теория NNM предлагает рекурсивные формулы для вычисления $\Delta G_K(A, B)$ для каждого типа K замыкающей петли, в зависимости от энергий $\Delta G(x, y)$ для других спариваний (x, y), входящих в замыкающую петлю, и длин неспаренных участков. Для нас представляет интерес формула для случая внутренней петли:

$$\Delta G_2(A, B) =$$
$$= \min\{ \Delta G(x, y) + \Delta G_{Internal\_Loop}(A, B; x, y) \mid (x, y) \in U(A, B)\} \qquad (3.4.3)$$

3.4.3. Вычисление энергий внутренних петель: общее описание.

3.4.3.1. Уточнение постановки задачи

Алгоритмы, которые предсказывают вторичную структуру РНК в рамках общепринятой в настоящее время модели NNM, следуют схеме, изображенной на рис. 3.4.4. Алгоритмы просматривают все допустимые спаривания строка за строкой в порядке возрастания номеров строк $B \in [1, L]$. При этом для каждого спаривания $(A, B) \in ROW_B$ вычисляется оптимальная структура каждого типа $K \in \{0, 1, 2, 3\}$.

```
        algorithm OptimalRNA(input: RNA[1..L])
        begin
                // 1. Пролог
1.          Prolog;                          // Инициализировать структуры данных

                // 2. Основной цикл
            for all B from 1 to L do begin
2.1.            HairpinRow(B);               // Вычисление ΔG1(x, B), x ∈ ROW_B
                                             // шпильки
2.2.                SimpleLoopRow(B);        // Вычисление ΔG2*(x, B), x ∈ ROW_B :
                                             // особые случаи внутренних петель
2.3.                InternalLoopRow(B);      // Вычисление ΔG2(x, B), x ∈ ROW_B
                                             // стандартные внутренние петли
```

```
2.4.    Multi-branchLoopRow(B);    // Вычисление ΔG3(x, B) и ΔG0(x, B), x ∈ ROW_B
                                   //  ветвящиеся петли
2.5.    OptimalLoopRow(B);         // минимум из 2.1 – 2.4
        end
            // 3. Эпилог
3.      Epilog;                    // Восстановить оптимальную структуру РНК
                                   //  из полученных данных
    end
```

Рис. 3.4.4. Алгоритм построения оптимальной вторичной структуры для данной последовательности РНК. Каждая из функций 2.1 – 2.4 находит значения величин $\Delta G_K(x, B)$ для спариваний, принадлежащих строке ROW_B (см. комментарии в тексте). В момент вызова этих функций уже известны энергии $\overleftarrow{G}(x, y)$ оптимальных структур с заданной замыкающей парой (x, y) для всех $(x, y) \in U$ таких, что $y < B$.

По техническим причинам функция *MultibranchLoopRow(B)* одновременно вычисляет как значения $\Delta G_3(x, B)$ (спаривание (x, B) замыкает ветвящуюся петлю), так и $\Delta G_0(x, B)$) (спаривание (x, B) отсутствует). Функция *OptimalLoopRow(B)* для каждой пары $(x, B) \in ROW_B$, находит оптимальную структуру среди всех (x, B)-структур, как минимум энергий, полученных в строках 2.1 – 2.4. Кроме минимальных энергий указанные алгоритмы запоминают ссылки, которые позволяют восстановить сами оптимальные структуры (строка 3).

В работе [242] мы представили новый алгоритм предсказания оптимальной вторичной структуры РНК. Его отличие – в реализации функции *InternalLoopRow* (см. строку 2.3 на рис. 3.4.4), реализация остальных функций остается такой же, как, например, в [342]. Предложенный алгоритм использует метод динамического программирования для разреженных матриц [115] и существенно превосходит по быстродействию лучший из ранее известных методов [204].

3.4.3.2. Алгоритм InternalLoopRow

В подразделе *3.4.3.2* описана общая схема алгоритма *InternalLoopRow*. Реализация вспомогательных алгоритмов описана в п. 3.4.4. Оценки сложности приведены в п. 3.4.5. Напомним, что в момент вызова функции *InternalLoopRow*(B) уже известны энергии $\overleftarrow{G}(x, y)$ оптимальных структур с заданной замыкающей парой (x, y) для всех $(x, y) \in U$ таких, что $y < B$.

```
algorithm InternalLoopRow(input: B){
begin
  for all B from 2 to L do begin  // Т.к. B>A≥1, то B>2
     InternalLoopMainRow(B); // Вычисляет ΔG_Main(A, B) для всех пар (A, B) ∈ ROW_B
     InternalLoopStripRow(B); //  Вычисляет ΔG_Strip(A, B) для всех пар (A, B) ∈ ROW_B
     InternalLoopFinalRow(B); // Для всех пар (A, B) ∈ ROW_B вычисляет
                              // ΔG_IStruct(A, B) =
                              //      = min{ base_value + ΔG_Main(A, B), ΔG_Strip(A, B) }
  end
end
```

Рис. 3.4.5 Алгоритм *InternalLoopRow(B)*.

Пусть r, x, y – целые числа, $r > x+y$. Имея в виду формулы (3.4.1) и (3.4.2), обозначим через $\Delta G(r, x, y)$ сумму

$$\Delta G(r, x, y) = \Delta G(x, y) + f_{Diff}(r-(y+x)) \qquad (3.4.4)$$

Используя (3.4.1), (3.4.2) и (3.4.4), формулу (3.4.3) можно преобразовать следующим образом:

$$\Delta G_2(A, B) = \min\{$$
$$\min\{\Delta G(x, y) + f_{Len}((B-A) - (y-x+2)) + base\_level \mid (x, y) \in U(A, B)\},$$
$$\min\{\Delta G(x, y) + f_{Len}((B-A) - (y-x+2)) + f_{Diff}((B+A) - (y+x)) \mid (x, y) \in STRIP_{A+B}\},$$
$$\} =$$

$$=\min\{$$
$$base\_level + \min\{\Delta G(x, y) + f_{Len}((B-A) - (y-x+2)) \mid (x, y) \in U(A, B)\},$$
$$\min\{\Delta G(A+B, x, y) + f_{Len}((B-A) - (y-x+2)) \mid (x, y) \in STRIP_{A+B}\}$$
$$\}$$

(3.4.5)

Алгоритм выполнения вызова *InternalLoopRow(B)* (см. рис. 3.4.4) представлен на рис. 3.4.5.

При обработке строки $\mathrm{ROW_B}$ наш алгоритм сначала вычисляет значения величин

$$\Delta G_{Main}(A, B) = \min\{\Delta G(x, y) + f_{Len}((B–A) – (y–x+2)) \mid (x, y) \in U(A, B)\} \quad (3.4.6)$$

для всех $(A, B) \in ROW_B$ (см. вызов *InternalLoopMainRow(B))*. Затем вычисляются величины

$$\Delta G_{Strip}(A, B) = \min\{\Delta G(A+B, x, y) + f_{Len}((B–A) – (y–x+2))| (x, y) \in STRIP_{A+B}\} \quad (3.4.7)$$

(см. вызов *InternalLoopStripRow(B)*). И, наконец, по формуле (3.4.5) вычисляются значения $\Delta G_2(A, B)$ (см. вызов *InternalLoopFinalRow(B)*)

При вызовах алгоритмов *InternalLoopMainRow(B)* и *InternalLoopStripRow(B),* кроме вычислений указанных величин, также выполняются действия, связанные с поддержкой необходимых структур данных (см. ниже). Алгоритмы *InternalLoopMainRow(B)* и *InternalLoopStripRow(B)*существенно используют выпуклость функции $f_{Len}(t)$, они подробно описаны в разделе 3.4.4.

3.4.4. Функции *InternalLoopMainRow(B)* и *InternalLoopStripRow(B)*

3.4.4.1. «Разделяй и властвуй». Вспомогательные области.

Вычисление значения $\Delta G_{Main}(A, B)$ и $\Delta G_{Strip}(A, B)$ для всех пар $(A, B) \in ROW_B$ основывается на методе динамического программирования для разреженных матриц, предложенного в [117]. Как и алгоритм 3 из [117], наш алгоритм использует подход «разделяй и властвуй» [24]. При этом, в отличие от стандартных реализаций подхода «разделяй и властвуй», мы предварительно явно строим вспомогательные области (т.н. области предпросмотра и обратного просмотра), к которым применяется очередной этап процедуры «разделяй и властвуй». Это сделано ввиду необходимости использовать алгоритм *InternalLoopRow(B)*, а, следовательно, и его под-алгоритм *InternalLoopMainRow(B)*, внутри алгоритма *OptimalRNA*, см. рис. 3.4.4.

Во время инициализации структур данных алгоритма *OptimalRNA* (см. строку 1 на рис. 3.4.4) для каждой строки ROW_B вычисляются два числа p_B и q_B так, что $p_B < B \le q_B$; алгоритм вычисления этих чисел обсуждается ниже.

Определение 3.4.7. Областью *предпросмотра* $FORWARD_B$ строки B называется множество допустимых склеек, лежащих в строках с номерами от B до q_B Областью *обратного просмотра* $FORWARD_B$ строки B называется множество допустимых склеек, лежащих в строках с номерами от p_B до B-1. Иными словами,

$$BACKWARD_B = \{(x, y) \mid (x, y) \in U \,\&\, p_B \le y \le B\text{-}1\}$$

$$FORWARD_B = \{(x, y) \mid (x, y) \in U \,\&\, B \le y \le q_B\}$$

Каждая пара областей $BACKWARD_B$ и $FORWARD_B$ соответствует выполнению одного шага процедуры «разделяй и властвуй» в [117]; алгоритм вычисления значений p_B, q_B следует из этой процедуры. Свойства областей $BACKWARD_B$ и $FORWARD_B$ описывает приведенная ниже лемма 3.4.1.

Определение 3.4.8. Пусть $q, B \in [1, L]$, $q > B$. Положим

$$WORKEDZONES(q; B) = \cup \{BACKWARD_k \mid k \le B < q \le q_k\} \qquad (3.4.9)$$

Другими словами, $WORKEDZONES(q; B)$ – это объединение всех таких областей $BACKWARD_k$, где $k < B$, что соответствующая область $FORWARD_k$ содержит строку ROW_q.

Лемма 3.4.1 Алгоритм *SetZones* (см. рис. 3.4.6) для всех $B \in \{1, \ldots, L\}$ определяет значения чисел p_B, q_B и, соответственно, области $BACKWARD_B$; $FORWARD_B$.При этом:

(i). Каждая строка ROW_B принадлежит к $O(\log(L))$ backward- и forward-областей.

(ii). Для любого B выполнено:

$$WORKEDZONES(B; B) = Z(1, B–1).$$

(iii) Время работы алгоритма *SetZones* равно $O(L \bullet \log(L))$.

Доказательство.

Работа алгоритма *SetZones* сводится к вызову рекурсивного алгоритма *Zones* для всего диапазона строк *[1, L]* (т.е. с $p=1; q=L$). Третий входной параметр алгоритма *Zones* равен глубине текущего вызова алгоритма *Zones* и нужен только для доказательства настоящей леммы.

```
// Для каждого B ∈ {2, ..., L} алгоритм SetZones устанавливает:
//        P[B] = p_B; Q[B] = q_B;
```

```
int P[1..L]; Q[1..L]; Rang[1..L];
algorithm SetZones
        Zones(1, L, 0)
end

algorithm Zones(arg int p, q)
// Устанавливает значения P[B]; Q[B] для B ∈ [p+1, q]
begin
        int S; t
        if (q=p+1) then begin
                P[q] =p; Q[q] =q;
        end else begin
                S = |U|;                                  // |U| = Σ_{i=p,...,q}|ROW_i|;
                t = min{r| Σ_{i=p,...,r}|ROW_i| ≥ S/2};
                P[t]=p; Q[t] = q;
                P[t+1]=t; Q[t] = q;
                if t> p+1 then Zones(p, t-1);
                if q> t+1 then Zones(t+1, q);
        end
end
```

Рис. 3.4.6. Алгоритм *SetZones,* определяющий значения величин p_B и q_B.

Индукцией по разности *q-p* входных параметров алгоритма *Zones* легко показать, что значения p_B и q_B вычисляются для всех значений $B \in [p+1, q]$, причем при выполнении алгоритма *SetZones* для каждого $B \in [2, L]$ значения *P{B}, Q[B]* присваиваются только один раз. Аналогично, индукцией по *q-p* доказывается утверждение (ii) леммы (подробнее см. [242], дополнительные материалы, стр. 16). Для доказательства утверждения (i) рассмотрим некоторую строку $B \in [1, L]$ и дерево вызовов алгоритма *Zones*. Очевидно, если для двух вызовов $Zones(p_1, q_1, r_1)$ и $Zones(p_2, q_2, r_2)$ выполнено $B \in [p_1, q_1]$ и $B \in [p_2, q_2]$, то эти вызовы находятся на одном пути из корня в лист в дереве вызовов. Но при каждом вызове разность параметров $|p-q|$ уменьшается по крайней мере в два раза. Поэтому высота дерева вызовов равна *O(log(L)),* что и завершает доказательство утверждения (ii). Оценка времени работы алгоритма проводится стандартным для метода «разделяй и властвуй» способом и следует из двукратного уменьшения разности $|p-q|$ при каждом вызове, а также, что время выполнения собственно вызова *Zones(p, q)* без учета времени выполнения вложенных вызовов равно *O(q-p)*. Лемма доказана.

3.4.4.2. E-алгоритм

Определение 3.4.9. Пусть (A, B) – допустимое спаривание и $P \subseteq U(A, B)$. Через $\Delta G_{MainPARTIAL}(A, B; P)$ обозначим величину

$$\Delta G_{MainPARTIAL}(A, B; P) =$$

$$\min\{\Delta G(x, y) + f_{Len}((B-A) - (y-x+2)) \mid (x, y) \in P\} \qquad (3.4.8)$$

Пусть $Z(p, q)$, где $1 \le p \le q \le L$, обозначает множество всех допустимых спариваний, принадлежащих строкам с номерами от p до q:

$$Z(p, q) = ROW_p \cup \ldots \cup ROW_q = \{(x, y) \mid (x, y) \in U \,\&\, p \le y \le q\}$$

Рассмотрим следующую вспомогательную проблему.

Проблема E [117].

Дано:

1) целые p, k, q; где $1 \le p < k \le q \le L$;
2) два множества допустимых спариваний: $P \subseteq Z(p, k-1)$ и $Q \subseteq Z(k, q)$;
3) энергии $\Delta G(x, y)$ для всех $(x, y) \in P$.

Надо: для всех $(A, B) \in Q$, вычислить $\Delta G_{MainPARTIAL}(A, B; P)$

Алгоритм, решающий проблему E (ниже он называется E-алгоритм, основан на процедуре динамической минимизации (dynamic minimization procedure), описанной в разделе 2 работы [117]. Его время работы $O(N \bullet log(N))$, где N – общее количество спариваний в $P \cup Q$, и потребная память равна $O(N)$. E-алгоритм просматривает все пары из $P \cup Q$ в порядке убывания их x-координат, т.е. меньших (первых) элементов. В момент $t \in [1, L]$ алгоритм ECORE хранит множество

$$ACTIVE_t \subseteq \{(x, y) \mid (x, y) \in P, x \ge t\},$$

которое состоит из всех спариваний $(x, y) \in P$, принадлежащих множеству

$\{(x, y) \in P | x \ge t$ и

(x, y) может соответствовать минимуму в (3.4.12)

для некоторого спаривания $(p, q) \in Q$, где $p < t$

$\}$.

Т.к. функция f_{Len} выпуклая, то модификация множества $ACTIVE_t$ (если текущее спаривание $P \cup Q$ принадлежит P) и поиск в $ACTIVE_t$ (если текущее спаривание $P \cup Q$ принадлежит Q) может в среднем быть выполнено за $\log(N)$

операций. Описание структуры данных ACTIVE, используемой для хранения текущего множества $ACTIVE_t$ и детали процедуры динамической минимизации см. [117].

3.4.4.3. Реализация алгоритма InternalLoopMainRow(B)

В качестве глобальных объектов алгоритм *InternalLoopMainRow* использует вещественные переменные *MainWork*[*p*, *q*] – для каждого спаривания (*p*, *q*) ∈ *U*. Переменные инициализируются значениями +∞ во время препроцессинга алгоритма *OptimalRNA*, см. рис. 3.4.4, строка 1. После вызова *InternalLoopMainRow(B)* для переменной *MainWork*[*p*, q] выполнено:

$$MainWork[p, q] = Main_{PARTIAL}(p, q; WORKEDZONES\ (q; B))$$

(см. опредедения 3.4.8, 3.4.9).

Это обеспечивается следующим образом. При вызове *InternalLoopMainRow(B)* решается проблема E с набором входных данных: (1) $p_B < B < q_B$, (2) $P = BACKWARD_B$, $Q = FORWARD_B$; и (3) значения $\Delta G(x, y)$ для всех (x, y), таких, что $y<B$. В соответствии с (3.4.8) в итоге для всех спариваний $(p, q) \in Z(B, q_B)$ будут вычислены значения

$$Main_{PARTIAL}\ (p, q; BACKWARD_B).$$

Далее, значения *MainWork*[*p*, *q*] для всех $(p, q) \in Z(B, q_B)$ изменяются следующим образом:

$$\begin{aligned}
&MainWork[p, q] := \\
&=\min\{Main_{PARTIAL}(p, q; BACKWARD_B), MainWork[p, q]\} = \\
&= \min\{Main_{PARTIAL}(p, q; BACKWARD_B), \\
&\qquad Main_{PARTIAL}(p, q; WORKEDZONES(q; B-1))\} \\
&= Main_{PARTIAL}(p, q; WORKEDZONES\ (q; B)),
\end{aligned}$$

что и требовалось. Ввиду утверждения (ii) леммы 3.4.1, после *L* вызовов *InternalLoopMainRow(B)*, $B \in [1, L]$. значения *Main(p, q)* для всех $(p, q) \in U$ будут вычислены корректно.

Время вызова *InternalLoopMainRow(B)* определяется временем работы E-алгоритма и равно $O(N \bullet log(N))$, где *N* - общее число спариваний в $BACKWARD_B$ и $FORWARD_B$. Таким образом, в силу утверждения (i) Леммы

3.4.1,. общее время работы всех L вызовов *InternalLoopMainRaw,* включая инициализацию глобальных объектов равно $O(M \bullet \log^2(L))$.

3.4.4.4. Реализация алгоритма InternalLoopStripRow(B)

Вычисление значений $\Delta G_{Strip}(A, B)$ для всех пар $(A, B) \in ROW_B$ также основано на решении E-проблемы и использовании вспомогательных областей (см. п. 3.4.4.1). При вызове алгоритма *InternalLoopStripRow(B)* решается несколько экземпляров проблемы E, каждый из экземпляров соответствует диагонали $DIAG_r$, $r = B+1, \ldots, 2*B-1$. Для всех копий целочисленные элементы входных данных – это p_B, B and q_B, которые были вычислены алгоритмом *SetZones* (см. рис. 3.4.6). Входные множества спариваний для копии, соответствующей диагонали $DIAG_r$ - это множества

$$P = BACKWARD_B \cap STRIP_r; \qquad Q = FORWARD_B \cap DIAG_r;$$

веса спариваний $\Delta G(x, y)$, где $(x, y) \in P$, как и в случае алгоритма *InternalLoopMainRow(B)* к моменту вызова *InternalLoopStripRow(B)*уже вычислены.

Для каждого спаривания $(x, y) \in U$ мы используем переменную *STRIPWORK*[x, y], аналогичную переменной *MainWork*[x, y], описанной выше. Во время вызова *InternalLoopStripRow(B)*, все $B-1$ копий проблемы E решаются одновременно. Для этого используется массив *ACTIVESTRIP* длины L, его элементами являются структуры *ACTIVE* (см. 3.4.4.3); элемент *ACTIVESTRIP*[i] содержит текущее значение множества *ACTIVE* для копии проблемы E, соответствующей диагонали $DIAG_{B+i}$. Для спаривания $(x, y) \in BACKWARD_B$ мы определяем диапазон значений r, для которых $(x, y) \in STRIP_r$ и затем модифицируем структуры *ACTIVE*[$r-B$] для всех r из этого диапазона. Для спаривания $(p, q) \in FORWARD_B$, мы ищем минимум относительно элементов структуры *ACTIVE*[$p+q-B$]. Каждое спаривание $(p, q) \in BACKWARD_B$ принадлежит не более, чем $2*width-1$ полосам. Поэтому, время T_{STRIP} выполнения всех вызовов *InternalLoopStripRow* не более, чем в $2*width-1$ раз больше, чем аналогичное время для *InternalLoopMainRow*. Поэтому $T_{STRIP} = O(width*M*\log^2(L))$.

3.5. Оценки сложности.

Как показано в п. 3.4.3.3, общее время всех вычислений по формуле (3.4.6) оценивается сверху величиной

$$T_{MAIN} = c_1 * M * \log^2(L) \qquad (3.4.9)$$

а дополнительная память, используемая при этих вычислениях, равна *O(M)*.

Т.к. каждая пара (*x*, *y*) принадлежит только 2**width*–1 полосам, то общее время T_{DIAG} вычисления значений $\Delta G(A+B, x, y)$ для всех $(A, B) \in U$ и $(x, y) \in STRIP_{A+B}$ (эти значения нужны для вычисления (3.4.7) задается формулой:

$$T_{DIAG} = c_2 * width * M \qquad (3.4.10)$$

Собственно для вычисления (3.4.7) (при предвычисленных значениях $\Delta G(A+B, x, y)$), как показано в п. 3.4.4.4, суммарно проводятся за

$$T_{STRIP} = c_3 * width * M * \log^2(L) \qquad (3.4.11)$$

при необходимой дополнительной памяти O*(M*).

Таким образом, (см. (3.4.9) – (3.4.11)), для общего времени работы алгоритма получаем оценку O(*width***M**log^2(*L*)) и оценку памяти O*(M*).

Замечание. Для вычисления (3.4.7) можно использовать альтернативный алгоритм, основанный на использовании списков кандидатов, а не на методе «разделяй и властвуй». Это дает несколько лучшее время

$$T'_{STRIP} = c_4 * width * M * \log(L)$$

ценой некоторого увеличения размера используемой памяти. Тем не менее, и в этом случае необходимая память есть *O(M)*. Таким образом, при использовании этого алгоритма получаем несколько лучшую общую оценку времени:

$$O(M * \log^2(L) + width * M * \log(L))$$

при прежней оценке потребной памяти O*(M*).

Глава 4. Алгоритмы парного выравнивания, основанные на выделении локальных сходств.

4.0. Введение.

В предыдущих главах была подробно рассмотрены методы построения оптимального выравнивания двух последовательностей относительно заданного определения веса выравнивания. Эти алгоритмы имеют квадратичную временную сложность – их время работы (в худшем случае) пропорционально произведению длин последовательностей. Такое время слишком велико для многих биоинформатических приложений, в частности, - для сравнения синтенных фрагментов геномов (длина ~ 10^7 – 10^8 нуклеотидов). В разделе 4.1 представлен иерархический алгоритм геномного выравнивания OWEN, основанный на построении систем коллинеарных локальных сходств. Этот алгоритм не гарантирует оптимальности построенного выравнивания относительно как-либо определенного веса выравнивания, тем не менее, его результаты биологически адекватны. Разделы 4.2 и 4.3 посвящены поиску локальных сходств с помощью т.н. затравок – наиболее распространенному и быстродействующему из известных методов построения локальных сходств. В разделе 4.2 дано определение затравки в наиболее общем виде и рассмотрен простейший класс «неклассических» затравок – разреженные затравки, предложенные в [206]. Для этого класса указан вид оптимальной затравки среди затравок с одной несущественной позицией. Раздел 4.3 посвящен введенному нами классу затравок – классификационным затравкам, и некоторым алгоритмическим проблемам, связанным с вычислением чувствительности затравок этого класса [186,187,188]. Задача вычисления чувствительности произвольных затравок рассмотрена в разделе 5.2 главы 5.

4.1. Иерархическое выравнивание геномов.

4.1.1. Обоснование подхода

Задача выравнивания геномов, точнее – выравнивания синтенных (коллинеарных) участков генома (см. главу 1, п 1.3), на рубеже веков стала

одной из основных задач алгоритмической биоинформатики. Эта задача существенно отличается от рассмотренных в предыдущих главах задач выравнивания белков и относительно коротких фрагментов ДНК. В последнем случае существует сходство между последовательностями в целом, это сходство может быть представлено глобальным выравниванием, которое, в свою очередь, может быть (с погрешностями) восстановлено путем построения оптимального выравнивания относительно подходящей весовой функции. Существенным обстоятельством является и то, что для белков возможно построение структурных выравниваний, которые могут служить эталоном при проверке адекватности выбранной весовой функции (см. подробнее п.3.1).

Сходство синтенных участков геномов «средне близких» организмов (человек – мышь, *C.Elegance – C.Briggsae* и т.п.) существенно неоднородно и распадается на локальные сходства. Как правило, эти локальные сходства являются статистически достоверными, т.е. имеют достаточно низкое значение *P*-значения (*P-value*, иногда употребляется термин уровень значимости, см. [4]) для заданных длин последовательностей и распределения вероятностей. Большинство статистически достоверных локальных сходств коллинеарны, т.е. проекции на оба сравниваемых генома расположены в одинаковом порядке [286]. Это объясняется тем, что основные эволюционные события (замена нуклеотида, удаление/вставка фрагмента) не нарушают коллинеарности. Эволюционные события, нарушающие коллинеарность и создающие *конфликты* между сходствами (см. рис. 4.1.1), существенно более редки [268].

Есть три основных источника нарушения коллинеарности локальных сходств или *микроколлинеаоности* (термин «микроколлинеаоность» используется, чтобы отличить коллинеарность локальных сходств, как кодирующих, так и некодирующих, от *макроколлинеарности* - коллинеарности генов в синтенных участков геномов). Этими источниками являются (1) локальная конвергентная эволюция, (2) консервативный повтор, присущий обоим геномам, но по-разному расположенный в них; (3) локальные перестановки в геноме.

В случае средне-близких геномов конвергентная эволюция, как правило, не приводит к достаточно высокому уровню сходства, и, следовательно, при конфликте такого сходства со сходством ортологичных сегментов именно последние будут иметь более высокий уровень значимости. Вставки повторов могут (хотя и редко) привести к появлению неортологичных сходств высокого уровня значимости, поэтому повторы желательно предварительно замаскировать. В случае локальной перестройки оба сходства имеют равное право на включение в основную цепочку локальных сходств. Поэтому и в этом случае выбор на основе величины значимости локального сходства оправдан.

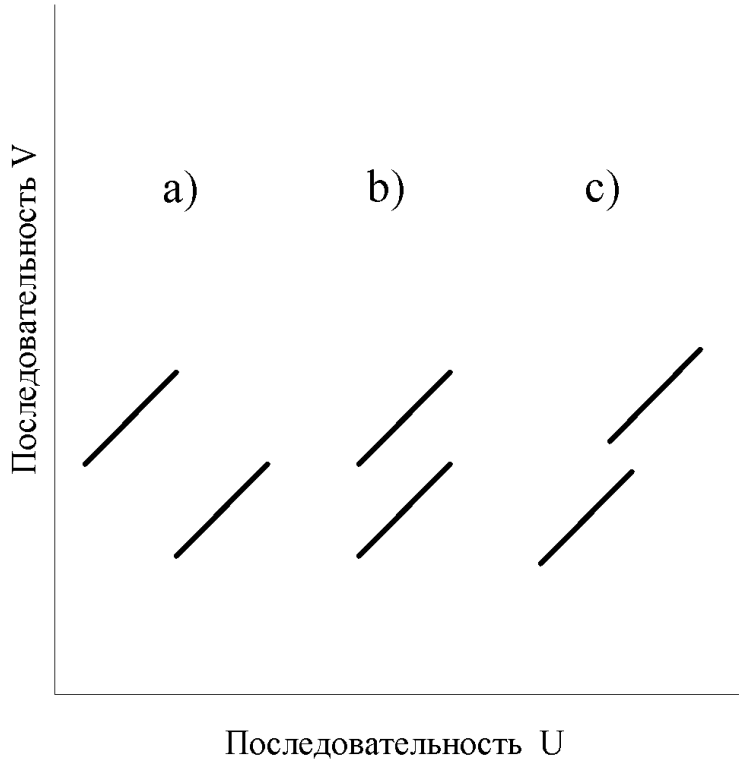


Рис. 4.1.1. Матрица сходств (dot-matrix), на которой показаны различные вида конфликтов между локальные сходствами a) Неустранимый конфликт: сходные фрагменты на разных последовательностях расположены в разном порядке b) Неустранимый конфликт: проекции сходств на последовательность *U* существенно перекрываются. c) Есть незначительное перекрытие проекций сходств на последовательность *U.* В этом случае конфликт может быть разрешен путем обрезания концов сходств.

Таким образом, отдавая предпочтение более значимому из конфликтующих сходств, мы, как правило, выбираем сходство между ортологичными фрагментами геномов (см. рис.4.1.2).

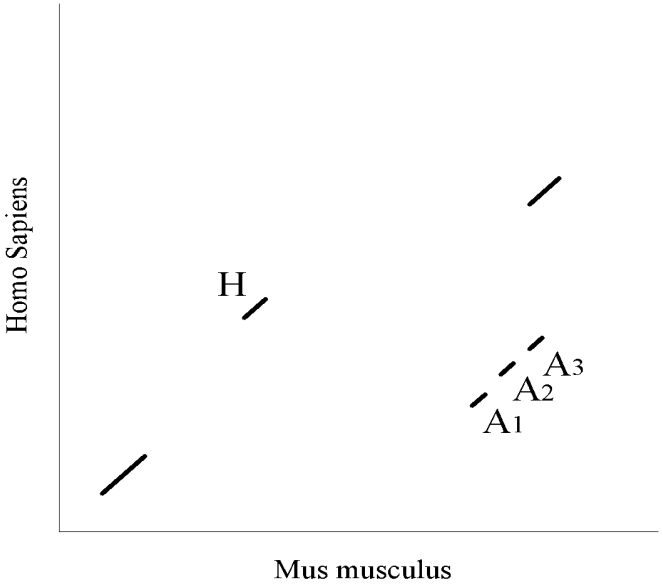


Рис. 4.1.2. Сравнение синтенных участков последовательностей мыши (AF139987) и человека (AF045555). Наиболее сильное сходство *H* – это ортологичное сходство, представляющее экзон 10 локусе Rfc2 (нуклеотиды 104514-104583 по последовательности мыши). Каждое из коллинеарных между собой сходств *A1, A2, и A3* не является ортологичным. Однако их общий вес выше, чем вес сходства *H*.

Ввиду сказанного выше, геномное выравнивание можно строить, исходя из следующих принципов.

1. Все конфликты между статистически значимым локальным сходством и любым количеством индивидуально менее значимых сходств разрешаются в пользу первого. Таким образом, сходство, которое не имеет конфликтов с более значимыми сходствами, всегда включается в цепь локальных сходств, образующих глобальное выравнивание.

2. Описанный выше принцип применяется как при сравнении последовательностей в целом, так и при сравнении их ортологичных фрагментов («фрактальность»). Благодаря этому, после того, как выделена цепь локальных сходств, значимых на уровне последовательностей в целом, алгоритм может пополнить эту цепь менее сильными сходствами, расположенными в коллинеарных участках между уже выделенными сходствами. Эти «дополнительные» сходства не являлись значимыми при

сравнении исходных последовательностей длиной, скажем, 10^7, но становятся значимыми при сравнении коллинеарных фрагментов длиной, скажем 10^3. Иными словами, более «сильные» локальные сходства поддерживают коллинеарные с ними более слабые сходства.

Цепь локальных сходств, построенную исходя из сформулированных выше принципов, будем называть *остовной* (backbone) цепью. Мы полагаем (см. аргументацию выше), что она близка к «эволюционно правильной» цепи, содержащей все ортологичные локальные сходства. В отличие от выравнивания белков (см. разделы 3.1, 3.2), при выравнивании геномов мы вынуждены полагаться только на индивидуальную статистическую значимость локальных сходств.

Описанный подход реализован в диалоговой программной системе *OWEN* [242, 244, 276]*;* используемые в ней алгоритмы описаны ниже. Распознавание локальных сходств при описанном подходе является функциональным параметром алгоритма геномного выравнивания (см., например, [242], [31], [206] и другие). Выбор алгоритма и его параметров, в частности определяет, считается ли некоторое сходство единым, или распадается на цепочку более мелких (и, следовательно, менее значимых) сходств. Таким образом, мы не избавляемся полностью (что невозможно) от проблемы выбора параметров. Достоинство подхода в том, что мы разделяем анализ последовательностей «в малом» (внутри локальных сходств) и «в большом» (построение цепи локальных сходств).

Параметры выравнивания, в частности, штрафы за удаления, используются только при поиске «в малом», где есть возможность опираться на экспериментально подтвержденные выравнивания. При поиске «в большом» единственным параметром остается выбор порога для уровня значимости (ср. п. 3.4). Зависимость от этого параметра достаточно грубая, кроме того, в диалоговой программной системе OWEN есть возможность «ручного» управления отбором локальных сходств.

Алгоритмически индивидуальное разрешение конфликтов имеет два достоинства. Во-первых, оно дает возможность использовать жадный алгоритм при выборе остовной цепи в заданном множестве локальных сходств. Во вторых, остовная цепь может строиться иерархически. Построение

начинается с выделения очень сильных («безусловно достоверных») выравниваний и разрешения конфликтов между ними. Затем аналогично и независимо друг от друга исследуются пропуски между элементами построенной цепи и т.д. Таким образом, мы избегаем затратного по времени анализа относительно слабых локальных сходств по последовательности в целом.

4.1.2. Основные определения.

Фиксируем две последовательности *U* и *V* в алфавите ДНК *Д = {a, c, g, t}*.

4.1.2.1 Локальные сходства и их цепи

Определение 4.1.1. *Локальным сходством* для последовательностей *U* и *V* будем называть пару фрагментов $H = <U[x_1, x_2], V[y_1, y_2]>$. Фрагменты $U[x_1, x_2]$ и $V[y_1, y_2]$ называются проекциями *H* соответственно на *U* и на *V*.

Мы будем предполагать, что каждому локальному сходству *H* приписан *вес Score(H)*. Мы не уточняем, как именно определяется вес сходства, ограничиваясь предположением, что *Score(H)* растет с ростом количества совпадений и убывает с ростом числа несовпадений и удалений (см. п. 2.3), так что сходство, имеющее больший вес, - лучше.

Определение 4.1.2. Сходство *H1 предшествует* сходству *H2* (обозначение: *H1 < H2),* если проекция *H1* на каждую из последовательностей *U, V* предшествует соответствующей проекции сходства *H2*. Сходства *H1* и *H2* образуют *конфликт,* если ни одно из них не предшествует другому. Два сходства *коллинеарны,* если они не образуют конфликт. *Цепь сходств* – это такая последовательность сходств *{H1, H2, …, HN}* , что *Hi-1<Hi* (*i = 2, …, N).* Сходство *H коллинеарно цепи* сходств *B= {H1, H2, …, HN),* если оно коллинеарно всем элементам *B.*

Пусть *H* - сходство и *B = {H1, H2, …, HN* – цепь сходств. Сходство *H* расположено *между Hk-1,* и *Hk,* если *Hk-1 < H < Hk* (*k = 2, …, N)* Сходство *H* расположено *перед B,* если *H < H* . Сходство *H* расположено *после B,* если *HN < H.*

4.1.2.2. Качество локальных сходств

Качество сходства может быть описано т.н. *P*-значением *(P-value,* см. [109, 226], синоним – уровень значимости (см. [4]). Неформально, P-значение

сходства веса S между последовательностями U и V – это вероятность того, что две независимые случайные последовательности той же длины и с теми же статистическими свойствами содержат хотя бы одно сходство веса не менее S. Чем более значимо сходство, тем меньше P-значение.

Для более формального определения того, что такое P-значение, необходимо ввести распределение вероятностей на множестве пар последовательностей длин $|U|$ и $|V|$ соответственно. Вместо этого мы будем считать, что задана функция *Pval(S, L1, L2)* такая, что

1) $0 \le P(S, L1, L2) \le 1$;
2) *Pval(S, L1, L2)* убывает с ростом S и возрастает с ростом *L1, L2*.

Определение 4.1.3. *P-значением* сходства H между последовательностями U и V называется величина *Pval(Score(H), |U|, |V|)*. Если ясно, о каких последовательностях U и V идет речь, то *P-значение* сходства H будет обозначаться *Pval(H)*.

4.1.2.3. Относительная достоверность локальных сходств

Определение 4.1.4. Пусть H – сходство между фрагментами $U[c_1, f_1]$ и $V[c_2, f_2]$; $p \in [0, 1]$; $1 \le b_1 \le c_1 < f_1 \le e_1 \le |U|$; $1 \le b_2 \le c_2 < f_2 \le e_2 \le |V|$. Сходство H называется *p-достоверным относительно* $U[b_1, e_1]$ и $V[b_2, e_2]$, если

$Pval(Score(H), e_1 - b_1 + 1, e_2 - b_2 + 1) < p)$.

Определение 4.1.5. Пусть F – цепь сходств и H – элемент F. Рассмотрим подцепь F_H цепи F, состоящую из всех сходств с весом большим, чем *Score(H)*. Пусть $U[c_1, f_1]$ и $V[c_2, f_2]$ – фрагменты соответственно последовательностей U и V, ограниченные соседними с H элементами цепи F_H. Сходство H называется *p-достоверным относительно F,* если H p-достоверно в $U[c_1, f_1]$, $V[c_2, f_2]$ (см. рис. 4.1.3).

Определение 4.1.6. Цепь сходств F называется *p-достоверной,* если каждое, входящее в F сходство p-достоверно в F.

4.1.2.4. Сравнение множеств локальных сходств

Определение 4.1.7. Пусть $R = \{R_i | i = 1, \ldots, n\}$– набор из n сходств. *Вектор весов* $<R>$ - это вектор $<r_1, r_2, \ldots, r_n>$, содержащий веса сходств R_i, записанные в убывающем порядке. При сравнении векторов сходств мы

используем лексикографическую процедуру. Пусть $\boldsymbol{r} = \langle r_1, r_2, \ldots, r_n \rangle$ и $\boldsymbol{s} = \langle s_1, s_2, \ldots s_m \rangle$ - вектора сходств длины n и m соответственно; $\boldsymbol{r} \neq \boldsymbol{s}$ и k – такое минимальное число, что $r_k \neq s_k$. Если k определено, то $\boldsymbol{r} > \boldsymbol{s} \Leftrightarrow r_k > s_k$. Если k не определено, то $\boldsymbol{r} > \boldsymbol{s} \Leftrightarrow n > m$.

Определение 4.1.8. Пусть *R, Q* – два множества сходств. Множество *R сильнее* множества *R'*, если $\langle R \rangle > \langle R' \rangle$. Цепь *F сильнее* цепи *F',* если множество сходств, входящих в *F* сильнее, чем аналогичное множество для *F'*.

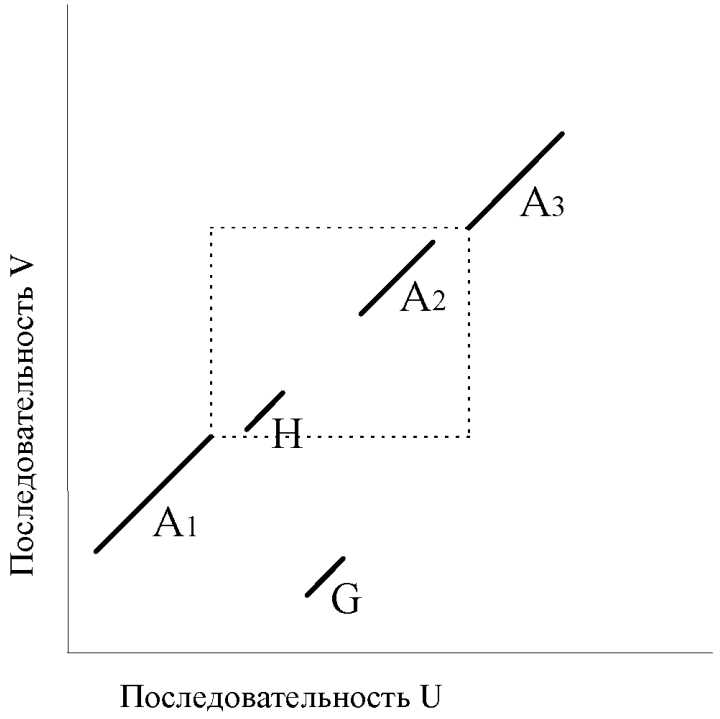


Рис. 4.1.3. Иллюстрация понятия *p*-достоверности сходства относительно цепи. Пусть цепь $F = \{A_1, H, A_2, A_3\}$ и $Score(H) < Score(A2) < Score(A3) < Score(A1)$. *Тогда (1)* $F_H = \{A_1, A_2, A_3\}$, $FA2 = \{A1, A3\}$, $FA3 = \{A1\}$, и *FA1* пусто; (2) A2 *p*-достоверно в F, если в выделенном прямоугольнике $U[End(A1,U)+1, Beg(A3,U)-1] \times V[End(A1,V)+1, Beg(A3,V)-1]$, ограниченном сходствами *A1* и *A3*; (3) *A1 p*-достоверно в *F*, если оно *p*-достоверно относительно последовательностей *U* и *V* в целом.

4.1.2.4. Остовная цепь.

Определение 4.1.7. Пусть *R* – некоторое множество локальных сходств для последовательностей *U* и *V*. Цепь сходств *F* называется *остовной цепью множества сходств R при пороге значимости p* (синоним: *p-остовная* цепь), если (i) *F* является *p*-достоверной;

(ii) каждое сходство, входящее в F, принадлежит *R;*

(iii) никакая цепь, удовлетворяющая (i) и (ii) не является более сильной, чем цепь *F*.

Замечание. На практике бывает удобно ослабить условие (ii) и использовать вместо него условие

(ii') каждое сходство, входящее в F, принадлежит *R* или является подсходством некоторого сходства, принадлежащего *R;*

Ниже мы будем использовать условие (ii), считая, что все «разумные» подсходства сходств из множества *R* тоже входят в множество *R.* Возможность динамического построения подсходств обсуждается в п. 4.1.5.

Определение 4.1.8. *Остовная цепь при пороге значимости p (p-остовная цепь)* последовательностей *U* и *V* – это *p*-остовная цепь для множества всех локальных сходств последовательностей *U* и *V.*

4.1.3. Алгоритмы: общее описание.

В разделах 4.1.4, 4.1.6 представлены два алгоритма, которые строят *p*-остовную цепь *Z(U, V, p)* для заданных последовательностей *U, V* при заданном пороге значимости *p.* Кроме собственно остовной цепи, алгоритмы строят множество *Conf(U, V, p),* состоящее из значимых локальных сходств, найденных при построении цепи *Z(U, V ,p),* но не включенных в нее из-за наличия конфликтов. Построение множества *Conf(U, V, p)* важно при практическом использовании метода (см. [242, 244]): пользователь может просмотреть его и убедиться, что при построении остовной цепи ничего существенного не потеряно. Алгоритмически построение множества *Conf(U, V, p)* трудностей не представляет. Чтобы не загромождать текст излишними деталями ниже мы не будем касаться построения множества *Conf(U, V, p).*

При описании алгоритмов построения цепи *Z(U, V, p)* мы будем исходить из следующих предположений. Будем считать, что есть алгоритм *SetLocSim (**int** BegU, BegV, EndU, EndV, **real** p),* который строит все *p*-достоверные локальные сходства для фрагментов *U[BegU, EndU]* и *V[BegV, EndV].* Чтобы упростить изложение, будем считать, что все сходства имеют разные веса. На практике это означает, что при выборе между конфликтующими сходствами равного веса используется некоторый эвристический критерий, например, выбирается первое сходство в лексикографическом порядке.

При построении остовной цепи *Z(U, V, p)* возможны две стратегии. Первая стратегия предполагает, что множество *R* всех необходимых сходств

дано (см. алгоритм *Chain,* п.4.1.4*).* Время работы алгоритма *Chain ~N·log(T),* где *N* – количество сходств в *R; T* – длина цепи *Z(U, V, p),* т.е. количество сходств в ней. В случае, когда в исходное множество сходств *R* явно не включены допустимые подсходства допустимых сходств, возникает необходимость обрабатывать перекрытия сходств (см. рис. 4.1.1c), которые имеют место при конфликтах. Необходимое для этого время зависит от количества таких перекрытий и их суммарной длины, но, как правило, не превышает *O(NlogN),* см. п.4.1.5.

Альтернативная стратегия состоит в том, чтобы не строить явно множество всех сходств-кандидатов, а действовать иерархически. Эта стратегия реализована в алгоритме *Fractal* (см. п.4.1.6). Этот алгоритм сначала использует алгоритм *SetLocSim* (см. выше) для построения множества R_0 сходств, *p*-достоверных для сравниваемых последовательностей в целом. Затем, используя модифицированный алгоритм *Chain*, отбирает из R_0 сходства, принадлежащие искомой остовной цепи. Далее иерархически заполняются промежутки между найденными элементами остовной цепи. В худшем (и практически невероятном) случае время работы алгоритма *Fractal* совпадает с временем работы алгоритма *Chain*. Как правило, *Fractal* строит остовную цепь за время *c(p)•T,* где *c(p)* зависит только от *p*

4.1.4. Алгоритм *Chain.*

Алгоритм *Chain* изображен на рис. 4.1.4. Входными данными являются последовательности *U, V,* множество локальных сходств *R* и порог уровня значимости (*P-value*) *p;* результатом работы – *p*-остовная цепь для множества сходств *R.*

Алгоритм *Chain* обрабатывает элементы множества *R* в порядке убывания весов. Для очередного сходства *H* требуется ответить на два вопроса: (1) добавлять ли сходство *H* к уже построенной подцепи *B = {A1, ..., As}* искомой остовной цепи? и (2) если да, - в какое место цепи *B* следует вставить сходство *H?*

Алгоритм *Chain* использует два глобальных списка сходств: (i) *CurrentLocSim*, который содержит список сходств, подлежащих обработке в порядке убывания их весов; при инициализации этот список содержит все сходства из *R*, и (ii) *BackboneChain,* который содержит остовную цепь для уже

обработанного подмножества множества *R*. При инициализации этот список пуст, в конце работы он содержит искомую остовную цепь.

```
    algorithm Chain (sequence U, V, set of similarities R, real p)
    begin
1.      CurrentLocSim := R;
2.      sort CurrentLocSim by similarities' scores in decreasing order;
3.      BackboneChain  = empty;
4.      while (CurrentLocSim is NOT empty) do begin
5.              H = first(CurrentLocSim);
6.              delete H from CurrentLocSim;
                // Определить положение границ сходства H относительно BackboneChain,
                // См. пояснения в тексте
7.1                     BegU = Segment(Beg(H, U), BackboneChain, U),
7.2                     BegV = Segment(Beg(H, V), BackboneChain, V),
7.3                     EndU = Segment(End(H, U), BackboneChain, U),
7.4                     EndV = Segment(End(H, V), BackboneChain, V)
                // Проверить наличие конфликта сходства H с BackboneChain
                // См. пояснения в тексте
8.              NoConflict = (SegmentBegU = SegmentBegV=  SegmentEndU = SegmentEndV)
                                && SegmentBegU is even
                              )
9.              if (NoConflict) then begin
                // Конфликта нет => попытаться вставить H в BackboneChain
                        // 1) K – номер сходства из BackboneChain, после которого
                        //    нужно вставлять H
9.1.                    K := BegU/2;
                        // 2) Вычислить размеры LU и LV прямоугольника между соседними
                        //    сходствами цепи BacknoneChain, в который нужно вставить
                        //    сходство H
9.2.                    FindBoxSize(BegU, LU, LV)
                        // 3) Вычислить уровень значимости сходства H относительно
                        //    прямоугольника со сторонами  LU и LV.
9.3.                    P = P(Score(H), LU, LV);
                        // 4) Если сходство H значимо – включить его в BackboneChain
9.4.                    if (P < p) then
9.5.                            include H to BackboneChain
9.6.                    end
9.7.            end else begin
                // Конфликт есть – требуется обработка подсходств
                // Строятся нужные подсходства и добавляются в CurrentLocSim;
10.                     WorkConflict (H)        // См. рис. 4.1.5.
                end
        end
11.     return Backbone_Chain;
    end
```

Рис. 4.1.4. Алгоритм *Chain.* Функция Segment описана в тексте. Алгоритм *WorkConflict* обсуждается в п. 4.1.5.

Пусть цепь *BackboneChain* содержит T сходств $C_1 < ... < C_T$. Тогда границы проекций сходств из BackboneChain на каждую из последовательностей разбивают каждую из последовательностей U, V на $2T+1$ участка: $F_0(U) = U[1,Beg(C_1, U)-1]$; $F_0(U) = U[Beg(C_1, U), End(C_1, U),]$; и т.д. При этом участки с нечетными номерами соответствуют проекциям сходств из BackboneChain, а участки с нечетными номерами – промежуткам между сходствами. Некоторые из промежутков могут быть пустыми.

При заданных $x \in [1,|U|]$ и $y \in [1,|V|]$ через Segment(x, BackboneChain, U) обозначим номер участка из разбиения, задаваемого цепью BackboneChain на последовательности U, которому принадлежит позиция x. Значение Segment(y, BackboneChain, V) определяется аналогично.

Чтобы обеспечить эффективность работы со списком BackboneChain, на этом списке поддерживается структура 2-3 дерева [24]. Эта структура позволяет за время $O(logT)$, где T – количество сходств в цепи BackboneChain выполнить следующие операции:

1) вычисление Segment(x, BackboneChain, U) и Segment(y, BackboneChain, V) (см. строки 7.1- 7.4 на рис. 4.1.4);
2) включение сходства H в цепь BackboneChain (см. строке 9.5 на рис. 4.1.4).

Оценим время $Time_1$ обработки одного сходства в алгоритме *Chain* без учета обработки его подсходств (строки 5 – 9 на рис. 4.1.4). Пусть T – количество сходств в текущей цепи BackboneChain. Т.к. множество CurrentLocSim; упорядочено, то строки 5, 6 выполняются за фиксированное время. Каждая из строк 7.1 – 7.4 и 9.5 выполняется за время $O(logT)$, см. выше, а каждая из остальных строк – за фиксированное время. Операция *WorkConflict* (см. строку 10) сводится к записи сходства H в множество «отвергнутых» сходств $Conf(U, V, p)$ и выполняется за фиксированное время. Таким образом,

$$Time_1 = O(logT)$$

и, следовательно, для алгоритма *Chain* получаем оценку времени $O(N \bullet logT)$, где N – общее количество рассмотренных локальных сходств.

Более общий случай обработки конфликтов описан ниже.

4.1.5. Обработка конфликтов.

На практике для экономии памяти удобно не строить явно все допустимые сходства, включая и возможные подсходства допустимых сходств, а достраивать необходимые подсходства динамически, по мере потребности. Как видно из описания алгоритма *Chain,* потребность в построении подсходств сходства *H* может возникнуть только при наличии конфликта между сходством *H* и текущей остовной цепью (см. функцию *WorkConflict,* строка 10 рис. 4.1.4).

В наиболее общем виде алгоритм построения подсходств и их включения в множество допустимых сходств приведен на рис. 4.1.5.

```
   algorithm WorkConflict (similarity H, int BegU,  BegV, EndU, EndV)
// Обработка конфликта, включающая построение подсходств
  //    BegU,  BegV, EndU, EndV определяют положение границ сходства H
  //    относительно BackboneChain. См. пояснения в тексте и пп. 7.1 – 7.4 алгоритма Chain
   begin
          // Определить номера первого и последнего промежутка в цепи BackboneChain, в
          //    которые могут быть вставлены подсходства сходства H
1.1.    StartSegmTemp = min {x | x ≥ BegU && x ≥ BegV && x нечетно};
1.2.    FinSegmTemp   = max {x | x ≤ EndU && x ≤ EndV && x нечетно};
1.3.    StartSegm = (StartSegmTemp-1)/2;
1.4.    FinSegm = (FinSegmTemp-1)/2;
          // Цикл по промежуткам
2.      for t= StartSegm to FinSegm do begin
                  // Найти подсходство G сходства H, имеющее наибольший вес
                  //    среди подсходств, лежащих между t-м и (t+1)-м
                  //    сходствами цепи Backbone
2.1.              G = FindBestSubSim(H, t);
2.2.              include G into CurrentLocSim;
        end
   end
```

Рис. 4.1.5. Вариант алгоритма *WorkConflict,* включающий построение подсходств сходства *H.*

Время обработки подсходств (вызов алгоритма *WorkConflict,* см. рис. 4.1.5 и строку 10 на рис. 4.1.4) зависит от количества

$$F(H) = FinSegm\text{-}StartSegm+1$$

(см. строки 1.3, 1.4 на рис. 4.1.5) промежутков между звеньями цепи BackboneChain, в которые могут быть вставлены подсходства сходства *H* (см. строку 2 на рис. 4.1.5) и от времени построения каждого подсходства (см. строку 2.1 на рис. 4.1.5). Операции строк 1.3, 1.4 выполняются за

фиксированное время. Включение каждого подсходства в список CurrentLocSim (см. строку 2.2 на рис. 4.1.5) и поиск мест возможных вставок фрагментов сходства *H* (см. строки 1.1, 1.2) выполняется за время *O(log*|CurrentLocSim|*)*.

Суммарное время выполнения операций FindBestSubSim(H, t), см.строку 2.1, в худшем случае выполняется за время, не превосходящее длины выравнивания *H* [29] и, следовательно, общее время выполнения таких операций не превосходит времени T_{LocSim} построения множества локальных сходств. Фактически, выполнение построения подсходств (операция FindBestSubSim(H, t)) является частью алгоритма построения множества исходных локальных сходств; она переносится в обработку конфликтов для экономии памяти на хранение исходных локальных сходств (см. ниже п. 4.1.6).

Таким образом, время работы алгоритма рис. 4.1.5 определяется средним (агрегированным) значением величины *F(H)*. В общем случае построение нетривиальной оценки этого среднего представляет трудную теоретическую задачу. Однако при естественных ограничениях на множество исходных локальных сходств можно считать, что $F(H) \leq 1$. Действительно, $F(H) > 1$, если для некоторого сходства *C* из цепи BackboneChain обе проекции сходства *C* лежат строго внутри соответствующих проекций сходства *H,* но вес сходства *H* меньше веса сходства *C* (ввиду того, что сходства обрабатываются в порядке убывания весов).

Таким образом, сходство *H* является «ухудшением» своего «подсходства» *C* и его естественно не включать в исходный набор локальных сходств (сравни с определением локального сходства в [167]). Следовательно, вызов алгоритма *WorkConflict* выполняется за время $O(log|CurrentLocSim|) \leq O(N)$. Для общего времени работы алгоритма *Chain* при динамическом построении подсходств это дает оценку времени *O(Nlog(N))*.

4.1.6. Алгоритм Fractal

Определение 4.1.9. *S-ограничение* цепи *Z* (множества сходств *R*) – это подцепь цепи *Z* (соответственно, - подмножество множества сходств *R*), включающая все сходсва с весом не менее *S*.

Утверждение 4.1.1 («утверждение о жадности»). Пусть R – множество сходств последовательностей U и V, все сходства в R имеют различный вес и $p \in [0,1]$.

1. p-остовная цепь для R единственна.

2. Пусть R_S – S-ограничение множества R; F – p-остовная цепь для множества R_S. Тогда F является S-ограничением для p-остовной цепи G для множества R.

Доказательство. 1. Пусть $\boldsymbol{r} = \langle r_1, r_2, \ldots, r_n \rangle$ - вектор весов сходств некоторой p-остовной цепи для множества R. Индукцией по номеру k элемента вектора весов легко показать, что первые k элементов вектора $\boldsymbol{r}$ определены однозначно. В силу условия однозначности весов это означает однозначность выбора сходств p-остовной цепи для множества R.

2. Пусть $\boldsymbol{r} = \langle r_1, r_2, \ldots, r_n \rangle$ - вектор весов p-остовной цепи G для множества R и $\boldsymbol{r'}$ – вектор весов S-ограничения G_S цепи G. Очевидно, вектор $\boldsymbol{r'}$ является началом вектора $\boldsymbol{r}$, т.е. $\boldsymbol{r'} = \langle r_1, r_2, \ldots, r_k \rangle$ для некоторого значения $k \leq n$. Пусть $\boldsymbol{t} = \langle t_1, t_2, \ldots, t_m \rangle$ - вектор весов цепи сходств F. Т.к. F - p-остовная цепь для множества R_S, то $\boldsymbol{t \geq r'}$. При этом невозможно, что для некоторого $i \leq min(m, k)$ выполнено $t_2 > r_i$. Действительно, в этом случае мы получили бы $\boldsymbol{t > r}$, что противоречит тому, что $\boldsymbol{r}$ – вектор весов p-остовной цепи G для множества R. По той же причине (учитывая, что k – наибольшее такое число, что $r_k > S$) невозможно, что $m > k$. Таким образом, $\boldsymbol{t = r'}$, откуда, в силу условия несовпадения весов сходств, следует $F = G_S$. Утверждение доказано.

Замечание. При отсутствии требования несовпадения весов утверждение 4.1.1 может оказаться неверным. Однако с практической точки зрения этими случаями можно пренебречь, кроме того, используя множество отвергнутых локальных сходств $Conf(U, V, p)$ (см. выше 4.1.3.1) можно диагностировать случаи, когда утверждение 4.1.1 не выполняется. Учитывая это, в программе OWEN реализован именно описываемый в настоящем разделе алгоритм *Fractal*.

Алгоритм *Fractal* (см. рис. 4.1.6) основан на утверждении 4.1.1. Пусть S – такое число, что $Pval(S, |U|, |V|) = p$, т.е. S – минимальный вес, обеспечивающий уровень значимости p при сравнении последовательностей U и V целиком. Используя алгоритм *SetLocSim*, строится множество R_S всех

сходств между *U* и *V,* имеющих вес не менее *S.* Затем, используя модифицированный алгоритм *Chain,* строится *p*-остовная цепь K_S для R_S. По утверждению 4.1.1, K_S может быть дополнена до *p*-остовной цепи *K* для *R*. Поэтому далее описанная выше процедура рекурсивно применяется к промежуткам между соседними сходствами уже построенной подцепи искомой остовной цепи.

Отметим, что *Fractal* использует парадигму жадных алгоритмов [24] дважды. Во-первых, утверждение 4.1.1 является «жадным» - оно утверждает, что *S*-ограничение остовной цепи для множества *R* не зависит от элементов *R,* имеющих вес менее *S.* Во-вторых, построение остовной цепи для текущего множества сходств выполняется жадным алгоритмом *Chain.*

```
        algorithm Fractal(input: sequence U, sequence V, real epsilon)
        begin
                // 1. Пролог
1.1.        MaskTransposons          // Маскировать транспозоны и участки малой
                                     // сложности в последовательностях U, V
                                     //  (см.[335]
1.2.        BackboneChain := empty;
1.3         MainBox := <1, length(U), 1, length(V)>;
1.4         WorkBoxList := {MainBox};

                // 2. Основной цикл
            while WorkBoxList is not empty do begin
2.1.            CurrentBox := first (WorkBoxList);
2.2             delete CurrentBox from WorkBoxList;
2.3             ProceedBox (CurrentBox, epsilon);
            end

                // 3. Эпилог
3.1.        return BackboneChain;
        end
```

Рис. 4.1.6. Алгоритм *Fractal.* Подалгоритм *ProceedBox* изображен на рис. 4.1.7.

Определим *область* <b_1, e1, b_2, e_2> как пару фрагментов $U[b_1, e_1]$ и $V[b_2, e_2]$ последовательностей *U* и *V*. Алгоритм *Fractal* (см. рис. 4.1.6) использует два глобальных объекта:

(i) список сходств *BackboneChain* (в итоге он содержит искомую остовную цепь) и

(ii) список непересекающихся областей *WorkBoxList*, который содержит все области, для которых должна быть построена остовная цепь.

В начале работы список *BackboneChain* пуст, а список *WorkBoxList* содержит единственную область *<1, |U|, 1, |V|>,* соответствующую сравнению исходных последовательностей целиком.

При обработке очередной области алгоритм *ProceedBox* (см. рис. 4.1.7) сначала строит соответствующее множество локальных сходств *CurrentSim* (см. вызов *SetLocalSim* в строке 2.1), а затем строит *S*-ограничение CurrentBackboneChain *p*-остовной цепи для этого множества (строка 2.4). Процедура *Chain_R* – это модификация алгоритма *Chain* (см. п. 4.1.4). Единственное отличие состоит в том, что подсходства веса менее *S* не добавляются в список обрабатываемых сходств. При этом CurrentBackboneChain не пусто, поскольку не пусто множество CurrentLocSim. Далее сходства из CurrentBackboneChain включаются в BackboneChain (строка 2.5), а области между этими сходствами включаются в WorkBoxList (строки 2.6 и 2.7). Если CurrentBackboneChain пусто (в текущей области нет p-достоверных сходств), то вызывается специальный алгоритм обработки «пустой области» *ProceedEmptyBox.* Этот алгоритм – функциональный параметр алгоритма. В простейшем случае (соответствующей приведенной выше формальной постановке задачи) никакой обработки не производится. С практической точки зрения некоторая обработка может быть полезна, Например, может быть сделана попытка построить глобальное выравнивание.

Оценим время работы алгоритма *Fractal.* Общее количество обработанный областей не превышает *2T+1*, где *T* – количество сходств в окончательной остовной цепи. Отсюда следует, что вызов процедуры ProceedBox (строка 2.3, Рис. 4.1.4) происходит не более, чем *2T+1* раз, а время $Time_1$ работы алгоритма *Fractal* без учета времени выполнения подалгоритма ProceedBox не превосходит *O(T)*.

Чтобы оценить время работы алгоритма ProceedBox, рассмотрим количество M элементов множества сходств CurrentSim, т.е. количество p-достоверных сходств в текущей области. Величину *M* можно представить в виде

$$M = Mt + Mr$$

где *Mt* – количество «эволюционно верных» (гомологических) сходств в множестве CurrentSim, а *Mr* – число случайных сходств.

Сумма величин *Mt* для всех обработанных областей, по определению, оценивается величиной *O(T)*. Чтобы дать оценку суммы величин *Mr* следует уточнить определение функции *P(S, L1, L2)* , алгоритма *SetLocSim,* а также вероятностную модель. Говоря неформально, т.к. *p* мало, то величина *Mr* для каждой области будет ограничена константой *c(p)*, зависящей от величины *p*, но не от длин сравниваемых фрагментов.

```
    algorithm ProceedBox (box CurrentBox, real p)
    begin
                    // 1. Пролог
          //  Пусть CurrentBox = <b1, e1, b2, e2>, т.е. он  соответствует фрагментам  U[b1, e1] и
          // V[b2, e2]. Последовательности  U_Temp и V_Temp содержат эти фрагменты.
1.1       U_Temp = U[b1, e1];              L_U = length(U_Temp);
1.2.      V_Temp = V[b1, e1];              L_V = length(V_Temp);
          // Найти порог весов S, соответствующий порогу P-значений p.
1.3.      S = min{Score_P (Score, L_U, L_V) ≤ p};

                    // 2. Основная часть
          //  Построить все сходства с P-значением не более p
2.1.      CurrentSim = SetLocalSim(b1, e1, b2, e2, p);
2.2.      if (CurrentSim is NOT empty)
2.3.      then
                    //     Построить S-ограничение CurrentBackboneChain p-остовной цепи
                    // для множества сходств CurrentSim.
2.4.                CurrentBackboneChain= Chain_R(CurrentSim, p, S);
2.5.                include CurrentBackboneChain into Backbone_Chain;
2.6..                     //     Создать список BoxTemp областей, расположенных между
                          // сходствами из CurrentBackboneChain.
                          BoxTemp := SetBoxTemp(CurrentBackboneChain, CurrentBox);
2.7                 include BoxTemp into WorkBoxList;
2.8                 else
                    // Обработать «пустую» область. См. пояснения в тексте.
2.9.                ProceedEmotyBox(CurrentBox);
          end
    end
```

Рис. 4.1.7. Подалгоритм *ProceedBox* алгоритма *Fractal.* Алгоритм Chain_R является модификацией алгоритма *Chain,* описанного в п. 4.1.4 (см. пояснения в тексте). Алгоритм *SetBoxTemp* тривиален и подробно не описывается.

Таким образом, сумма величин *Mr* для всех обработанных областей не превосходит *c(p)•T,* где *c(p)* не зависит от *T*. Таким образом, суммарное время $Time_2$ всех вызовов алгоритма *ProceedBox,* исключая вызовы алгоритмов *Chain_R* и *SetLocSim* не превосходит *c'(p) •T*.

Как указывалось выше, алгоритм *SetLocSim* является функциональным параметром алгоритма *Fractal.* Построение множества локальных сходств –

хорошо изученная проблема и решается за время $O(|U|+|V|+|M|) = O(|U|+|V|+|T|)$, см. выше.

4.2. Алгоритмические задачи, связанные с построением затравок для поиска локальных сходств.

4.2.1. Использование затравок при поиске локальных сходств.

Алгоритмы поиска локальных сходств [200, 251, 31] и др. основаны на фильтрации пространства поиска с помощью предварительного выделения т.н. затравочных сходств. Приведенные ниже определения обобщают определения из работ [74, 206] и ряда других.

Определение 4.2.1. *Затравкой* [seed] будем называть произвольное множество локальных выравниваний (=сходств), эти выравнивания называются *затравочными* выравниваниями.

Определение 4.2.2. Затравка Z *допускает* выравнивание $G = \langle v, w, E\rangle$, если существует затравочное выравнивание $z_0 \in Z$ такое, что z_0 – подвыравнивание выравнивания G.

Простейший пример затравки – это множество точных совпадений заданной длины m. В этом случае затравка состоит из выравниваний вида $\langle u, u, S_0\rangle$, где u – слово длины m, S_0 – тривиальное выравнивание слова с самим собой. Для пары слов (v, w) и затравки π будем писать, что $(v, w) \in \pi$, если некоторое выравнивание слов v и w принадлежит затравке π.

Пусть π – затравка. Назовем слова w', w'' π-эквивалентными, если

$$\forall v((v, w') \in \pi \Leftrightarrow (v, w'') \in \pi).$$

Очевидно, множество $C(\pi, v)$ всех слов w, образующих с данным словом v затравочное сходство в смысле затравки π, есть объединение классов π-эквивалентности. На этом наблюдении основан стандартный способ поиска затравочных сходств для данных последовательностей $v, w.$. Он состоит из двух этапов.

1. Индексирование. Для каждого класса π-эквивалентными Z строится список всех вхождений элементов этого класса в последовательность v (т.н. индексный список).

2. Поиск. Просматриваем последовательность *w* слева направо. Чтобы найти все фрагменты последовательности *v,* образующие затравочное сходство с очередным фрагментом *w[x, x+d],* достаточно просмотреть индексные списки для всех классов π -эквивалентности *Z* таких, что $Z \subseteq C(\pi, w[x, x+d])$.

Предполагается, что индексирование может быть выполнено за время, пропорциональное общей длине индексных списков, а поиск затравочных сходств – за время, пропорциональное их общему количеству. Таким образом, на построение каждого затравочного сходства тратится конечное время.

Предположим, что все (или почти все, см. ниже) *целевые* (т.е. представляющие интерес для исследователя) локальные сходства содержат затравочное подсходство, иными словами, допускаются затравкой. В этом случае возможна следующая двухэтапная схема поиска целевых локальных сходств.

Этап 1. Поиск затравок (см. выше).

Этап 2. Поиск *в окрестности затравок* локальных сходств, представляющих интерес с точки зрения поставленной биологической задачи.

Благодаря первому этапу, на котором эффективно осуществляется фильтрация пространства поиска, алгоритмы, построенные по такой схеме, работают достаточно быстро, их время работы пропорционально общему количеству выделенных затравочных сходств. Платой за это является возможная потеря целевых локальных сходств, которые не содержат ни одного затравочного сходства.

Говоря неформально, затравка характеризуется двумя величинами – *чувствительностью,* т.е. долей значимых локальных сходств, в которых содержатся затравочные сходства, и *избирательностью,* т.е. вероятностью появления затравочного сходства в рамках выбранной вероятностной модели независимых случайных последовательностей.

Вычисление избирательности затравки определяется вероятностью того, что фрагменты независимых случайных последовательностей, которые начинаются в заданных позициях, образуют затравочное сходство. Например, для рассмотренной выше затравки «точное совпадение длины n» и бернуллиевских случайных последовательностей избирательность равна p^n. В

то же время вычисление чувствительности затравки представляет определенную сложность. Вычисление чувствительности затравки означает вычисление вероятности (в рамках заданной *вероятностной модели)* того, что случайное выравнивание из заданного *множества целевых выравниваний* содержит под-выравнивание, принадлежащее затравке. Алгоритмы вычисления чувствительности затравок рассмотрены в разделе 5.2 главы 5.

Упомянутые выше точные совпадения были первым (и длительное время – единственным, см. главу 1) использовавшимся классом затравок. Однако в последние годы был предложен ряд новых классов. Два таких класса рассмотрены ниже

4.2.2. Разреженные затравки.

Определение 4.2.3. [75, 186] *Разреженным термом E* называется список положительных целых чисел $\{p_1, ...,p_d\}$ таких, что $p_1< ...<p_d$ и $p_1=1$. *Размером* терма *E* называется величина $span(E) = p_d$; *весом* терма *E* называется число $weight = d$.

Терм $E =\{p_1, ...,p_d\}$ часто для наглядности представляют словом t_E длины p_d в двухбуквенном алфавите {#, -} таким, что $t_E[i] = '\#' \Leftrightarrow i \in \{p_1, ...,p_d\}$. Символ '-' называют *джокером.*

Определение 4.2.4. Выравнивание $G = <v, w, T>$ называется *безделеционным,* если $|v|=|w\backslash$ и список склеек *T* состоит из $|v|=|w\backslash$ склеек вида (i, i), где $i=1, ...,|v|$.

Определение 4.2.5. Пусть $G = <v, w, T>$ - безделеционное выравнивание и $E =\{p_1, ...,p_d\}$ – разреженный терм. Выравнивание *G соответствует* терму *E,* если

1) $|v|=|w\backslash = p_d$;

2) $\forall j \in \{1, ...d\}(v[k+ p_j+1] = w[k+ p_j+1])$.

Определение 4.2.6. Пусть $E =\{p_1, ...,p_d\}$ – разреженный терм. *Разреженная затравка, задаваемая термом E* – это множество всех выравниваний, соответствующих терму *E.*

Замечание 1. В ряде работ, например в [195] вместо «разреженный терм» и «разреженная затравка, соответствующая терму $E=\{p_1, ...,p_d\}$» говорится просто «разреженная затравка $E=\{p_1, ...,p_d\}$». Ниже мы, если это не будет вести к недоразумениям, тоже будем применять слова «разреженная

затравка» как к затравке (т.е. множеству сходств), задаваемой термом *E,* так и к самому терму *E.* Конец замечания.

Определение 4.2.7. Пусть $A = \{1, 0\}$ и $G = \langle v, w, T \rangle$ - безделеционное выравнивание. *{1, 0} -представлением* выравнивания *G* называется такое слово $G_{A\{1.0\}}$ длины $|v|$ в алфавите *{1, 0},* что $G_{\{1.0\}}[i] = 1 \Leftrightarrow v[i]=w[i]$.

Утверждение 4.2.1. Пусть *G* и *H* – безделеционные выравнивания, причем $G_{\{1.0\}} = H_{\{1.0\}}$ и *E* - разреженная затравка. Затравка *E* допускает выравнивание *G* (см. определение 4.2.2) тогда и только тогда, когда *E* допускает выравнивание *H.*

Доказательство – очевидно.

В этом разделе выравнивания будут нас интересовать только с точки зрения их допустимости затравками определенных видов (в подразделах 4.2.2, 4.2.3 - разреженные затравки, в разделе 4.3 – т.н. классификационные затравки). Поэтому нам будет удобно представлять выравнивания словами в специальных алфавитах – т.н. *алфавитах выравниваний.* Утверждение 4.2.1 показывает, что для разреженных затравок адекватным алфавитом выравниваний является алфавит {#, -}. В частности, с учетом определения 4.2.7 и утверждения 4.2.1, определения 4.2.1 и 4.2.5 можно переформулировать следующим образом.

Определение 4.2.8. Пусть $g = \in\{1, 0\}^m$ - безделеционное выравнивание. Разреженный терм $E = \{p_1, \ldots, p_d\}$ *соответствует* выравниванию *G* в позиции *k,* если $\forall j \in \{1, \ldots d\}(g[k+ p_j+1] =1)$. Терм *E допускает* выравнивание *G,* если *E* соответствует выравниванию *G* в некоторой позиции *k.*

Определение 4.2.9. Выравнивание $g = \in\{1, 0\}^m$ называется *(m, k)*-выравниванием, если в нем есть ровно *k* нулей и *m-k* единиц.

Определение 4.2.10. Затравка *E решает (m, k)-проблему,* если она допускает все *(m, k)*-выравнивания.

Отметим, что вес *wight(E)*, разреженной затравки *E* прямо связан с ее избирательностью (см. подраздел 4.2.1): чем выше вес, тем выше избирательность затравки.

4.2.3. Разреженные затравки с одним джокером.

Фиксируем некоторый класс разреженных затравок, например, затравки имеющие вес не менее данного. Задача конструирования затравок для фильтрации без потерь может быть сформулирована различными способами. Например, можно фиксировать длину сходства *m* и искать в выбранном классе затравку которая решает (m, k)- проблему при наибольшем возможном *k.* Наоборот, можно фиксировать допустимое количество несовпадений *k* и минимизировать длину сходства *m,* при которой затравка уже решает (m, k)-проблему. В этом разделе мы используем второе из приведенных альтернативных определений и представляем решение задачи для класса разреженных затравок с одним джокером. Этот результат был обобщен на случай затравок с несколькими джокерами в [118].

Определение 4.2.11. Пусть *Q* - затравка и *k* – допустимое количество несовпадений. *k-критическая длина* для *Q* – это минимальная длина *m,* при которой *Q* решает *(m, k)*-проблему. Затравка *Q* называется *k-оптимальной* в некотором классе затравок, если она имеет минимальную *k-критическую длину* в этом классе.

Естественными классами затравок являются классы разреженных затравок с ограниченным количеством джокеров, т.е.величины *span(Q)-weight(Q).*

Утверждение 4.2.2. Пусть $d > 0$ – натуральное число. Рассмотрим класс затравок веса не менее *d* с одним джокером. Тогда *k-оптимальной* затравкой будет затравка $\#^{d-r}\text{-}\#^{r}$, где

r = [d/2] , если *k=1*

r = [d/3] , если *k> 1*

где [x] – ближайшее целое к числу x и [n+1/2] = n.

Доказательство. 1. *k=1.* Пусть $Q = \#^{p}\text{-}\#^{q}$, где p+q=d; для определенности будем считать, что p ≥ q. Самое длинное (m, 1)-сходство, которое не распознается затравкой Q – это сходство $1^{p-1}01^{p+q}$. Таким образом, нам нужно минимизировать величину q+p при условиях $p+q = d;\ \ p \geq q$.

2. Пусть $k > 1$. Как и в предыдущем случае рассмотрим затравку $Q = Q = \#^{p}\text{-}\#^{q}$, где p+q=d; для определенности будем считать, что p ≥ q. Пусть *S(k)* - самое длинное слово из $(1*0)^{k}1*$, где k>1, которое не распознается затравкой

Q. Пусть далее $L(k) = |S(k)|$. Отметим, что в силу доказанного выше, $S(1) = 1^{p-1}01^{p+q}$ и $L(1) = 2p+q$.

Легко видеть, что для любого k слово $S(k)$ начинается либо с $1^{p-1}0$, либо с $1^{p+q}0$. Действительно, Пусть $S(k)$ начинается с 1^x0. Тогда $x \le p+q$ (иначе Q распознает S(k)). Далее, если $x < p-1$, то слово $1^{p-1-x}S(k)$ с длиной более $|S(k)|$ не распознается затравкой Q, что противоречит условию. Аналогично, невозможно $p-1 < x < p+q$ (в последнем случае возможно нераспознаваемое затравкой Q слово $S' = 1^{p+q-x}S(k)$ с $|S'| > |S(k)|$. Аналогично, можно показать, что если $S(k)$ начинается либо с $1^{p+q}0$, то оно имеет вид $1^{p+q}01^{q-1}0w$.

Пусть L'(k) – максимальная длина такого слова из $(1^* 0)^k1^*$, которое не распознается затравкой Q и начинается с $1^{q-1}0$. Поскольку вхождение затравки Q в слово вида $1^{q-1}0w$ не может начинаться ранее (q+1)-й позиции (напомним: $q \le p$), то

$$| L'(k)| = p+|L(k)|$$

При этом L'(1)=p+2q, что соответствует слову $1^{p-1}01^{p+q}$). Таким образом, мы имеем следующую рекурсию для $k \ge 2$:

$$L'(k) = q+L(k-1), \qquad (4.2.1)$$

$$L(k) = \max\{p+L(k-1), p+q+1+L'(k-1)\}, \qquad (4.2.2)$$

с начальными условиями

L'(1)=p+2q, L(1)=2p+q.

Рассмотрим отдельно два случая.

1) Пусть $p \ge 2q+1$. Тогда индукцией по k легко показать, что первый член в правой части (4.2.2) всегда будет большим и, следовательно,

$$L(k)=(k+1)p+q, \qquad (4.2.3)$$

и

$$S(k)=(1^{p-1}\,0\,)^k\,1^{p+q}. \qquad (4.2.4)$$

Если $q \le p \le 2q+1$, то по индукции получим:

$$L(k)= (s+1)p+(k+1)q + s, \text{ если } k = 2s;$$

$$L(k)= (s+2)p+kq+s, \text{ если } k = 2s+1, \qquad (4.2.5)$$

и

$$S(k)= (\,1^{p+q}\,0\,1^{q-1}\,0\,)^s\,1^{p+q}\,, \text{ если } k = 2s;$$

$$S(k)= (1^{p-1}\,0\,1^{p+q}\,0\,1^{q-1}\,0)^{\,s}1^{p+q}\,, \text{ если } k = 2s+1.$$

По определению L(k), затравка $Q = \#^p\text{-}\#^q$ распознает любое слово из $(1^* 0)^k 1^*$ длины не менее L(k)+1, и эта граница не улучшаема. Таким образом, нам нужно найти значения p и q, которые минимизируют L(k). Учитывая, что p+q=d, можно заметить, что при $p \geq 2q+1$ значение L(k) (см. (4.2.3)) возрастает по p, а при $p \leq 2q+1$, значение L(k) (см. (4.2.6)) убывает по (p). Следовательно, обе функции для L(k) имеют минимум при p=2q+1. Отсюда, в случае d = 1 (mod 3) L(k) достигает минимума при p=2q+1. Несложные вычисления показывают, что при d=0 (mod 3) минимум достигается при p= 2d/3; q = d/3 и при d=2 (mod 3) минимум достигается при q = (d+1)/3; p = d-q = (2d-1)/3. Суммируя сказанное, получим, что L(k) минимально при q = [d/3]. Утверждение доказано.

Пример 4.2.1. Затравка $\#^4\text{-}\#^2$ – оптимальная в смысле приведенного выше определения среди затравок веса 6 с одним джокером. Это, в частности, означает, что она решает (m, 2) – проблему для всех $m \geq 16$ и это – наилучшая оценка в рассматриваемом классе. Аналогично, эта затравка решает (m, 3) – проблему для всех $m \geq 20$ и это – наилучшая оценка в рассматриваемом классе и т.д.

4.3. Класификационные затравки.

4.3.1 Затравки и индексирование.

Пусть π – затравка. Напомним (см. п. 4.2.1), что слова *w', w''* называются π-эквивалентными, если $\forall v((v, w') \in \pi \Leftrightarrow (v, w'') \in \pi)$. Очевидно, множество $C(\pi, v)$ всех слов *w,* образующих с данным словом *v* затравочное сходство в смысле затравки π , есть объединение классов π-эквивалентности.

Для произвольной разреженной затравки *E*, очевидно, каждый класс $C(E, v)$ состоит ровно из одного класса *E*-эквивалентности (говорят, что разреженные затравки *допускают однозначное индексирование*), что упрощает поиск соответствующих затравочных сходств. В то же время, разреженные сходства различают только два вида отношений между символами – совпадение и несовпадение. В [64] была предложена более гибкая и общая модель – векторные затравки, в которых критерий соответствия между

затравкой и выравниванием построен на интегральной характеристике - сумме вкладов разных позиций. Платой за эту общность является то, что для векторной затравки R класс $C(R, v)$ может состоять из десятков классов R–эквивалентности, что усложняет как поиск, так и индексирование.

Ниже рассмотрены т.н. классификационные затравки (см. [187]), которые занимают промежуточное положение между разреженными и векторными затравками. Они, с одной стороны, могут работать с алфавитом выравниваний произвольного размера (т.е. не обязательно различая только два вида сопоставлений «совпал» - «не совпал»), а, с другой стороны, сохраняют возможность однозначного индексирования. Такая возможность, в частности, бывает необходима при создании программ для специализированных компьютеров [255].

Мы также предлагаем конструкцию, которая по данной классификационной затравке строит т.н. «автомат затравки» - конечный автомат, распознающий множество выравниваний, допустимых этой затравкой. Автомат содержит $O(w \cdot 2^{s-w})$ состояний независимо от размера используемого алфавита выравниваний, здесь s – это длина затравки, w - количество «единичных» позиций затравки (т.е. позиций, требующих обязательного совпадения) Для двоичного алфавита это совпадает с оценкой числа состояний в автомате Ахо-Корасика. Однако, в случае произвольного алфавита выравниваний A оценка размера автомата Ахо-Корасика хуже: $O(w \cdot |A|^{s-w})$. Проведенные компьютерные эксперименты показывают, что даже в случае двоичного алфавита наша конструкция ведет к существенно меньшему количеству состояний, чем конструкция Ахо-Корасика. В главе 5 мы увидим, что размер автомата затравки играет ключевую роль при вычислении чувствительности затравки.

4.3.2 Определения

Фиксируем алфавит последовательностей $П$. Будем, как и раньше, рассматривать только безделеционные выравнивания (общий случай кратко обсуждается в главе 5). Пусть $M = \{(a, a) | a \in П\}$ – множество всех пар-совпадений. Рассмотрим алфавит выравниваний A; мы будем предполагать, что A содержит символ 1, который интерпретируется, как совпадение

Определение. 4.2.12. *Классификационный* алфавит B – это алфавит, каждая буква $b \in B$ которого соответствует непустому подмножеству $П_b \subseteq П \times П$, причем

1) $\forall b \in B$ $(M \subseteq П_b)$ (любая классификационная буква допускает совпадение);
2) В B есть буква #, для которой $П_{\#} = M$ (# требует только совпадений, она ниже называется *единичной* буквой).

Очевидно, алфавит разреженных затравок {#, -} – классификационный, причем джокеру соответствует множество $П \times П$.

Определение. 4.3.1. *Классификационная* затравка – это слово в некотором классификационном алфавите B. Затравка $\pi = b_1 \ldots b_m \in B^m$ допускает фрагмент выравнивания $a_1 \ldots a_m \in (П \times П)^m$, если $\forall i \in \{1, \ldots, m\}$ $a_i \in b_i$. Размером $span(\pi)$ затравки π называется ее длина m, #-весом – количество $sharp(\pi)$ букв '#' среди $b_1, \ldots, b_m$

Определение. 4.3.2. Назовем буквы $x, y \in П \times П$ эквивалентными относительно алфавита B (B-эквивалентными), если $\forall b \in B$ $(x \in П_b \Leftrightarrow y \in П_b)$.

Из определений 4.3.1, 4.3.2 следует, что при использовании классификационных затравок в алфавите B, исходный алфавит выравниваний $П \times П$ можно факторизовать по отношению B-эквивалентности. Полученный алфавит будем называть *алфавитом выравниваний, порожденным* классификационным алфавитом B. При работе с данным классификационным алфавитом B мы все слова будем представлять, как слова в соответствующем алфавите выравниваний A.

Пример 4.3.1. Рассмотрим выравнивания нуклеотидных последовательностей, т.е. выравнивания над алфавитом последовательностей {*A, C, G, T*}. *Транзицией* называется одна из четырех пар *{(A, T), (T, A), (C, G), (G, C) }*. *Трансверсией* называется любое несовпадение, отличное от транзиции. Известно, что среди замен в нуклеотидных выравниваниях частота транзиций существенно выше, чем частота трансверсий. В качестве алфавита затравок возьмем алфавит B = { #, @, -}; элементы B представляют соответственно только совпадение (#), совпадение или транзицию (@), любое сопоставление (-). Очевидно, порожденный алфавитом B алфавит

выравниваний A включает три множества: (1) совпадения (символ: *1*); (2) транзиции (символ*:* *h*); (3) трансверсии, т.е. несовпадения, отличные от транзиций (символ:*0*).

Рассмотрим затравку π = #@-# в алфавите B. Она соответствует фрагментам *g[4, 7]* и *g[6, 9]* выравнивания $g = 10h1h1101 \in A^*$ и, следовательно, допускает g. При этом $span(\pi) = 4$; $sharp(\pi) = 1$. Конец примера.

В отличие от веса разреженных затравок, #-вес классификационных затравок не определяет избирательность затравки. Для того, чтобы, сравнивая затравки, можно было бы сравнивать их избирательности, можно использовать следующее определение (см. [275]).

Пусть B – алфавит классификационных затравок. Будем считать, что каждой букве $c \in B$ сопоставлена вероятность $\beta(c)$. Это можно сделать, например, так. Рассмотрим алфавит последовательностей $П$, связанный с алфавитом B. Пусть на $П$ задано распределение вероятностей $\beta: П \to [0,1]$, где

$$\Sigma_{x \in П} \beta(x) = 1$$

Это распределение индуцирует распределение вероятностей на множестве пар $П \times П$ по правилу $\beta(x, y) = \beta(x)\,\beta(y)$. Так как буквы классификационного алфавита соответствуют подмножествам в $П \times П$, то, тем самым, зная распределение β, каждому классификационному символу c можно приписать вероятность $\beta(c)$. Отметим, что, вообще говоря,

$$\Sigma_{c \in B} \beta(c) \neq 1$$

Определение 4.3.3. Пусть B – алфавит классификационных затравок и каждой букве $c \in B$ сопоставлена вероятность $\beta(c) \in [0, 1]$. Пусть # - единичный символ алфавита B, т.е. символ, соответствующий множеству совпадений $M = \{(x, x) | x \in П\}$. *Весом* произвольного *символа* $c \in B$ называется величина

$$weight(c) = -\log(\beta(c))/\log(\beta(\#))$$

Весом $weight(\pi)$ классификационной *затравки* π называется сумма весов входящих в нее букв.

Утверждение 4.3.1. Пусть $\pi = b_1 \ldots b_r$ – классификационная затравка над алфавитом последовательностей $П$; $E(\pi)$ – множество всех выравниваний

длины *r,* которые соответствуют затравке π. Пусть на *П* задано распределение вероятностей *β,* которое индуцирует бернуллиевское распределение на множестве $(П \times П)^r$ выравниваний длины *r.* Тогда

1. Вероятность множества *E(*π) при этом распределении

$$Prob_{\beta}(E(\pi)) = -log(weight(\pi))/log(\beta(\#))$$

2. Для произвольной буквы $c \in B$ выполнено:

$$0 \le weight(c) \le 1,$$

причем *weight(c)* = 1 тогда и только тогда, когда *c* – это единичный символ '#'; *weight(c)* = 0 тогда и только тогда, когда *c* – это джокер '-'.

Доказательство – очевидно (в п.2 используется, что для произвольного $c \in B$ множество сопоставление *E(c)* включает множество сопоставление *M).*

Пример 4.3.2 (продолжение примера 4.3.1). Пусть на алфавите *{A, C, G, T}* задано равномерное распределение вероятностей. Тогда *weight(#) = 1; weight(@) = ½; weight(-) = 0.* Вес *weight(*π) затравки π = #@-# равен 2.5.

4.3.3. Автомат классификационных затравок.

Фиксируем алфавит выравниваний *A,* алфавит затравок *B.*

Определение 4.3.4. Пусть $\pi == b_1 \ldots b_m \in B^m$ - классификационная затравка длины *m* и #-веса *w.*Через $R_\pi \subseteq \{1, \ldots, m\}$ обозначим множество всех позиций затравки *π,* содержащих символы, отличные от #. Очевидно, $|R_\pi| = r = m\text{-}w$.

Определение 4.3.5. Пусть $\pi == b_1 \ldots b_m \in B^m$ - классификационная затравка длины *m* и #-веса *w.* Через $R_\pi \subseteq \{1, \ldots, m\}$ обозначим множество всех позиций затравки *π,* содержащих символы, отличные от #. Очевидно, $|R_\pi| = m\text{-}w$.

Определение 4.3.6. Пусть $\pi == b_1 \ldots b_m \in B^m$ - классификационная затравка и $g = g_1 \ldots g_p \in A^p$ – выравнивание. Через $t(g) \in \{1, \ldots, m\}$ обозначим длину максимального суффикса вида 1^t в слове *g.* Через $X(g, \pi) \subseteq R_\pi$ - это множество всех таких позиций $r \in R_\pi$, что префикс $b_1 \ldots b_r$ затравки π соответствует некоторому фрагменту $g_{p-t-r+1} \ldots g_{p-t}$ выравнивания *g.*

Пример 4.3.3 (продолжение примеров 4.3.1, 4.3.2). Пусть *B ={#, @, -}* – классификационный алфавит и *A = {1, h, 0}* – соответствующий ему алфавит выравниваний (см. подробнее в примере 4.3.1). Рассмотрим затравку π = #@#-##-### и выравнивание *s = 111h1011h11*. Длина *t* максимального

1-суффикса для *s* равна 2; слово *s'* , которое получается из *s* удалением максимального 1-суффикса – это слово *s'* = . *111h1011h.* Перебором нетрудно убедиться, что некоторый суффикс слова *s'* допускается следующими префиксами затравки π: *π[1, 2]* (см. рис. 4.2.1а) и *π[1, 7]* (см. рис. 4.2.1в). Следовательно, после обработки слова *s = 111h1011h11* классификационный автомат затравки π = #@#-##-### приходит в состояние <*{2, 7}, 2*>. На рис. 4.2.1в показано, что префикс *π[1, 4]* не допускает суффикс длины 4 слова *s'*.

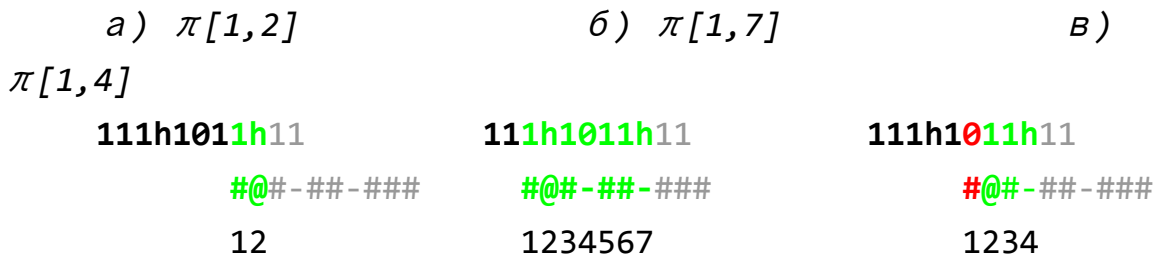


Рис. 4.3.1. Сопоставление префиксов затравки π = #@#-##-### и суффиксов выравнивания *s' = 111h1011h* (из исходного выравнивания *s = 111h1011h11* удален максимальный 1-суффикс «11» длины 2). Префиксы *π[1, 2]* и *π[1, 7]* допускают соответствующие суффиксы (показано зеленым), а префикс *π[1, 4]* – нет (показано красным).

Определение 4.3.7. Пусть $\pi == b_1 \ldots b_m \in B^m$ - классификационная затравка длины *m* и #-веса *w*. Положим $Q = \{\langle X(g, \pi),\ t(g)\rangle | g \in A^* \}$; $q_0 = \langle \varnothing, 0\rangle$, $Q_F = \{\langle X, t\rangle \in Q \mid max(X)+t = m\}$. Пусть $q_F \notin Q$.

Классификационный автомат S_π – это автомат $S_\pi = \langle A, Q_S = (Q - Q_F) \cup \{q_F\}, q_0, \{q_F\}, \psi: Q_S \times A \to Q_S \rangle$, где функция переходов ψ определена следующим образом.

Для любого $a \in A$ выполнено $\psi(q_F, a) = q_F$. Пусть $q = \langle X, t\rangle \in Q - Q_F$. Тогда

при $a=1$: $\psi(\langle X, t\rangle, a) = \langle X, t+1\rangle$, если $\langle X, t+1\rangle \notin Q_F$

$\psi(\langle X, t\rangle, a) = q_F$. – в противном случае

при $a \neq 1$: $\psi((\langle X, t\rangle, a) = \langle Y_{OLD} \cup Y_{NEW}, 0\rangle$, где

$$Y_{OLD} = \{x+t+1 | x \in X \ \&\ a \in b_{x+t+1}\};$$

$$Y_{NEW} = \{1 \le y \le t+1 | a \in b_y\}$$

(если $\langle X, t+1\rangle \notin Q_F$)

$\psi(<X, t>, a) = q_F$. – в противном случае.

Говоря неформально, мы заменяем все исходные «допускающие» состояния $Q_F \subseteq Q$ одним тупиковым допускающим состоянием q_F.

Утверждение 4.3.2. Пусть $\pi == b_1 \ldots b_m \in B^m$ - классификационная затравка длины m и #-веса w. Тогда классификационный автомат S_π распознает множество выравниваний, допустимых затравкой π.

Доказательство. Очевидно, $q_0 = <\varnothing, 0> = <X(\varepsilon, \pi), t(\varepsilon)>$. Из определения функции переходов следует, что

$$\psi(<X(g, \pi), t(g)>, a) = <X(ga, \pi), t(ga)>$$

Поэтому для произвольного выравнивания $g = g_1 \ldots g_p \in A^p$ выполнено:

$$\psi(q_0, g) = <X(g, \pi), t(g)>$$

Далее, если $(q_0, g) = <X(g, \pi), t(g)> \in Q_F$ тогда и только тогда, когда π соответствует суффиксу g длины $span(\pi)$. Это, учитывая, определение функции переходов для заключительного состояния q_F завершает доказательство утверждения.

Утверждение 4.3.3. Пусть $\pi == b_1 \ldots b_m \in B^m$ - классификационная затравка длины m и #-веса w и $S_\pi = < A, Q, q_0, Q_F, \psi: Q \times A \to Q>$ - его классификационный автомат. Тогда количество состояний автомата S_π не превосходит $(w+1) \cdot 2^r$, где $r = |R_\pi|$.

Доказательство. Пусть $R_\pi = \{x_1, \ldots, x_r\}$ и $x_1 < \ldots < x_r$. Пусть Q_i – множество всех таких нетерминальных состояний $<X, t>$, что $max(X) = x_i$., $i = 1, \ldots, r$. Для состояний $<X,t> \in Q_i$ есть 2^{i-1} возможных вариантов для X и $m - x_i$ вариантов для t, а именно: $0, \ldots, m - x_i - 1$ (при больших t получим $max(X)+t \geq m$). Поэтому

$$| Q_i| \leq (m - x_i) \, 2^{i-1} \leq (m - i) \, 2^{i-1} \qquad (4.2.4)$$

и

$$\Sigma_{i=1, r} | Q_i| \leq \Sigma_{i=1, r} (m - i) \, 2^{i-1} = (m-r+1) \, 2^r - m - 1 \qquad (4.2.5)$$

Кроме состояний из множеств Q_i, автомат S_π содержит m состояний вида $<\varnothing, t>$, $t = 0, 1, \ldots, m-1$ и заключительное состояние q_F. Таким образом,

$$|Q_S| \leq (m-r+1) \, 2^r = (w+1) \, 2^r$$

Следствие. Пусть затравка π начинается с символа # (это всегда верно для двоичного алфавита выравниваний $\{\#, -\}$). Тогда $x_i \geq i+1$ и, следовательно,

верхняя оценка в (4.2.4) приобретает вид:

$$| Q_i| \leq(m- i-1) 2^{i-1}$$

что приводит к оценке $|Q_S| \leq w 2^r$. Последняя оценка совпадает с оценкой числа состояний автомата Ахо-Корасика [72].

Замечание 1. Легко построить сюръекцию множества состояний автомата Ахо-Корасика на описанное выше множество состояний классификационного автомата. Таким образом, для любой классификационной затравки π ее классификационный автомат содержит не более состояний, чем соответствующий автомат Ахо-Корасик.

Замечание 2. Алгоритм порождения таблицы переходов автомата S_π, непосредственно следующий из определения, имеет временную сложность $O(r w 2^r |A|)$

Следующее утверждение показывает, что конструкция автомата S_π в общем случае не улучшаема.

Утверждение 4.3.4. Пусть $A = \{\#, -\}$ и $\pi = \#-^r\#$ (между символами # расположено r джокеров). Тогда автомат S_π неприводим, т.е.

(i) любое состояние достижимо из начального состояния q_0;

(ii) никакие два состояния не эквивалентны.

Доказательство.

(i). Пусть $q =< X, t >$ - не допускающее состояние автомата S_π, $X = \{x_1, \ldots, x_k\}$ и $x_1 < \ldots < x_k$. Так как $q =< X, t >$ - не допускающее, то

$$x_k + t < r + 2.$$

Пусть $s \in \{0, 1\}^*$ - выравнивание длины x_k такое, что для всех $i \in [1, x_k]$ выполнено:

$$s_i = 1 \Leftrightarrow \exists j \in [1, k] (i = x_k - x_j + 1) \qquad (4.2.5)$$

Отметим, что $\pi[1] = \#$, следовательно, по построению автомата S_π, $1 \notin X$ и, в силу (4.2.5), $s[x_k] = 0$. Очевидно, $\psi((<\varnothing, 0>, s\bullet 1^t) = q =< X, t >$.

(ii) Пусть $q_1 =< X_1, t_1 >$ и $q_2 =< X_2, t_2 >$ - не заключительные состояния автомата S_π, причем $X_1 \neq X_2$ или $t_1 \neq t_2$ Покажем, что $q_1 =< X_1, t_1 >$ и $q_2 =< X_2, t_2 >$ не эквивалентны. Пусть $X_1 = \{y_1, \ldots, y_a\}$, $X_2 = \{z_1, \ldots, z_b\}$, и $y_1 < \ldots < y_a$; $z_1 < \ldots < z_b$. Предположим, что $max\{X_1\} + t_1 > max\{X_2\} + t_2$ (случай $max\{X_1\} + t_1 < max\{X_2\} + t_2$ рассматривается аналогично) и пусть $d = (r + 2) - (max\{X_1\} + t_1)$. Очевидно, $\psi(q_1, 1^d)$ – заключительное состояние, а $\psi(q_2, 1^d)$ – нет, т.е. в этом

случае утверждение доказано. Пусть теперь $max\{X_1\} + t_1 = max\{X_2\} + t_2$. Для произвольного множества $X \subseteq \{1, ..., r + 1\}$ и числа t, положим

$$X\{t\} = \{v + t | v \in X \text{ и } v + t < r + 2\}.$$

Пусть

$$g = max\{v|(v + t_1 \in X_1 \ \& \ v + t_2 \notin X_2) \ or \ (v + t_2 \in X_2 \ \& \ v + t_1 \notin X_1)\}$$

и пусть $d = r + 1 - g$. Для определенности, пусть

$$g + t_1 \in X_1 \ \& \ g + t_2 \notin X_2$$

Тогда $\psi(q_1, 0^d 1)$ – заключительное состояние автомата S_π, а состояние $\psi(q_2, 0^d 1)$ нет. Это завершает доказательство утверждения.

4.3.4.Компьютерные эксперименты

4.3.4.1. Введение.

В этом разделе представлены результаты двух групп компьютерных экспериментов: (i) по сравнению количества состояний в автоматах Ахо-Корасик и автоматов, построенных по алгоритму п. 4.3.3, см. п. 4.3.4.2 и (ii) по сравнению чувствительности разреженных и классификационных затравок, см. п. 4.3.4.3.

В обоих случаях мы рассматривали два алфавита затравок:

1) $R = \{\#, -\}$ – алфавит разреженных затравок;

2) $B=\{\#, @, -\}$ – алфавит классификационных затравок из примеров 4.2.2, 4.2.3. В соответствии с примером 4.2.3, было принято, что веса букв алфавита B составляют соответственно *1, ½* и *0*.

Для обоих классов затравок были отдельно рассмотрены затравки каждого из весов $w = 9, 10, 11, 12$. При этом для разреженных затравок рассматривались затравки длины не более $w+8$, а для классификационных – затравки длины не более $w+5$ и содержащие не более двух символов @. Эти ограничения носят технический характер и не влияют на качественные результаты компьютерных экспериментов.

4.3.4.2. Количества состояний в распознающих автоматах

В этом разделе представлены результаты компьютерных экспериментов по сравнению количества состояний в автоматах Ахо-Корасик и автоматов, построенных по алгоритму п. 4.3.3. Автомат Ахо-Корасик строился в соответствии с алгоритмом, описанным в [72] для разреженных затравок; обобщение на случай произвольных классификационных затравок тривиально.

Таблицы 4.3.1 и 4.3.2 представляют данные соответственно для разреженных и классификационных затравок (см. п. 4.3.4.1). Для каждого веса *w* было подсчитано среднее количество состояний у автомата Ахо-Корасика (столбцы *«Ахо--Корасик»)* и классификационного автомата, описанного в разделе 4.3.3 (столбцы *«Классиф. авт-т»).* В каждом случае показывалось как среднее количество состояний (столбцы «*Размер»),* так и отношение этого среднего к среднему количеству состояний для соответствующего минимального автомата (столбцы «*Отн»).* Среднее количество состояний для минимального автомата (это число одинаково для обеих конструкций, т.к. соответствующие автоматы эквивалентны) приведено в столбце *«Размер мин авт-т».* Все средние вычислялись по описанным в п. 4.3.4.1 множествам затравок.

Таблица 4.3.1. Разреженные затравки.

Вес	***Ахо-Корасик***		***Классиф. Автомат***		***Размер мин. авт-т***
	Размер	***Отн***	***Размер***	***Отн***	
9	345.94	3.06	146.28	1.29	113.21
10	380.9	3.16	155.11	1.29	120.61
11	415.37	3.25	163.81	1.28	127.62
12	449.47	3.33	172.38	1.28	134.91
13	483.27	3.41	180.89	1.28	141.84

Данные о средних размерах (количестве состояний) автомата Ахо-Корасика, классификационного автомата (см. п. 4.2.4) и минимального автомата, соответствующего этим эквивалентным автоматам. Для автомата Ахо-Корасик и классификационного автомата дополнительно приведены отношения их размеров к размеру минимального автомата. О вычислении средних – см. пояснения в тексте.

Отметим, что наш автомат компактнее автомата Ахо-Корасика не только для 3-буквенного алфавита *B* (что было предсказано в п.4.3.3), но и для двухбуквенного алфавита *R.*

4.3.4.3. Сравнение классификационных и разреженных затравок.

В этом разделе приведены результаты компьютерных экспериментов (см. таблицы 4.2.3, 4.2.4) по сравнению чувствительности разреженных и классификационных затравок (см. п.4.2.4.1). Множество целевых выравниваний включало все безделеционные выравнивания длины 64 в алфавите {A, C, G, T}; выравнивания представляются словами длины 64 в алфавите выравниваний {1, h, 0} (см. п.4.2.1).

Были рассмотрены две вероятностные модели. Первая модель – бернуллиевская; вероятности символов 1, h, 0 индуцированы равномерным распределением символов A, C, G, T в сравниваемых последовательностях, т.е. *Prob(1) = ¼; Prob(h) = ¼; Prob(0) = ½* . Вторая модель – это скрытая марковская модель с тремя состояниями, предложенная в [64]. Эта модель ориентирована на поиск сходств в белок-кодирующих областях и по-разному оценивает вероятности появления нуклеотидов в различных позициях кодонов. Обучение модели проводилось на данных 40 бактериальных геномов, см. [187].

Таблица 4.3.2. Классификационные затравки

Вес	***Ахо-Корасик***		***Классиф. Автомат***		***Размер мин. авт-т***
	Размер	***Отн***	***Размер***	***Отн***	
9	1900.65	15.97	167.63	1.41	119
10	2103.99	16.5	177.92	1.4	127.49
11	2306.32	16.96	188.05	1.38	135.95
12	2507.85	17.42	198.12	1.38	144
13	2709.01	17.78	208.1	1.37	152.29

Данные о средних размерах (количестве состояний) автомата Ахо-Корасика, классификационного автомата (см. п. 4.2.4) и минимального автомата, соответствующего этим эквивалентным автоматам. Для автомата Ахо-Корасик и классификационного автомата дополнительно приведены отношения их размеров к размеру минимального автомата. О вычислении средних – см. пояснения в тексте.

Лучшая по чувствительности затравка для каждого из весов *w = 9, 10, 11, 12* в классах разреженных и классификационных затравок (см. п.4.3.4.1) была найдена прямым перебором с помощью алгоритма представленного в

разделе 5.2. Отметим, что время вычисления чувствительности одной затравки на компьютере Pentium IV 2.4GHz составило от 10 до 500 мс в зависимости от веса и длины затравки, а также от использованной модели. Как и следовало ожидать, классификационные затравки показали более высокую чувствительность.

Таблица 4.2.3.

Вес	Разреженные затравки	Чувствительность	Классификационные затравки, два @	Чувствительность
9	##-##-#-#-###	0.4183	###-#-#@#-@##	0.4443
10	##-##-##-#-###	0.2876	###-@#-@#-#-###	0.3077
11	###-#-#-###-###	0.1906	##@#-##-#-#-@###	0.2056
12	###-#-##-#-##-###	0.1375	##@#-#-##-#@-####	0.1481

Оптимальные затравки в классах разреженных затравок и классификационных затравок в алфавите {#, @, -} (см. пример 4.2.2) с не более, чем двумя символами @ для различных весов (от 9 до 12). Целевые выравнивания имеют длину 64. Распределение вероятностей – бернуллиевское, все символы равновероятны.

Таблица 4.2.4.

Вес	Разреженные затравки	Чувствительность	Классификационные затравки, два @	Чувствительность
9	##-##-##-###	0.435	##@-##-##-##@	0.4456
10	##-##-##-##-##	0.3106	##-##-@##-##@#	0.3173
11	##-##-##-##-###	0.2126	##@#@-##-##-###	0.2173
12	##-##-##-##-####	0.1418	##-@###-##-##@##	0.1477

Оптимальные затравки в классах разреженных затравок и классификационных затравок в алфавите {#, @, -} (см. пример 4.2.2) с не более, чем двумя символами @ для различных весов (от 9 до 12). Целевые выравнивания имеют длину 64. Распределение вероятностей задается скрытой марковской моделью с тремя состояниями, см. описание в тексте.

Глава 5. Использование обобщенных статистических сумм для вычисления вероятностей.

5.0. Введение.

В предыдущих главах были представлены алгоритмы построения парного выравнивания биологических последовательностей, ориентированные на широкий спектр постановок задач. Большинство из этих алгоритмов основано на методе динамического программирования. Как было показано, во Введении, динамическое программирование, наряду с построением оптимальных объектов, позволяет решать, в определенном смысле, двойственную задачу. Это задача вычисления т.н. обобщенных статистических сумм, которые характеризуют множество допустимых объектов в целом.

Настоящая глава, заключительная глава диссертации, посвящена применению обобщенных статистических сумм в задачах биоинформатики. Наиболее распространенной интегральной характеристикой множества объектов является его вероятность. Приведены достаточные условия, при которых вероятность множества символьных последовательностей может быть вычислена методом динамического программирования, описан соответствующий алгоритм вычисления вероятностей. Подход применен к вычислению чувствительностей затравок (см. главу 4 и раздел 5.2) и вероятностей обнаружения сигналов в последовательностях (раздел 5.3).

5.1. Общий метод вычисления вероятностей семейств символьных последовательностей

5.1.1. Введение.

Задача вычисления вероятности множества символьных последовательностей возникает в задачах биоинформатики, как правило, в связи с необходимостью оценки значимости результатов поиска сигналов (паттернов, pattern) некоторого вида. При этом множество, вероятность, которого необходимо найти – это множество всех последовательностей

данной длины m, которые содержат сигнал, превосходящий заданный уровень качества Q. При этом уровень качества сигнала может задаваться как скалярной, так и векторной величиной (см. п. 5.3, сравни также п. 2.3).

Для формальной постановки задачи о вычислении вероятности семейства последовательностей необходимо задать:

1) алфавит A;

2) длину последовательностей m;

3) распределение вероятностей ρ на множестве A^m.

4) множество последовательностей $S \subseteq A^m$;

В настоящем разделе представлен способ сведения задачи вычисления вероятностей к вычислению обобщенной статистической суммы для подходящего графа. Искомый граф строится исходя из конечно-автоматных представлений множества $S \subseteq A^m$ и распределения вероятностей ρ.

Замечание. Иногда с точки зрения биологических приложений бывает полезно считать универсумом не множество всех последовательностей данной длины, а конечное множество U другого вида (см. пример в п. 5.2). В этом случае можно считать, что распределение вероятностей, тем не менее, задано на множестве $A^{m(U)}$ таком, что $U \subseteq A^{m(U)}$, а значимость множества $S \subseteq U \subseteq A^{m(U)}$ оценивать условной вероятностью $Prob(S)/Prob(U)$. Ниже мы всюду считаем, что распределение вероятностей задано на множестве A^m для подходящих алфавита A и длины слова m.

5.1.2. Описание распределения вероятностей.

Будем предполагать, что распределение вероятностей на A^m задано с помощью конечно-автоматного генератора вероятностей (probability transducers), см. [187]. С точки зрения возможностей описания распределений вероятностей на множествах символьных последовательностей конечно-автоматные генераторы эквивалентны скрытым Марковским моделям (Hidden Markov Models, HMM), см. [109].

Говоря неформально, конечно-автоматный генератор вероятностей – это недетерминированный вероятностный автомат без заключительных состояний, который вместе с каждым переходом $q \to q'$, где q – состояние автомата , a – символ используемого алфавита, q' – новое состояние, порождает вероятность появления буквы a при таком переходе. Соответствие

между конечно-автоматными генераторами вероятностей и HMM аналогично соответствию между автоматами Мили и Мура (см. [17]).

Определение 5.1.1. Пусть A – алфавит. Конечно-автоматный генератор вероятностей над алфавитом A – это четверка $G = \langle Q, q^0, A, \rho \rangle$, где Q – множество состояний; q^0- начальное состояние; A - алфавит, $\rho: Q \times A \times Q \to [0, 1]$ – функция генерации вероятностей. При этом,

$$\forall q \in Q \ (\Sigma_{q' \in Q, a \in A} \rho(q, a, q') = 1)$$

Определение 5.1.2. Пусть $G = \langle Q, q^0, A, \rho \rangle$ конечно-автоматный генератор вероятностей над алфавитом A. *Переходом в* G называется тройка $e = \langle q,a,q' \rangle$ такая, что $\rho(q, a, q') > 0$. Буква a называется *меткой* перехода и обозначается $label(e)$. Состояния q и q' называются соответственно *начальным* и *конечным* состоянием перехода и обозначаются соответственно $start(e)$ и $end(e)$. Множество всех переходов в G обозначается $Tr(G)$; множество всех переходов с конечной вершиной q обозначается $Tr(G, q)$.

Определение 5.1.3. Последовательность $P = (e_1, ..., e_n)$ переходов в G называется *путем* в G, если для любого $i \in \{1,..., n-1\}$ выполнено $start(e_{i+1}) = end(e_i)$. *Меткой пути* $P = (e_1, ..., e_n)$ называется слово $label(e_1) .... label(e_n)$. Путь P называется *начальным*, если начальное состояние его первого перехода – это состояние q^0 генератора G. *Вероятностью* $\rho(P)$ пути $P = (e_1, ..., e_n)$ называется произведение $\rho(P) = \Pi_{i=1, n} \rho(e_i)$.

Генератор G называется детерминированным, если для любой пары $q \in Q, a \in A$ есть не более одного перехода вида $\langle q,a,q' \rangle$.

Определение 5.1.4. Пусть $G = \langle Q, q^0, A, \rho \rangle$ конечно-автоматный генератор вероятностей над алфавитом A. Вероятностью слова w относительно генератора G называется величина $P_G(w)$, равная сумме вероятностей всех начальных путей с меткой w; если таких путей нет, то $P_G(w)=0$. Вероятностью конечного набора слов (языка) $L \subseteq A^*$ называется сумма $P_G(L)$ вероятностей всех слов $w \in L$. Очевидно, для любого n выполнено $P_G(A^n) = 1$.

Традиционно используемые способы задания распределений вероятностей на словах могут быть переформулированы в терминах конечно-автоматных генераторов вероятностей. Бернуллиевское распределение (см., например, [195] описывается генератором с одним состоянием. В случае Марковских распределений порядка k [72] вероятность

очередного символа зависит от k предшествующих символов. Такое распределение может быть описано детерминированным генератором с не более, чем $|A|^k$ состояниями.

Распределения, описываемые скрытыми Марковскими моделями (HMM, см. [65], в общем случае, соответствуют недетерминированным генераторам. Множество состояний генератора – это множество состояний исходной HMM, к которым, возможно, добавлено специальное начальное состояние.

Утверждение 5.1.1. Пусть $G= <Q, q^0, A, \rho >$ конечно-автоматный генератор вероятностей над алфавитом A. Тогда по G можно построить скрытую Марковскую модель $H(G)$, которая задает на A^* то же распределение вероятностей, что и генератор G. Обратно, для каждой скрытой Марковской модели H можно построить конечно-автоматный генератор вероятностей $G(H)$, который задает на A^* то же распределение вероятностей, что и модель H.

Доказательство непосредственно следует из определений конечно-автоматного генератора вероятностей HMM (см. [109]).

5.1.3. Конечно-автоматное описание множеств слов.

Множество $S \subseteq A^m$, вероятность которого требуется найти – конечное, и, следовательно, допускается конечным детерминированным автоматом [17] с не более, чем $m|S|$ состояниями. Однако, часто может быть построен распознающий S автомат, имеющий значительно меньшее количество состояний. Как правило, множество S задается описанием вида:

$$S = A^m \cap S' \qquad (5.1.1)$$

где множество $S' \subseteq A^*$ также распознается конечным автоматом.

Определение 5.1.5. Пусть $B = <Q_B, q^0_B, Q^F_B, A, \varphi_B>$ - конечный автомат, распознающий множество $S' \subseteq A^*$; m – натуральное число. Через $B(m)$ обозначим автомат

$$B(m) = <Q_K, q^0_K, Q^F_K, A, \varphi_K>,$$

задаваемый следующим образом:

1) $Q_K = \{< q^0_B, 0>\} \cup Q_B \times \{1, ..., m\} \cup \{ q^T\}$, где q^T – дополнительное
тупиковое состояние;

2) $q^0_K = =< q^0_B, 0>;$

3) $Q^F_K = Q^F_K \times \{m\};$

4) $\varphi_K(q^T, a) = q^T$ для произвольного $a \in A;$

$\varphi_K(<q, k>, a) = < \varphi_B(q, a), k+1 >,$ если $k+1 \leq m;$

$\varphi_K(<q, k>, a) = q^T$ в противном случае.

Отметим, что автомат $B(m)$ может рассматриваться, как декартово произведение автомата B и автомата, распознающего множество A^m. Для количества $|Q_K|$ состояний автомата $B(m)$ выполнено:

$$|Q_K| = |Q_B| \cdot m+2 = O(|Q_B| \cdot m) \qquad (5.1.2)$$

Утверждение 5.1.2. Пусть множество $S' \subseteq A^*$ распознается конечным автоматом B, m – натуральное число. Тогда автомат $B(m)$ распознает множество $S = A^m \cap S'$.

Доказательство – очевидно.

Во многих приложениях множество S' задается следующим условием (здесь L – конечное множество слов в алфавите A):

$$S' = \{w \in A^* | \ \exists u \in L \ (u \text{ является подсловом } w\} \qquad (5.1.3)$$

Условие (5.1.3) мы будем называть условием Ахо-Корасик. В работе [22] описан алгоритм, который по множеству L строит автомат, распознающий множество $S'(L)$, задаваемое условием (5.1.3). Автомат Ахо-Корасик используется в разделе 5.3, там же приведено его подробное описание. Напомним, что в разделе 4.3.3 приведена более экономная конструкция автомата, распознающего множество $S'(L)$ для случая, когда множество L описывается классификационной затравкой.

5.1.4. Формальная постановка задачи. .

Проблема 5.1.1. Даны:

1) алфавит A,

2) длина слов m;

3) конечно-автоматный генератор $G= <Q, q^0, A, \rho >$, задающий распределение вероятностей ρ_G на A^m;

4) язык $L \subseteq A^m$ и конечный детерминированный автомат K, распознающий язык L

Требуется найти вероятность $P_G(L)$ множества L относительно распределения ρ_G.

5.1.5. Идея алгоритма. ВР-автомат.

Решение проблемы 5.1.1 основывается на понятии вероятностного распознающего автомата (probabilistic accepting automaton) или, для краткости, ВР-автомата. Говоря неформально, ВР-автомат – это недетерминированный конечно-автоматный генератор вероятностей, в котором выделено подмножество допускающих состояний. Мы определим декартово произведение автомата без выхода и вероятностного генератора над одним и тем же алфавитом A как ВР-автомат.

Определение 5.1.6. Пусть $K = <Q_K, q_K^0, Q^F_K, A, \varphi>$ - детерминированный конечный автомат без выхода; $G= <Q_G, q^0_G, A, \rho>$ - конечно-автоматный генератор вероятностей над алфавитом A.

ВР-автомат $W = K \times G$ - это пятерка $W= =<Q_W, q^0_W, Q^F_W, A, , \rho_W>$, где

$Q_W = Q_K \times Q_G$;

$q^0_W = q^0_K \times q^0_G$;

$Q^F_{W =} \{<k, g> \in = Q_K \times Q_G \mid k \in Q^F_K\}$;

$\rho_W(<k, g>, a, <k', g'>) = \rho(g, a, g')$, если $\varphi(k, . a) = k'$;

$\rho_W(<k, g>, a, <k', g'>) = 0$ – в противном случае.

Определение 5.1.7. Начальный путь в ВР-автомате называется *полным,* если он заканчивается в допускающем состоянии.

Утверждение 5.1.3. Пусть G – конечно-автоматный генератор вероятностей, $L \subseteq A^m$ – конечный язык и K – ациклический конечный детерминированный автомат, допускающий L. Тогда

1. Граф ВР-автомата $W = K \times G$ ациклический.
2. Вероятность $P_G(L)$ языка L, относительно генератора G равна сумме вероятностей всех полных путей в РА-автомате $W = K \times G$.

Доказательство.

1. Следует из того, что автомат K – ациклический автомат.

2.Т.к. K – детерминированный автомат, то

1) существует взаимно-однозначное соответствие между путями в G и путями в $W = K \times G$, при котором путь в G является проекцией соответствующего ему пути в W на автомат G;

2) для произвольного $w \in A^*$ либо любой путь с пометкой w ведет в допускающее состояние (при $w \in L$), либо ни один

путь с пометкой w не ведет в допускающее состояние (при $w \notin L$).

Таким образом, множество всех путей генератора G с пометками из L находится во взаимно-однозначном соответствии с множеством полных путей из $W=K \times G$, причем вероятности соответственных путей совпадают. Это завершает доказательство утверждения 5.1.3.

Следствие. Вычисление вероятности $P_G(L)$ сводится к вычислению обобщенной статистической суммы (см. главу 1, а также [120]) в графе ВР-автомата W.

Замечание. Т.к. L – конечный язык, то всегда можно построить ациклический конечный автомат, допускающий L.

5.1.6. Сложность алгоритма.

Следствие к утверждению 5.1.2 дает алгоритм вычисления вероятности конечного множества $S \subseteq A^m$, в предположении, что известны генератор вероятностей G и автомат B, распознающий условие множество S.

Утверждение 5.1.4. Рассмотрим алфавит A и конечное множество $L \subseteq A^m$. Пусть $K = <Q_K, q^0_K, Q^F_K, A, \varphi_K>$ - конечный детерминированный автомат, распознающий множество L и $G= <Q_G, q^0_G, A, \rho_G >$ - конечно-автоматный генератор, задающий распределение вероятностей ρ на A^m.

Тогда вероятность $P_G(L)$ множества L относительно распределения ρ_G может быть найдена методом динамического программирования за время $O(|Q_G|^2 \bullet |Q_K| \bullet |A|)$ с использованием памяти $O(|Q_G| \bullet |Q_K|)$.

Доказательство. По утверждению 5.1.2, вероятность $P_G(L)$ может быть вычислена как сумма вероятностей всех полных путей в ВР-автомате $W = K \times G$. Последняя сумма может быть найдена методом динамического программирования (см. [120, 24]) за время $O(|D_W|)$ с памятью $O(|Q_W|)$, где D_W - множество ребер в графе ВР-автомата W, Q_W - множество состояний ВР-автомата W. По построению, ВР-автомат W имеет $|Q_G| \bullet |Q_K|$ состояний и для каждой пары, состоящей из состояния q автомата W и символа $a \in A$ есть не более $|Q_G|$ исходящих из q ребер. Это завершает доказательство утверждения.

Утверждение 5.1.5. Рассмотрим алфавит A и конечное множество $L \subseteq A^m$. Пусть $K = <Q_K, q^0_K, Q^F_K, A, \varphi_K>$ - конечный детерминированный автомат,

распознающий множество L, и $G = \langle Q_G, q^0_G, A, \rho_G \rangle$ - конечно-автоматный детерминированный генератор, задающий распределение вероятностей ρ на A^m.

Тогда вероятность $P_G(L)$ множества L относительно распределения ρ_G может быть найдена методом динамического программирования за время $O(|Q_G| \bullet |Q_K| \bullet |A|)$ с использованием памяти $O(|Q_G| \bullet |Q_K|)$.

Доказательство. Идентично доказательству утверждения 5.1.3 с учетом того, что, в силу детерминированности генератора G, для величины $|D_W|$ верна оценка $|D_W| \leq |Q_G| \bullet |Q_K| \bullet |A|$.

Следствие 1. Пусть распределение вероятностей на множестве A^m – Бернуллиевское. Тогда для времени и памяти вычисления вероятности $P_G(L)$ верны оценки $Time_{Bern} \leq O(|Q_K| \bullet |A|)$ и $Space_{Bern} \leq O(|Q_K|)$

Доказательство следует из того, что в Бернуллиевском случае $|Q_G| = 1$.

Следствие 2. Пусть распределение вероятностей на множестве A^m – это Марковское распределение порядка r. Тогда для времени и памяти вычисления вероятности $P_G(L)$ верны оценки $Time_{Markov} \leq O(|Q_K| \bullet |A^{r+1}|)$ и $Space_{Markov} \leq O(|Q_K| \bullet |A^r|)$

Доказательство следует из того, что в случае Марковского распределения вероятностей порядка r выполнено: $|Q_G| = O(|\bullet|A^r|)$.

5.1.7. Декомпозиция распознающего автомата.

Важный случай, когда оценка утверждений 5.1.4 может быть усилена, описан ниже.

Утверждение 5.1.6. Рассмотрим алфавит A и конечное множество $L \subseteq A^m$. Пусть $B = \langle Q_B, q^0_B, Q^F_B, A, \varphi_B \rangle$ - автомат, распознающий множество L' такое, что $L' \cap A^m = L$ и $G = \langle Q_G, q^0_G, A, \rho_G \rangle$ - конечно-автоматный генератор, задающий распределение вероятностей ρ на A^m. Тогда вероятность $P_\rho(L)$ множества L относительно распределения ρ может быть найдена методом динамического программирования за время $O((|Q_B| \bullet |Q_G|^2 \bullet |A| \bullet m)$ с использованием памяти $O(|Q_B| \bullet |Q_G|)$.

Доказательство. В силу утверждения 5.1.2, автомат $B(m) = \langle Q_K, q^0_K, Q^F_K, A, \varphi_K \rangle$ (см. определение 5.1.5) распознает множество $L = L' \cap A^m$. Количество состояний $|Q_K| = |Q_B| \bullet m + 2 = O(|Q_B| \bullet m)$, откуда следует оценка для времени работы алгоритма. Для доказательства оценки для используемой

памяти потребуется подробнее рассмотреть алгоритм вычисления обобщенной статистической суммы (см. [120]), используемый при вычислении $P_\rho(L)$. Для $k \in Q_K$ и $a \in A$ определим множество *Pred(k)* как

$$Pred(k) = \{k' \in Q_K \mid \exists a \in A\ (\varphi_K(k', a) = k)$$

Напомним, что алгоритм вычисления обобщенной статистической суммы просматривает множество состояний $Q_W = Q_K \times Q_G$ в топологическом порядке. Для каждого состояния $(k, g) \in Q_W$ вычисляется суммарная вероятность $P(k, g)$ всех начальных путей, которые заканчиваются в состоянии (k, g). При вычислении $P(k, g)$ используются ранее вычисленные значения $P(k', g')$ только для таких (k', g'), что $k' \in Pred(k)$.

Назовем r-м слоем в автомате $B(m)$ множество $Layer(B, r)$ состояний вида $<b, r>$, где $b \in Q_B$, $r = 0, 1, \ldots, m$. Из определения автомата $B(m)$ получаем следующие вспомогательные утверждения. Аналогично, r-м слоем в ВР-автомате $W=B(m) \times G$ называется множество всех таких пар (x, g), что $x \in Layer(B, r)$.

(i) Пусть задан топологический порядок $<_B$ на множестве состояний Q_B. Тогда отношение $<_K$ на Q_K, где

$$(b, r) <_K (b', r') \Leftrightarrow (b <_B b') \vee ((b=b') \& (r<r'))$$

задает топологический порядок на Q_K. Иными словами, состояния $(k, g) \in Q_W$ можно обрабатывать послойно.

(ii) Топологический порядок на Q_K вместе с произвольным порядком на Q_G лексикографически задают топологический порядок на Q_W.

(iii) Пусть $k \in Q_K$ и $k \in Layer(B, r)$. Тогда $Pred(k) \subseteq Layer(B, r-1)$.

Из утверждений (i) – (iii) следует, что состояния $(k, g) \in Q_W$ можно обрабатывать послойно и, следовательно, в каждый момент достаточно хранить значения $P(k, g)$ только для двух слоев – текущего и предшествующего. Учитывая, что каждый слой ВР-автомата W содержит $|Q_B| \bullet |Q_G|$ состояний, это дает нужную оценку для используемой памяти. Утверждение 5.1.6 доказано.

Утверждение 5.1.7. Рассмотрим алфавит A и конечное множество $L \subseteq A^m$. Пусть $B = <Q_B, q^0_B, Q^F_B, A, \varphi_B>$ - автомат, распознающий множество L' такое, что $L' \cap A^m = L$ и $G = <Q_G, q^0_G, A, \rho_G>$ - детерминированный конечно-автоматный генератор, задающий распределение вероятностей ρ на

A^m. Тогда вероятность $P_\rho(L)$ множества L относительно распределения ρ может быть найдена методом динамического программирования за время $O((|Q_B|\bullet|Q_G|\bullet|A|\bullet m)$ с использованием памяти $O(|Q_B|\bullet|Q_G|)$.

Доказательство аналогично доказательствам утверждений 5.1.5 и 5.1.6.

Следствие 1. Пусть распределение вероятностей на множестве A^m – Бернуллиевское. Тогда для времени и памяти вычисления вероятности $P_G(L)$ верны оценки $Time_{Bern} \le O(|Q_B|\bullet|A|\bullet m)$ и $Space_{Bern} \le O(|Q_B|)$

Доказательство следует из того, что в Бернуллиевском случае $|Q_G|=1$.

Следствие 2. Пусть распределение вероятностей на множестве A^m – это Марковское распределение порядка r. Тогда для времени и памяти вычисления вероятности $P_G(L)$ верны оценки $Time_{Markov} \le O(|Q_B|\bullet|A^{r+1}|\bullet m)$ и $Spac_{Markov} \le O(|Q_B|\bullet|A^r|)$

Доказательство следует из того, что в случае Марковского распределения вероятностей порядка r выполнено: $|Q_G|=O(|\bullet|A^r|)$.

Последнее утверждение настоящего раздела касается важного специального случая – когда множество S задано условием Ахо-Корасика, а распределение вероятностей – Марковское распределение порядка $r > 1$. В этом случае оценки следствия 2 к утверждению 5.1.7 могут быть существенно улучшены.

Утверждение 5.1.8. Рассмотрим алфавит A и конечное множество $L \subseteq A^m$. Пусть M - конечное множества слов, множество L' – это множество

$L' = \{w \in A^* |$ w содержит подслово из $M\}$

и при этом выполнено

$$L = A^m \cap L'$$

Пусть далее, $B = <Q_B, q^0_B, Q^F_B, A, \varphi_B>$ - автомат Ахо-Корасик для множества M, распознающий множество L' и ρ – распределение вероятностей на A^m, задаваемое Марковской моделью порядка r.

Тогда вероятность $P_\rho(L)$ множества L относительно распределения ρ может быть найдена методом динамического программирования за время $O((|Q_B|+|A^r|)\bullet|A|\bullet m)$ с использованием памяти $O((|Q_B|+|A^r|))$.

Доказательство. Пусть $G= <Q_G, q^0_G, A, \rho_G >$ - конечно-автоматный детерминированный генератор, задающий Марковское распределение

вероятностей ρ на A^m. В силу утверждения 5.1.7, достаточно показать, что слой автомата $B(m) \times G$ содержит $O(|Q_B|+|A^r|)$ вершин.

Согласно п. 5.1.2, $|Q_G| = O(|A^r|)$, причем элементы Q_G – это слова в алфавите A, длиной не более r. Состояния автомата B – это слова $v \in Pref(M)$, см. п. 5.1.3. Поэтому, состояния ВР-автомата $W = B(m) \times G$ – это тройки вида $<v, w, s>$, где $v, w \in A^*$; $s \in \{0,1, ..., m\}$; v – состояние генератора G; $<w, s>$ - состояние автомата K и начальным состоянием ВР-автомата W является состояние $<\varepsilon, \varepsilon, 0>$.

Пусть λ – функция переходов ВР-автомата W. По построению автомата Ахо-Корасик B, генератора G и ВР-автомата W, если для некоторого слова $u \in A^*$ выполнено

$$\lambda(<\varepsilon, \varepsilon, 0>, u) = <v, w, s>,$$

то слова v и w являются суффиксами слова u. Из этого следует, для всякого состояния $<v, w, s>$, достижимого из начального состояния ВР-автомата W выполнено:

$$v \text{ – суффикс } w \text{ или } \quad w \text{ - суффикс } v \qquad (5.1.4)$$

Из (5.1.4) следует, что количество состояний в слое ВР-автомата W ограничено сверху величиной $O(|Q_B|+|Q_G|)$, что и требовалось показать.

Следствие. Пусть $R = \{u_1, ..., u_t\}$ – множество слов такое, что $max\{ u_i \mid i = 1, ..., t\} \le m$; $B = <Q_B, q^0_B, Q^F_B, A, \varphi_B>$ – автомат Ахо-Корасик для слов $\{u_1, ..., u_r\}$ и L – множество всех слов длины m в алфавите A, которые содержат одно из слов $\{u_1, ..., u_r\}$. Пусть на множестве A^m задано Марковское распределение вероятностей ρ порядка r. Тогда вероятность $P_\rho(L)$ может быть вычислена за время время $O((|Q_B|+|A^r|)\bullet|A|\bullet m)$ с использованием памяти $O((|Q_B|+|A^r|)\bullet m)$.

Доказательство непосредственно следует из утверждения 5.1.5.

Отметим, что следствие описывает основной способ применения утверждения 5.1.5.

5.2. Вычисление чувствительности затравок.

5.2.1. Введение.

Определение затравки для поиска локальных сходств приведено в п. 4.3.1. Напомним, что *чувствительностью затравки* называется вероятность (в рамках заданной *вероятностной модели)* того, что случайное выравнивание из

заданного *множества целевых выравниваний* содержит под-выравнивание, принадлежащее затравке. Таким образом, для того, чтобы дать формальную постановку задачи о вычислении чувствительности затравки, необходимо уточнить:

1) класс целевых (т.е. представляющих интерес в рамках рассматриваемой проблемы) выравниваний;
2) способ задания вероятностного распределения на множестве целевых выравниваний;
3) класс используемых затравок («модель затравок»).

Ниже мы приводим соответствующие уточнения, которые позволят свести задачу вычисления чувствительности затравки к приведенному в разделе 5.1 алгоритму вычисления вероятности семейства последовательностей. Перечисленные выше понятия, прежде всего используемая модель затравок, по-разному уточнялись в различных работах, посвященных рассматриваемому вопросу. Отметим, что все приведенные в этих работах алгоритмы основываются на методе динамического программирования; первой работой этого ряда была работа [171].

5.2.2. Представление выравнивания.

Вслед за работами [74, 206] в качестве целевых а, следовательно, и затравочных, выравниваний мы будем рассматривать только безделеционные выравнивания. Этого подхода придерживались все авторы, занимавшиеся данной проблемой (см., например, [65, 72, 83,84, 171]). Безделеционные выравнивания представляются словами над алфавитом выравниваний A. В общем случае $A = П \times П$, где $П$ – алфавит последовательностей. Однако для многих классов затравок алфавит $П \times П$ может быть редуцирован к более простым алфавитам путем факторизации этого алфавита относительно отношения эквивалентности, задаваемому затравкой (т.н. отношение π-эквивалентности, см. определение 4.2.2, п. 4.2.1). Так, для затравок, учитывающих только совпадения, достаточно использовать бинарный алфавит $\{0,1\}$, где 1 соответствует парам вида $\{x, x\}$, $x \in П$, а 0 соответствует всем остальным парам. Если, например, затравка различает транзиции и трансверсии в ДНК [237], то можно рассматривать трехбуквенный алфавит

вида {совпадение, транзиция, трансверсия}. Подробнее см. п. 4.3.2, примеры 4.3.2, 4.3.3.

Как правило (см. [65, 72, 83,84, 171, 206]) в качестве множества целевых выравниваний рассматривается множество всех безделеционных выравниваний фиксированной длины m. Больший или меньший для исследователя интерес тех или иных выравниваний описывается распределением вероятностей на множестве целевых выравниваний. Мы далее рассматриваем именно этот случай. Отметим, тем не менее, два возможных обобщения. Во-первых, описанный ниже подход обобщается на случай целевых выравниваний, содержащих делеции. Для этого достаточно в алфавит $П \times П$ добавить буквы вида $<x, ->$ и $<-, x>$, где $x \in П$, и '-' соответствует удалению символа. Во-вторых, в некоторых работах в качестве множества целевых выравниваний рассмотрены не все выравнивания данной длины, а лишь некоторые из них. Так, в [237], в качестве целевого множества рассматривались {1,0}-последовательности длины m, в которых количество нулей не превосходило заданного порога k (сравни с определением *(m, k)*-проблемы в п. 4.3.2 и определением величины %ID в п.3.1). В подобных случаях чувствительность затравок можно оценивать в терминах условной вероятности и тем самым свести задачу к основному случаю (см. подробнее п. 5.1.1). Отметим, что при вычислении условной вероятности дополнительно используется автомат, распознающий требуемое множество целевых выравниваний.

Таким образом, далее множество целевых выравниваний представляется множеством всех слов заданной длины m в некотором алфавите A, который называется алфавитом *выравниваний.*

5.2.2. Формальная постановка задачи.

Фиксируем алфавит последовательностей П, связанный с ним алфавит выравниваний A и длину m целевых выравниваний. Множество целевых выравниваний, как сказано выше, представляется множеством A^m. Будем считать, что задан конечно-автоматный генератор вероятностей $G = <Q_G, q^0_G, A, \rho>$, определяющий распределение вероятностей на множестве A^m.

Пусть π – затравка, т.е. множество выравниваний, каждое из которых описывается словом в алфавите *A;* эти выравнивания будут называться затравочными выравниваниями.

Пример 5.2.1. Пусть П – это алфавит ДНК $Д = \{a, c, g, t\}$. Рассмотрим затравку $\pi = \{ <v, w, E> | E = \{(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7)\}; v, w \in П^7; v[i]=w[i]$ для $i \in \{1, 2, 3, 4, 6,7\}$. Затравочные выравнивания – это все выравнивания вида $z_1z_2z_3z_4z_5z_6z_7$, где $z \in \{(d, d) | d \in Д\}_i$, $(i \in \{1,2,3,4,6,7\}; z_5 \in Д^2$. Затравка π относится к классу разреженных затравок (см. п.4.3.2). Она может быть описывается термом {*1,2,3,4,6,7*} или словом ####-## в алфавите затравок {#, -}, см. пример 4.3.1.

Определение 5.2.1 Через $S(\pi) \subseteq A^*$ обозначается множество всех выравниваний, допускаемых затравкой π. *Автоматом затравки* π будем называть детерминированный конечный автомат без выхода, распознающий множество $S(\pi)$.

Через $S(\pi, m) \subseteq A^m$ будет обозначаться множество всех выравниваний, допускаемых затравкой π

Будем считать, что вместе с затравкой π задан и ее автомат $B(\pi) = <Q_B, q_B^0, Q^F_B, A, \varphi_B>$. Как отмечалось в п. 5.1.1, такой автомат для всех известных классов затравок может быть построен как автомат Ахо-Корасик [22], специальная конструкция автомата для классификационных затравок представлена в п.4.3.4.

Конечный автомат, распознающий множество всех выравниваний, которые допускаются данной разреженной затравкой был использован в [72] для вычисления чувствительности разреженных затравок для Бернуллиевских целевых выравниваний. Представленный в этом разделе алгоритм можно рассматривать как обобщение алгоритма [72] на случай произвольных затравок и произвольных распределений вероятностей.

Таким образом, задача о вычислении чувствительности затравки может быть сформулирована следующим образом.

Проблема 5.2.1. Дано:

1) алфавит выравниваний *A;*

2) длина целевых выравниваний *m;*

3) конечно-автоматный генератор $G= <Q_G, q^0_G, A, \rho_G>$, задающий распределение вероятностей ρ на множестве A^m;

4) затравка π и соответствующий ей автомат затравки $B(\pi) = <Q_B, q_B^0, Q^F_B, A, \varphi_B>$.

Требуется найти вероятность множества $S(\pi, m)$ относительно распределения вероятностей ρ.

Решение. Проблема 5.2.1. сводится к проблеме вычисления вероятности семейства слов (проблема 5.1.1). Для этого достаточно по автомату $B(\pi)$ построить конечный детерминированный автомат K, распознающий множество $S(\pi, m)$. Для этого достаточно построить декартово произведение автоматов $K= B(\pi) \times T_m$, где T_m, - это автомат, распознающий множество A^m. Очевидно,

$| T_m | \le m$ и, следовательно,

$$| K | \le m | B(\pi) \qquad (5.2.1)$$

Таким образом, утверждения 5.1.4 – 5.1.8 (см. п. 5.1.6) могут быть переформулированы, как утверждения о сложности алгоритмов вычисления чувствительности затравок. Эти утверждения приведены ниже.

Утверждение 5.2.1.Пусть π – затравка в алфавите последовательностей $П$, A – алфавит выравниваний, согласованный с затравкой π и $B(\pi) = <Q_B, q_B^0, Q^F_B, A, \varphi_B>$ - автомат затравки π. Пусть далее $G= <Q_G, q^0_G, A, \rho_G>$ - конечно-автоматный генератор, задающий распределение вероятностей ρ на A^m.

Тогда чувствительность $Sens(\pi, m, \rho)$ затравки π относительно множества выравниваний A^m и распределения ρ может быть найдена методом динамического программирования за время $O(|Q_G|^2 \bullet |Q_B| \bullet m \bullet |A|)$ с использованием памяти $O(|Q_G| \bullet |Q_B|)$.

Доказательство. По определению, $Sens(\pi, m, \rho) = P_\rho(S(\pi, m))$. Автомат $K = <Q_K, q^0_K, Q^F_K, A, \varphi_K>$, распознающий множество $S(\pi, m, \rho)$ имеет не более $|Q_B| \bullet m$ состояний (см. (5.2.1)). Теперь утверждение непосредственно следует из утверждения 5.1.3.

Утверждение 5.2.2. Пусть π – затравка в алфавите последовательностей $П$, . A – алфавит выравниваний, согласованный с затравкой π и $B(\pi) = <Q_B, q_B^0, Q^F_B, A, \varphi_B>$ - автомат затравки π. Пусть далее $G= <Q_G, q^0_G, A, \rho_G>$ -

детерминированный конечно-автоматный генератор, задающий распределение вероятностей ρ на A^m.

Тогда чувствительность $Sens(\pi, m, \rho)$ затравки π относительно множества выравниваний A^m и распределения ρ может быть найдена за время $O(|Q_G|\bullet|Q_B|\bullet m\bullet|A|)$ с использованием памяти $O(|Q_G|\bullet|Q_B|)$.

Доказательство. Следует из утверждения 5.1.4 по аналогии с доказательством утверждения 5.2.1.

Следствие 1. Пусть распределение вероятностей на множестве A^m – Бернуллиевское. Тогда для времени и памяти вычисления чувствительности $Sens(\pi, m)$ верны оценки $Time_{Bern} \leq O(|Q_B|\bullet m \bullet|A|)$ и $Space_{Bern} \leq O(|Q_B|\bullet m)$

Доказательство следует из того, что в Бернуллиевском случае $|Q_G|=1$.

Следствие 2. Пусть распределение вероятностей на множестве A^m – это Марковское распределение вероятностей ρ порядка r. Тогда для времени и памяти вычисления вероятности $P_G(L)$ верны оценки $Time_{Markov} \leq O(|Q_B|\bullet m \bullet|A^{r+1}|)$ и $Spac_{Markov} \leq O(|Q_B|\bullet m \bullet|A^r|)$

Доказательство следует из того, что в случае Марковского распределения вероятностей порядка r выполнено: $|Q_G|=O(|A^r|)$.

Утверждение 5.2.3.Пусть π – затравка в алфавите последовательностей Π, A – алфавит выравниваний, согласованный с затравкой π и автомат $B(\pi) = <Q_B, q_B^0, Q^F{}_B, A, \varphi_B>$ является автоматом Ахо-Корасик. Пусть далее на множестве A^m задано Марковское распределение вероятностей ρ порядка r.

Тогда чувствительность $Sens(\pi, m, \rho)$ затравки π относительно множества выравниваний A^m и распределения ρ может быть найдена методом динамического программирования за время $O((|Q_B|+|A^r|)\bullet|A|\bullet m)$ с использованием памяти $O((|Q_B|+|A^r|))$.

Доказательство – следует из утверждения 5.1.8.

5.3. Вероятность обнаружения мотивов в случайных последовательностях.

5.3.1. Биологическая мотивация: кластеры регуляторных сайтов.

Белки, регулирующие транскрипцию генов, т.н. *факторы регуляции транскрипции, ФРТ,* связываются с фрагментами ДНК, содержащими

специфичную для данного фактора последовательность нуклеотидов [190]. Как правило, все последовательности, с которыми может связываться данный ФРТ имеют одну и туже длину. Множество таких последовательностей называется *мотивом связывания ФРТ (*сокращенно – *МФРТ),* а место расположения (позиция) МФРТ в геноме – *сайтом связывания ФРТ (*сокращенно – *СФРТ)* подробнее обзор [191]. В геноме область связывания ФРТ расположена перед регулируемым геном и называется *цис-регуляторных модулях (ЦРМ),* см. [52]. Обзор экспериментальных методов распознавания ЦРМ и сайтов связывания ФРТ приведен в [73]; обзоры алгоритмов распознавания мотивов даны в [208, 95]. Отметим, что задача подсчета числа вхождений слов – одна из классических задач алгоритмики текстов («stringology»), см., например, работы [89, 235,236, 49, 232] и монографии [88, 202].

В последние годы было показано (как экспериментально [144, 86], так и биоинформатически [338, 314, 198]), что регуляция транскрипции с помощью ФРТ носит кумулятивный характер. В регуляции может участвовать несколько ФРТ, причем, для каждого из них может быть несколько потенциальных мест связывания (вхождений соответствующих мотивов [68]). Таким образом, *цис-регуляторные модули* могут иметь достаточно сложную структуру [197, 164, 223, 177, 169, 153, 161, 194]. В частности, сайты антагонистических факторов часто перекрываются [210], а сайты синергетических факторов находятся друг от друга на расстоянии, кратном 10 нуклеотидам, что соответствует витку спирали ДНК [212]. ЦРМ называются *гомотипическими,* если они содержат сайты связывания только одного мотива, и *гетеротипическими* в противном случае.

В настоящее время есть достаточно большое количество программ распознавания как гомо-, так и гетеротипических ЦРМ [142, 21, 47, 54, 124,125, 296], все они основаны на поиске фрагментов генома, в котором перепредставлены соответствующие мотивы. Для практического использования этих программ необходимы средства оценки значимости полученных результатов. Как правило, это делается путем Монте-Карловского моделирования. Для случая гомотипических ЦРМ в [314, 245], была предложена приближенная формула, которая использовалась в программах

FLYENHANCER [211], SCORE [262], и CLUSTER [198]. Однако, эта формула плохо работает для случая перекрывающихся сайтов.

Обзоры работ по определению достоверности найденных мотивов в биологических последовательностях содержатся в работах [58, 333]. Отметим также нашу недавнюю работу [263].

5.3.2. Постановка задачи.

Мы дадим формальную постановку задачи оценки качества найденного кластера СФРТ. Интуитивно, качественный участок последовательности ДНК – это участок, который содержит слишком много сайтов для заданных длины участка и распределения вероятностей.

Ниже в этом разделе мы считаем фиксированным алфавит ДНК $Д=\{a, c, g, t\}$. Все результаты дословно переносятся на случай произвольного алфавита. Как и в разделе 5.2, будем считать, что на $Д^m$ есть распределение вероятностей ρ, которое задается конечно-автоматным генератором $G= <Q_G, q^0_G, A, \rho_G >$. В частности, это распределение может быть Бернуллиевским или Марковским некоторого порядка $r > 0$.

Определение 5.3.1. *Мотив* длины t – это множество слов $M \subseteq Д^t$.

Определение 5.3.2. Пусть M - мотив длины t, $D \in Д^m$ – последовательность ДНК длины m; $m \geq t$. *Вхождение* мотива M в последовательность D – это фрагмент $D[x, x+t-1]$ такой, что . $D[x, x+t-1] \in M$.

Пусть $D[x_1, x_1+t_1-1]$ – вхождение мотива M_1 и $D[x_2, x_2+t_2-1]$ – вхождение мотива M_2. Эти вхождения *пересекаются,* если пересекаются отрезки $[x_1, x_1+t_1-1]$ и $D[x_2, x_2+t_2-1]$.

Определение 5.3.2. Пусть дан набор $\boldsymbol{M} = < M_1 , ..., M_s >$, состоящий из s мотивов M_i длины t_i $(i=1, ..., s)$ и вектор $\boldsymbol{k} = < k_1 , ..., k_s >$, состоящий из s натуральных чисел k_i $(i=1, ..., s)$. Пусть $D \in Д^m$ – последовательность ДНК длины m. Набор мотивов $\boldsymbol{M}$ имеет $\boldsymbol{k}$ *–вхождение* в последовательность D, если для каждого $i \in \{1, ..., s\}$ в D есть не менее k_i (возможно, пересекающихся) вхождений мотива M_i.

Набор мотивов $\boldsymbol{M}$ имеет *точное* $\boldsymbol{k}$ *–вхождение* в последовательность D, если для каждого $i \in \{1, ..., s\}$ в D есть не менее k_i попарно непересекающихся вхождений мотива M_i.

Определение 5.3.3. Пусть дан набор $\boldsymbol{M} = < M_1 , ..., M_s >$, состоящий из s мотивов M_i длины t_i $(i=1, ..., s)$ и вектор $\boldsymbol{k} == < k_1 , ..., k_s >$, состоящий из s натуральных чисел k_i $(i=1, ..., s)$. Введем следующие обозначения:

$F(\boldsymbol{M, k})$ - множество всех последовательностей из $Д^*$, для которых существует $\boldsymbol{k}$ –вхождение набора $\boldsymbol{M;}$

$F(\boldsymbol{M, k,} m)$ - множество всех последовательностей из $Д^m$, для которых существует $\boldsymbol{k}$ –вхождение набора $\boldsymbol{M;}$

Проблема 5.3.1. Дано:

1) длина m участка последовательности ДНК;
2) конечно-автоматный генератор $G= <Q_G, q^0_G, A, \rho_G >$, задающий распределение вероятностей ρ на множестве $Д^m$;
3) набор $\boldsymbol{M} = < M_1 , ..., M_s >$, состоящий из s мотивов M_i длины t_i $(i=1, ..., s)$
4) вектор $\boldsymbol{k} == < k_1 , ..., k_s >$, состоящий из s натуральных чисел k_i $(i=1, ..., s)$.

Требуется найти вероятность $P_\rho(F(\boldsymbol{M, k,} m))$ множества $F(\boldsymbol{M, k,} m)$ относительно распределения вероятностей ρ

Алгоритм решения этой проблемы, т.е. вычисления вероятности $P_\rho(F(\boldsymbol{M, k,} m))$, основан на сведении проблемы 5.3.1 к проблеме 5.1.1. Как и в случае проблемы 5.2.1 (см. раздел 5.2) для этого достаточно построить конечный детерминированный автомат, распознающий множества $F(\boldsymbol{M, k})$.

Далее мы считаем фиксированными набор мотивов $\boldsymbol{M} = < M_1 , ..., M_s >$ и вектор с натуральными компонентами $\boldsymbol{k} == < k_1 , ..., k_s >$.

5.3.3. Автоматы, распознающие множества $F(\boldsymbol{M, k})$ и $FT(\boldsymbol{M, k})$.

5.3.3.1. Автомат Ахо-Корасик.

Автоматы, распознающие множества $F(\boldsymbol{M, k})$ и $FT(\boldsymbol{M, k})$ строятся на основе автомата Ахо-Корасик [22] для множества слов $M_0 = M_1 \cup ... \cup M_s$.

Определение 5.3.4. Пусть $M_0 \subseteq A^*$; $x \in Pref(M_0)$. Через $Fin(M_0, x)$ будем обозначать множество $Fin(M_0, x) = \{ v \in M_0 | v$ – суффикс слова $x\}$.

Определение 5.3.5 [22]. Пусть $M \in A^*$ и $Pref(M_0)$ – множество всех префиксов слов из множества M (включая слова из M и пустое слово) *Автомат Ахо-Корасик* для множества слов M – это конечный

детерминированный автомат $C = <Q_C, q^0_C, Q^F_C, A, \varphi_C>$, определенный следующим образом:

1) Q_C – это множество $Pref(M_0)$ всех префиксов слов из множества M_0 *(*включая и слова из M_0*)* (см. рис. 5.3.1) .

2) q^0_C – это состояние, соответствующее пустому слову;

3) заключительные состояния – это состояния, соответствующие словам из *M;*

4) Функция переходов φ_C определена следующим образом. Пусть $v \in Pref(M); a \in A$. Тогда:

$\varphi_C(v, a) = va$, если $va \in Pref(M);$

$\varphi_C(v, a) = max\{z | z$ – суффикс va и $z \in Pref(M)\}$.

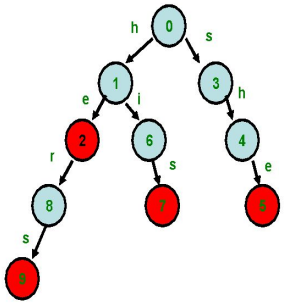


Рис. 5.3.1. Вершины автомата Ахо-Корасик для множества M ={he, she, his, hirs}. Вершины соответствуют элементам множества *Pref(M): 0* – пустое слово; *1 – h; 2 – he; 3 – 3; 4 – sh; 5 – she; 6 – hi; 7 – his; 8 – her; 9 – hers.* Красным (более темным) выделены вершины, соответствующие словам из *M* («заключительные вершины). Для каждой вершины $v \in Pref(M)$ определено множество *Fin(M, v)*, состоящее из принадлежащих *M* суффиксов слова *v* (см. Определение 5.3.4). Для вершины 5 («she") это множество содержит два слова *she* и *he;* для остальных заключительных вершин множество *Fin(M, v)* содержит только слово, соответствующее этой вершине, для незаключительных вершин множество *Fin(M, v)* пусто. (пример взят из статьи [22]).

В работе [22] показано, что выполнено следующее утверждение.

Утверждение 5.3.1 [22]. При работе над словом $z \in A^*$ автомат Ахо-Корасик для множества M проходит через заключительное состояние в такие и только такие моменты времени t, для которых некоторый суффикс $z[1, t]$ лежит в M. В последнем случае $\varphi_C(q^0_C, z)$ совпадает с самым длинным из таких суффиксов.

*5.3.3.2. Автомат, распознающий множество F(**M, k**).*

Опишем теперь автомат $B = <Q_B, q^0_B, Q^F_B, A, \varphi_B>$, распознающий множество $F(\boldsymbol{M, k})$. Неформально говоря, автомат B получается из автомата Ахо-Корасик для множества слов $M_0 = M_1 \cup \ldots \cup M_s$ присоединением s счетчиков, подсчитывающих количество вхождений для каждого из мотивов M_i $(i=1,\ldots,s)$.

Определение 5.3.6. Пусть $\boldsymbol{M} = < M_1, \ldots, M_s >$ - набор мотивов, $\boldsymbol{k} == < k_1, \ldots, k_s >$ вектор с натуральными компонентами и $C = <Q_C, q^0_C, Q^F_C, A, \varphi_C>$ - автомат Ахо-Корасик для множества $M_0 = M_1 \cup \ldots \cup M_s$.

Положим:

$R = Q_C \times\{0,\ldots, k_1\}\times\ldots\times\{0,\ldots, k_s\}$;

$R_F = \{<q, k_1,\ldots, k_s >| q \in Q_C\}$;

Для $v \in Q_C = Pref(M_0)$, $x_i \in \{0, \ldots k_i\}$ $(i = 1, \ldots, s)$ положим:

$\delta_i(v) = 1$, если у слова v есть суффикс, лежащий в M_i,

$\delta_i(v) = 0$ в противном случае.

$\tau_i(v, x_i) = x_i + \delta_i(v)$, если $x_i + \delta_i(v) \le k_i$;

$\tau_i(v, x_i) = k_i$; в противном случае.

Черев $B(\boldsymbol{M, k}) = <Q_B, q^0_B, Q^F_B, A, \varphi_B>$ обозначим такой автомат, что

1) $Q_B = (R - R_F) \cup \{ q^F_B \}$, где q^F_B – новое состояние, не принадлежащее множеству R;

2) $q^0_B = < q^0_C, 0, \ldots, 0>$

3) $Q^F_B = \{ q^F_B \}$

4a) $\varphi_B(q^F_B, a) = q^F_B$ для всех $a \in A$,

4b) Пусть $q \in Q_C$; $a \in A$, $x_i \in \{0, \ldots k_i\}$; $y_i = \tau_i(\varphi_C(q, a), x_i)$;

$d = max\{ k_i - y_i | i=1, \ldots s\}$. Тогда

$\varphi_B(<q, x_1, \ldots, x_s>, a) = < \varphi_C(q, a), y_1, \ldots, y_s>$, если $d \neq 0$;

$\varphi_B(<q, x_1, \ldots, x_s>, a) = q^F_B$ в противном случае.

Утверждение 5.3.2. Автомат $B(\boldsymbol{M, k})$ распознает множество $F(\boldsymbol{M, k})$. Количество состояний автомата $B(\boldsymbol{M, k})$ не превосходит $|Q_C|\bullet k_1 \bullet \ldots \bullet k_s$.

Доказательство следует из утверждения 5.3.1 и приведенного выше определения автомата B.

5.3.4. Вычисление вероятностей.

Вычисление вероятности множества $F(\boldsymbol{M, k, m})$ основывается на утверждениях 5.1.6, 5.1.7 и 5.1.8.

Определение 5.3.6. Пусть $\boldsymbol{M} = < M_1, \ldots, M_s >$ - набор мотивов, $\boldsymbol{k} == < k_1, \ldots, k_s >$. вектор с натуральными компонентами и $C = <Q_C, q^0_C, Q^F_C, A, \varphi_C>$ - автомат

Утверждение 5.3.3. Пусть $\boldsymbol{M} = < M_1, \ldots, M_s >$ - набор мотивов в алфавите A; $\boldsymbol{k} == < k_1, \ldots, k_s >$ - вектор с натуральными компонентами, и $C = <Q_C, q^0_C, Q^F_C, A, \varphi_C>$ - автомат Ахо-Корасик для множества $M_0 = M_1 \cup \ldots \cup M_s$ и $G= <Q_G, q^0_G, A, \rho_G >$ - конечно-автоматный генератор, задающий распределение вероятностей ρ на A^m.

Тогда вероятность $P_\rho(F(\boldsymbol{M, k, m}))$ множества $F(\boldsymbol{M, k, m})$ относительно распределения ρ может быть найдена методом динамического программирования за время $O((|Q_C|\bullet k_1 \bullet \ldots \bullet k_s \bullet m \bullet |Q_G|^2 \bullet |A|)$ с использованием памяти $O(|Q_C|\bullet k_1 \bullet \ldots \bullet k_s \bullet |Q_G|)$.

Доказательство. По утверждению 5.3.2, автомат $B(\boldsymbol{M, k})$ распознает множество $F(\boldsymbol{M, k})$ и содержит $O(|Q_C|\bullet k_1 \bullet \ldots \bullet k_s)$ состояний. С учетом этого, доказываемое утверждение непосредственно следует из утверждения 5.1.6.

Утверждение 5.3.4. Пусть $\boldsymbol{M} = < M_1, \ldots, M_s >$ - набор мотивов в алфавите A; $\boldsymbol{k} == < k_1, \ldots, k_s >$ - вектор с натуральными компонентами, и $C = <Q_C, q^0_C, Q^F_C, A, \varphi_C>$ - автомат Ахо-Корасик для множества $M_0 = M_1 \cup \ldots \cup M_s$ и $G= <Q_G, q^0_G, A, \rho_G >$ - детерминированный конечно-автоматный генератор, задающий распределение вероятностей ρ на A^m.

Тогда вероятность $P_\rho(F(\boldsymbol{M, k, m}))$ множества $F(\boldsymbol{M, k, m})$ относительно распределения ρ может быть найдена методом динамического программирования за время $O((|Q_C|\bullet k_1 \bullet \ldots \bullet k_s \bullet m \bullet |Q_G| \bullet |A|)$ с использованием памяти $O(|Q_C|\bullet k_1 \bullet \ldots \bullet k_s \bullet |Q_G|)$.

Доказательство. Следует из утверждения 5.3.2 (см. доказательство утверждения 5.3.3) и утверждения 5.1.6.

Следствие 1. Пусть распределение вероятностей на множестве A^m – Бернуллиевское. Тогда для времени и памяти вычисления вероятности $P_\rho(F(\boldsymbol{M, k,} m))$ верны оценки $Time_{Bern} \leq O((|Q_C| \bullet k_1 \bullet \ldots \bullet k_s \bullet m \bullet |A|)$ и $Space_{Bern} \leq O(|Q_C| \bullet k_1 \bullet \ldots \bullet k_s)$.

Доказательство следует из того, что в Бернуллиевском случае $|Q_G| = 1$.

Утверждение 5.3.5. Пусть $\boldsymbol{M} = <M_1, \ldots, M_s>$ - набор мотивов в алфавите A; $\boldsymbol{k} == < k_1, \ldots, k_s >$ - вектор с натуральными компонентами, $C = <Q_C, q^0_C, Q^F_C, A, \varphi_C>$ - автомат Ахо-Корасик для множества $M_0 = M_1 \cup \ldots \cup M_s$ и ρ – распределение вероятностей на A^m, задаваемое Марковской моделью порядка r.

Тогда вероятность $P_\rho(F(\boldsymbol{M, k,} m))$ множества $F(\boldsymbol{M, k,} m)$ относительно распределения ρ может быть найдена методом динамического программирования за время $O((|Q_C| + |A|^r) \bullet k_1 \bullet \ldots \bullet k_s \bullet m \bullet |A|)$ с использованием памяти $O(|Q_C| \bullet k_1 \bullet \ldots \bullet k_s + |A|^r)$

Доказательство аналогично доказательству утверждения 5.1.8.

ЗАКЛЮЧЕНИЕ.

Ниже перечислены основные результаты, представленные в настоящей диссертации.

1. На основе конечно-автоматного представления семейств последовательностей и конечно-автоматного описания вероятностных моделей предложен единый подход к вычислению вероятностей сигналов в случайных последовательностях и их выравниваниях. Разработаны методы построения затравок для поиска локальных сходств в нуклеотидных и аминокислотных последовательностях.

2. Предложен алгоритм оптимального парного выравнивания символьных последовательностей при штрафах за делеции, задаваемых кусочно-линейными функциями. Время работы этого алгоритма пропорционально произведению длин последовательностей. Предложен алгоритм построения множества Парето-оптимальных выравниваний символьных последовательностей. Это множество может быть построено за время, не превосходящее $O(m^2 n \log(n))$.

3. Точность алгоритмического выравнивания аминокислотных последовательностей белков относительно выравнивания этих последовательностей, основанных на пространственной структуре белков, лимитируется наличием в структурных выравниваниях участков, имеющих отрицательный вес. Это ограничение может быть преодолено за счет учета сведений о вторичной структуре сравниваемых белков. Соответствующий алгоритм имеет время работы, пропорциональное произведению длин последовательностей.

4. Предложен алгоритм предсказания внутренних петель при построении оптимальной вторичной структуры РНК с временной сложностью $O(P^* \log^2 n)$, где P – количество потенциальных спариваний нуклеотидов, n – длина последовательности РНК. Предложен алгоритм построения оптимального выравнивания последовательностей РНК с заданной вторичной структурой относительно аффинных штрафов за делеции с временной сложностью $O(m^2 n \log^2(n))$, где m, n – длины сравниваемых последовательностей.

5. Выравнивание геномов может быть проведено, не используя оптимизацию выравнивания в целом относительно какой-либо весовой функции и учитывая только оценки значимости отдельных локальных сходств. Предложен эффективный иерархический алгоритм построения геномного выравнивания, основанный на указанном подходе

ЛИТЕРАТУРА

1. Арлазаров В.Л., Диниц Е.А., Кронрод М.А., Фараджев И.А. Об экономном построении транзитивного замыкания ориентированного графа. //Докл. АН СССР. 1970. Т.134, №3. С.477-478.
2. Астахова Т. В., Олейникова Н.В., Ройтберг М А. Глава 6. Сравнительный анализ информационных биополимеров. //Компьютеры и суперкомпьютеры в биологии. Под.ред.В.Д.Лахно и М.Н.Устинина. Москва-Ижевск. 2002. С. 433-455.
3. Гельфанд М.С. Геномы и эволюция. // Вестник РАН. 2009 Т. 79. С. 411-418
4. Кобзарь А. И. Прикладная математическая статистика. Справочник для инженеров и научных работников.// М.: Физматлит, 2006. —С. 816 .
5. Корзинов О. М., Астахова Т. В., Власов П. К., Ройтберг М. А. Статистический анализ участков ДНК в окрестности сайтов сплайсинга. //Молекулярная биология. 2008. Т.42, №1, С.150-162.
6. Ландау, Л. Д., Лифшиц, Е. М. Статистическая физика. Часть 1. — Издание 3-е, дополненное. //М.: Наука, 1976. С.584 («Теоретическая физика», том V).
7. Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов. //Доклады Академий Наук СССР. 1965. Т. 163, №4. С. 845-848.
8. Литвинов И.И., Лобанов М.Ю., Миронов А.А., Финкельштейн А.В., Ройтберг М.А. Информация о вторичной структуре белка улучшает качество выравнивания. //Молекулярная биология. 2006. Т. 40, №3. С. 533-540.
9. Поляновский, В.О., Ройтберг, М.А., Туманян, В.Г. Новый подход к оценке досто-верности выявления вставок-делеций в парном выравнивании. // Биофизика. 2008. Т. 53, №4. С.533-537.
10. Птицын А.Б. Финкельштейн А.В. Связь вторичной структуры глобулярных белков с их первичной структурой. // Биофизика. 1970. Т. 15, С. 757–767.
11. Ройтберг М.А. Алгоритм определения гомологии первичных структур. // Пущино, НЦБИ. 1984. 24 С. (препринт).
12. Ройтберг М.А. Еще один подход к задаче выравнивания последовательностей: больше сходств, меньше делеций - и никаких весовых коэффициентов. // В: Труды конференции "Геном человека - 93" (Черноголовка, 10 - 12 марта 1993г.). 1993. С. 135
13. Ройтберг М.А.. Сравнительный анализ первичных структур нуклеиновых кислот и белков. // Молекулярная биология. 2004. Т.38 №1, стр. 92-103.
14. Ройтберг М.А. Вычисление вероятностей семейств биологических последовательностей. // Биофизика. 2009. Т.54. № 5. С. 718-724
15. Ройтберг М.А., Астахова Т.В., Гельфанд М.С. Алгоритм высокоспецифичного распознавания белок-кодирующих областей в последовательностях высших эукариот. //Молекулярная биология. 1997. Т. 31, № 1. С. 25-31.
16. Ройтберг М.А., Симеоненков М.Н., Таболина О.Ю. Парето-оптимальные выравнивания символьных последовательностей. //Биофизика. 1998. Т. 44, №4. С. 581-594.
17. Трахтенброт Б.А., Барздинь Я.М. Конечные автоматы. Поведение и синтез. //М., Наука. 1970. С.400.

18. Успенский В.А., Семенов А.Л. Теория алгоритмов: основные открытия и приложения. //М.Наука. 1987. С.288 .
19. Харари Ф. Теория графов.// М. УРСС. 2003.С. 300 .
20. Abagyan R.A, Batalov S. Do aligned sequences share the same fold? //J. Mol. Biol. 1997. Vol. 273, P. 355-368.
21. Aerts S., Loo P.V., Thijs G, Moreau Y, Moor BD. Computational detection of cis -regulatory modules.// Bioinformatics. 2003. Vol.19, P.I5–I14. doi: 10.1093/bioinformatics/btg1052.
22. Aho, A. V., Corasick, M. J. Efficient string matching: An aid to bibliographic search. //Communications of the ACM. 1975. Vol. 18, P.6.
23. Aho, A.V., Hirschberg, D.S., Ullman, J.D. Bounds on the Complexity of the Longest Common Subsequence Problem. //Journal of ACM. 1976. Vol. 23, N.1. P. 1-12.
24. Aho A., Hopcroft J., Ulman J. The design and analysis of computer algorithms // Addison-Wesley, Reading , MA., USA. 1974. P. 470.
25. Alexandrov N.N., Luethy R. Alignment algorithm for homology modeling and threading. //Protein Sci. 1998. Vol. 7, P. 254-258.
26. Altschul S.F. Amino acid substitution matrices from an information theoretic perspective. //Journal of molecular biology.1991. Vol.219, N.3. P. 555–65. PMID 2051488.
27. Altschul S.F. Generalized affine gap costs for protein sequence alignment. //Proteins. 1998. Vol. 32, P. 88-96.
28. Altschul S.F., Bundschuh R., Olsen R., Hwa T. The estimation of statistical parameters for local alignment score distributions. //Nucleic Acids Res. 2001. Vol. 15, N.29. P. 351-361.
29. Altschul S.F., Gish W., Miller W., Myers E., Lipman D.J. Basic local alignment search tool. //J Mol Biol 1990. Vol.215, P.403–410.
30. Altschul S.F, Gish W. Local alignment statistics. // R.F. Doolittle, ed. Methods in Enzimology. Computer methods in macromolecular sequence analysis. Academic Press. 1996. Vol. 266. P. 460 - 480.
31. Altschul S., Madden T., Schaffer 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. 1997. Vol. 25, N. 17. P. 3389-3402.
32. An Y., Friesner R.A. A novel fold recognition method using composite predicted secondary structures. //Proteins. 2002. Vol. 48, P.352–366.
33. Apostolico A., Giancarlo R. Seqeunce Alignment in Molecular Biology. // Journal of Computational Biology. 1998. Vol.5, N. 2 P. 173-196 .
34. Apostolico A., Guerra C. The longest common subsequence problem revisited. // Algorithmica. 1987. Vol.2, P. 315-316.
35. Arratia R., Waterman M. // Advances in Mathematics. 1985. Vol. 55, P. 13.
36. Arslan A.N., Egecioglu O., Pevzner P.A. A new approach to sequence comparison: normalized sequence alignment. //Bioinformatics. 2001. Vol. 17, P. 327-337.
37. Assis R., Kondrashov A. S. Rapid repetitive element-mediated expansion of piRNA clusters in mammalian evolution. // PNAS. 2009. Vol.106, N.17. P. 7079 – 7082.

38. Astakhova T.V., Petrova S.V. , Tsitovich I.I., Roytberg M.A. Recognition of coding regions in genome alignment. // Bioinformatics of Genome Regulation and Structure II. (Eds. N.Kolchanov and R. Hofestaedt) Springer Science+Business Med. 2006. P.3-10.
39. Asthana S., Roytberg M., Stamatoyannopoulos J., Sunyaev S. Analysis of sequence conservation at nucleotide resolution. // PLoS Comput Biol. 2007. Vol.3, N.12 :e254. Epub 2007 Nov 14.
40. Aurora R., Rose G.D. Seeking an ancient enzyme in Methanococcus jannaschii using ORF, a program based on predicted secondary structure comparisons. //Proc. Natl. Acad. Sci. USA. 1998. Vol.95, P. 2818–2823.
41. Backofen R., Chen S., Hermelin D., Landau G.M., Roytberg M.A., Weimann O., Zhang K. Locality and gaps in RNA comparison. // J Comput Biol. 2007. Vol. 14, N 8. P.1074-87.
42. Backofen R., Hermelin D., Landau G.M., Weimann O. Normalized similarity of RNA sequences. //In Proceedings of the 12th symposium on String Processing and Information Retrieval. 2005. P. 360-369.
43. Backofen R., Hermelin D., Landau G.M., Weimann O. Local alignment of RNA sequences with arbitrary scoring schemes. // Proceedings of the 17th annual symposium on Combinatorial Pattern Matching (CPM). 2006. P.211.
44. Backofen R., Will S. Local sequence-structure motifs in RNA. //Journal of Bioinformatics and Computational Biology (JBCB). 2004. Vol. 2, N.4. P.681-698.
45. Bafna V., Muthukrishnan S., Ravi R. Comparing similarity between rna strings. // Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching, LNCS 937. 1995. P.1–16.
46. Bahr A., Thompson J., Thierry J., Poch O. BAliBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. // Nucleic Acids Research. 2001. Vol. 29, N. 1. P. 323–326.
47. Bailey T., Noble W. Searching for statistically significant regulatory modules. //Bioinformatics. 2003. Vol.19, P.II16–II25. doi: 10.1093/bioinformatics/btg1054.
48. Barton, G. J., Sternberg, M. J. E., Evaluation and Improvements in the Automatic Alignment of Protein Sequences. // Prot. Eng., 1987. V.1, P. 89-94
49. Bassino F., Clement J., Fayolle J., Nicodeme P. Counting occurrences for a finite set of words: an inclusion-exclusion approach. // DMTCS Proceedings of the International Conference on Analysis of Algorithms AofA07, Juan-les-Pins,), Discrete Mathematics and Theoretical Computer Science. 2007. P. 12.
50. Bateman A, Birney E. Searching databases to find protein domain organization. //Adv. Protein Chem. 2000. Vol.54. P.137–157.
51. Bellman R. On the theory of dynamic programming, //Proc. Nat. Acad. Sci. U.S.A. 1952. Vol. 38, P. 716-719.
52. Ben-Tabou de-Leon, S, Davidson E.H. Gene regulation: gene control network in development. //Annual Review of Biophysics and Biomolecular Structure Vol. 36, P.191-212
53. Beridze T., Tsirekidze N., Roytberg M.A. On the tertiary structure of satellite DNA. // Biochimie. 1992. Vol.74, N.1. P. 187-194.
54. Berman B, Pfeiffer B, Laverty T, Salzberg S, Rubin G, Eisen M, Celniker S. Computational identification of developmental enhancers: conservation and function of

transcription factor binding-site clusters in Drosophila melanogaster and Drosophila pseudoobscura. //Genome Biol. 2004. Vol.5, R61. doi: 10.1186/gb-2004-5-9-r61.
55. Berman H.M, Westbrook J., Feng Z., Gilliland G., Bhat T.N., Weissig H., Shindyalov I.N., Bourne P.E. The Protein Data Bank. // Nucleic Acids Res. 2000. Vol.28, N.1. P.235-42.
56. Bernat, J.A., Crawford, G.E., Ogurtsov, A.Yu., Collins, F.S., Ginsburg, D., Kondrashov, A.S. Distant conserved sequences flanking endothelial-specific promoters contain tissue-specific DNase-hypersensitive sites and over-represented motif. //Human Molecular Genetics. 2006 Vol.15, N.13. P2098-2105; doi:10.1093/hmg/ddl133
57. Bille P. A survey on tree edit distance and related problems. // Theoretical Computer Science. 2005. V. 337 , P. 217 - 239
58. Blanchette M., Sinha S. Separating real motifs from their artifacts. //Bioinformatics. 2001. Vol.17, P. 30–8.
59. Blin G., Touzet H. How to Compare Arc-Annotated Sequences: The Alignment Hierarchy. // String Processing and Information Retrieval. Springer Berlin / Heidelberg. 2006. LNCS V.4209/2006. P. 291-303.
60. Boeva V. , Clement J., Regnier M., Roytberg M., Makeev V. Exact p-value calculation for heterotypic clusters of regulatory motifs and its application in computational annotation of cis-regulatory modules. // Algorithms for Molecular Biology. 2007. Vol. 2, N.13. P.2-13.
61. Bordo A. and Argos P. Suggestions for "safe" residue substitutions in site-directed mutagenesis. // J. Mol. Biol. 1991. Vol. 244, P. 86–99.
62. Bork P., Koonin E.V. Predicting functions from protein sequences— where are the bottlenecks? // Nat. Genet. 1998. Vol.18, P.313–318.
63. Bourque G., Pevzner P. Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species // Genome Research. 2002. Vol. 12, P. 12:26–36.
64. Brejova B., Brown, D., Vinar T. Vector seeds: an extension to spaced seeds allows substantial improvements in sensitivity and specificity. // In Benson, G., Page, R., eds.: //Proceedings of the 3rd International Workshop in Algorithms in Bioinformatics. Vol. 812 of Lecture Notes in Computer Science., Springer. 2003.
65. Brejova B., Brown D., Vinar T. Optimal spaced seeds for homologous coding regions. //Journal of Bioinformatics and Computational Biology. 2004. Vol. 1, P. 595–610.
66. Brenner S.A, Cohen M.A, Gonnet G.H. Amino acid substitution during functionally constrained divergent evolution of protein sequences. //Protein Eng. 1994. Vol.7, P.1323–1332.
67. Brink M., Verbeet M., de Boer H. 1Formation of the central pseudo- knot in 16S rRNA is essential for initiation of translation. // EMBO J. 1993. Vol.12, P.3987–3996.
68. Brown C.T., Rust A.G., Clarke P.J., Pan Z., Schilstra M.J., De Buysscher T., Griffin G., Wold B.J., Cameron R.A., Davidson E.H., Bolouri H. New computational approaches for analysis of cis-regulatory networks. //Dev Biol. 2002. Vol.246,P..
69. Brown D. Optimizing multiple seeds for protein homology search. //IEEE Transactions on Computational Biology and Bioinformatics. 2005. Vol. 2 , P. 29 – 38.
70. Brudno M, Malde S, Poliakov A, Do CB, Couronne O, Dubchak I, Batzoglou S. Glocal alignment: finding rearrangements during alignment. // Bioinformatics. 2003. Vol.19, N. 1P. P. 54-62.

71. Buhler J. Provably Sensitive Indexing Strategies for Biosequence Similarity Search. //Proc. Sixth Ann. Int'l Conf. Computational Molecular Biology (RECOMB '02). 2002. P. 90-99.
72. Buhler J., Keich U., Sun Y. Designing Seeds for Similarity Search in Genomic DNA. // Proc. Seventh Ann. Int'l Conf. Computational Molecular Biology (RECOMB '03). 2003. P. 67-75.
73. Bulyk M.L. DNA microarray technologies for measuring protein-DNA interactions. //Curr Opin Biotechnol. 2006. Vol. 17, P.422–30. doi: 0.1016/j.copbio.2006.06.015.
74. Burkhardt S., Karkkainen J. One-Gapped q-Gram Filters for Levenshtein Distance. // Proc. 13th Symp. Combinatorial Pattern Matching (CPM '02). 2002. Vol 2373, P. 225-234.
75. Burkhardt S., Karkkainen J., Better Filtering with Gapped q-Grams. // Fundamenta Informaticae. 2003. Vol. 56, N. 1-2. P. 51-70, preliminary version in Combinatorial Pattern Matching 2001.
76. Byers T.M., Waterman M.S. Determining all optimal and nearoptimal solutions when solving shortest path problems by dynamic programming. //Oper Res. 1984. Vol.32, P.1381-1384.
77. Califano A., Rigoutsos I. Flash: A Fast Look-Up Algorithm for String Homology. //Proc. First Int'l Conf. Intelligent Systems for Molecular Biology. 1993. P. 56-64.
78. Carroll H,, Beckstead W., O'Connor T., Ebbert M., Clement M., Snell Q., McClellan D. DNA reference alignment benchmarks based on tertiary structure of encoded proteins. //Bioinformatics. 2007. Vol23, P.2648–2649.
79. Cartwright. R.A.Logarithmic gap costs decrease alignment accuracy. BMC Bioinformatics 2006, 7:527
80. Chen S.-J. RNA Folding: Conformational Statistics, Folding Kinetics, and Ion Electrostatics. // Annu Rev Biophys. 2008. Vol. 37. P. 197–214.
81. Chen S., Wang Z., Zhang K. Pattern matching and local alignment for RNA structures. // Proceedings of the international conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS). 2002. P. 55-61.
82. Chen, W., Sung, W.K. On half gapped seed. //Genome Informatics. 2003. Vol. 14, P. 176–185.
83. Choi K.P., Zeng F., Zhang L. Good Spaced Seeds For Homology Search. //Bioinformatics. 2004. Vol. 20, P. 1053–1059.
84. Choi K., Zhang L. Sensitivity analysis and efficient method for identifying optimal spaced seeds. //Journal of computer and System Sciences. 2004. Vol. 68, P. 22–40.
85. Chou P.Y., Fasman G.D. Conformational parameters for amino acids in helical, beta-sheet, and random coil regions calculated from proteins. //Biochemistry. 1974. Vol.13,P.211–222.
86. Clyde D.E, Corado M.S, Wu X., Pare A., Papatsenko D., Small S. A self-organizing system of repressor gradients establishes segmental complexity in Drosophila. //Nature. 2003. Vol.426, P.849–53. doi: 10.1038/nature02189.
87. Cocke J., Schwartz J.T. Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University. 1970.

88. Crochemore M., Rytter W. Jewels in Stringology. World Scientific Publishing, Hong Kong. 2002.
89. Crochemore M., Stefanov V. Waiting time and complexity for matching patterns with automata. //Information Processing Letters. 2003. Vol.87, N.3. P.119–125.
90. Csuros M. Performing Local Similarity Searches with Variable Length Seed. // Proc. 15th Ann. Combinatorial Pattern Matching Symp. (CPM). 2004. P. 373-387.
91. Csuros M., Ma B. Rapid homology search with neighbor seeds. // Algorithmica. 2007. Vol. 48, N. 2, P. 187–202.
92. Cuff J.A., Barton G.J. Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. //Proteins. 1999. Vol.34, P.508–519.
93. Currey K.M., Sasha D., Shapiro B.A., Wang J., Zhang K. An algorithm for finding the largest approximately common substructure of two trees. // IEEE Transactions on Pattern Analysis and Machine Intelligence. 1998. Vol. 20, N.8. P.889-895.
94. Darling, A.C., Mau, B., Blattner, F.R., Nicole T. Perna: Mauve: Multiple Alignment of Conserved Genomic Sequence With Rearrangements//.Genome Res. 2004. V.14. N 7. P. 1394–1403
95. Das M.K., Dai H-K. A survey of DNA motif finding algorithms. // BMC Bioinformatics. 2007. Vol.8, N.52. P.247.
96. David R., Korenberg M.J, Hunter I.W. 3D-1D threading methods for protein fold recognition. //Pharmacogenomics. 2000. Vol.1, P. 445–455.
97. Dayhoff M. O., Schwartz R. M., Orcutt B. C. A model of evolutionary change in proteins. //Atlas of Protein Sequence and Structure. 1978. Vol. 5, N.3. P. 345–352.
98. Demaine E.D., Mozes S. Rossman B., Weimann O. An O(n3)-time algorithm for tree edit distance. //Technical Report arXiv:cs.DS/0604037, Cornell University. 2006.
99. Di Francesco V., Garnier J., Munson P.J. Protein topology recognition from secondary structure sequences: application of the Hidden Markov Models to the alpha class proteins.// J. Mol. Biol. 1997. Vol. 267, P.446–463.
100. Di Francesco V., Geetha V., Garnier J., Munson P.J. Fold recognition using predicted secondary structure sequences and Hidden Markov Models of protein folds.// Proteins. 1997. Vol.1, P.123–128.
101. Dijkstra E. W. A note on two problems in connexion with graphs. // Numerische Mathematik. 1959.Vol 1, P.. 269–271.
102. Do C.B., Foo C.S., Batzoglou S. A max-margin model for efficient simultaneous alignment and folding of RNA sequences. // Bioinformatics. 2008. Vol. 24, N. 13. P. i68-76.
103. Domingues, F.S., Lackner, P., Andreeva, A., Sippl M.J. Structure-based evaluation of sequence comparison and fold recognition alignment accuracy. // J. Mol. Biol. 2000.Vol. 297, P.1003-1013.
104. Doolittle, R.F. Similar amino acid sequences: chance or common ancestry? //Science. 1981. Vol.214, P.149-159.
105. Doolittle W.F. The nature of the universal ancestor and the evolution of the proteome. // Curr. Opin. Struc. Biol. 2000. Vol.10, P.355-358.
106. Dowell R.D., Eddy S.R. Efficient pairwise RNA structure prediction and alignment using sequence alignment constraints. // BMC Bioinformatics. 2006. Vol. 7, P.400.

107. Dulucq S., Tichi L. : RNA secondary structure comparison: exact analysis of the Zhang-Shasha tree edit algorithm. // Theor. Comput. Sci.2003. Vol. 306, P. 471-484
108. Dumas J.P., Ninio J. Efficient algorithms for folding and comparing nucleic acid sequences // Nucleic Acids Res. 1982. V. 10. P. 197–206.
109. Durbin R., Eddy D., Krogh A., Mitchison G. Pairwise alignment. Biological sequence analyses. Probabilistic models of proteins and nucleic acids. //CambridgeUniversity Press, Cambridge, UK. 1998. P. 12-45.
110. Eckardt N.A. Everything in its place: Conservation of gene order among distantly related plant species. //Plant Cell. 2001. Vol.13, P. 723-725.
111. Eddy S.R. Where did the BLOSUM62 alignment score matrix come from?. //Nature biotechnology. 2004. Vol. 22, N.8. P. 1035–1036.
112. Edgar R. C. MUSCLE: multiple sequence alignment with high accuracy and high throughput. // Nucleic Acids Research. 2004. Vol. 32, N. 5. P. 1792–1797.
113. Edgar R.C. MUSCLE: A Multiple Sequence Alignment Method with Reduced Time and Space Complexity. //BMC Bioinformatics. 2004. Vol. 5, N.1. P.113.
114. Edgar R.C, Batzoglou S. Multiple Sequence Alignment. //Current Opinion in Structural Biology. 2006. Vol. 16, P.368-373.
115. Eppstein, D., Galil Z., Giancarlo R., Italiano G Sparse dynamic programming I: linear cost functions. //J. ACM. 1992. Vol. 39, P. 513 - 522
116. Eppstein, D., Galil Z., Giancarlo R., Italiano G. Sparse dynamic programming II: convex and concave cost functions //J. ACM. 1992. Vol. 39, P. 546 - 567
117. EVA server: http://cubic.bioc.columbia.edu/eva/
118. Farach-Colton M., Landau G., Sahinalp S.C., Tsur D. Optimal spaced seeds for faster approximate string matching. //J. Comput. Syst. Sci. 2007. Vol. 73, N.7. P. 1035-1044 .
119. Fernandez-Baca. D., Srinivasam S. Constructing tile minimization diagram of a two-parameter problem. //Operat. Res. Letters. 1991. Vol. 10, P.87-93.
120. Finkelstein, A.V. Roytberg, M.A. Computation of biopolymers: a general approach to different problems. // BioSystems. (spec. volume "Computer genetics". P.A.Pevzner, M.S.Gelfand, eds.). 1993. Vol.30, P.1–19.
121. Fischel-Ghodsian F., Mathiowitz G., Smith T.F. Alignment of protein sequences using secondary structure: a modified dynamic programming method. //Protein Eng. 1990. Vol. 3, P.577–581.
122. Fischer D., Eisenberg D. Protein fold recognition using sequence-derived predictions. //Protein Sci. 1996. Vol.5, P. 947–955.
123. Fitch W. M., Smith T.F. Optimal sequence alignments. // Proc. Natl. Acad. Sci. USA. 1983. Vol. 80, P. 1382–1386.
124. Frith M, Hansen U, Weng Z. Detection of cis-element clusters in higher eukaryotic DNA. //Bioinformatics. 2001. Vol.17, P.878–889. 125. Frith M.C., Li M.C., Weng Z. Cluster-Buster: Finding dense clusters of motifs in DNA sequences. //Nucleic Acids Res. 2003. Vol.31, P.3666–3668. doi: 10.1093/nar/gkg540.
126. Furletova E., Kucherov G., Noe L., Roytberg M., Tsitovich I. Transitive subset seeds for protein alignment. // Proseedings of The Sixth International Conference on Bioinformatics of Genome Regulation and Structure (BGRS'2008). 2008. P.77.
127. Gardner P.P, Wilm A., Washietl S. A benchmark of multiple sequence alignment programs upon structural RNAs. //Nucleic Acids Res. 2005. Vol. 33, P.2433-2439.

128. Garnier J., Osguthorpe D.-J., Robson B. Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins. // J. Mol. Biol. 1978. Vol. 120, P. 97–120.
129. Gelfand M.S., Koonin E.V. Avoidance of Palindromic Words in Bacterial and Archaeal Genomes: a Close Connection with Restriction Enzymess. //Nucleic Acids Research. 1997. Vol.25, N.12. P.2430–2439.
130. Gelfand M., Koonin E., Mironov A. Prediction of Transcription Regulatory Sites in Archaea by a Comparative Genome Approach. //Nucleic Acids Research. 2000. Vol. 28, P. 695–705.
131. Gelfand M.S., Podolsky L.I., Astakhova T.V., Roytberg M.A. Prediction of the exon-intron structure and multicriterial optimization. //Bioinformatics and Genome Research (H.A.Lim, C.R.Cantor, eds.). World Scientific Publ. Co., Singapore. 1995. P. 173-183.
132. Gelfand M.S., Podolsky L.I., Astakhova T.V., Roytberg M.A. Recognition of genes in human DNA sequences. // Journal of Computational Biology. 1996. Vol.3, N.2. P.223-234.
133. Gelfand M.S., Roytberg M.A.. Prediction of the exon-intron structure by a dynamic programming approach. // Biotechnology Software. 1992. P. 13-18.
134. Gerstein M, Levitt M. A structural census of the current population of protein sequences. Proc Natl Acad Sci U S A. 1997 Oct 28;94(22):11911-11916.
135. Giegerich R., Hochsmann M., Kurtz S., Toller T. Local similarity in RNA secondary structures. //In Proceedings of Computational Systems Bioinformatics (CSB). 2003. P. 159-168.
136. Goad W.B, Kanehisa M.I. Pattern recognition in nucleic acid sequences. I. A general method for finding local homologies and symmetries.// Nucleic Acids Res. 1982 . Vol.11. N.10. P. 247–263.
137. Gonnet G.H, Cohen M.A, Benner S.A. Exhaustive matching of the entire protein sequence database. //Science. 1992. Vol. 256, N.5062. P.1443-1445.
138. Gotoh O. An improved algorithm for matching biological sequences. // J. Mol. Biol. 1982. Vol.162, P.705-708.
139. Gukov, I.M., Astahova, T.V., Roytberg, M.A., Tsitovich I.I. One Shift" Alignments of Biological Sequences and their Statistics. // Proceedings of Moscow Conference on Computational Molecular Biology (MCCMB'05, Moscow, July 18-21, 2005. P. 136.
140. Gusfeld D. Algorithms on Strings, Trees, and Sequences. //Computer Science and Computational Biology. Press Syndicate of the University of Cambridge. 1997.
141. Gusfield D., Balasubramian K., Naor K. //Proc. 3rd Ann. ACM-SIAM Discrete Algorithms. 1992. P. 432–439.
142. Halfon MS, Michelson AM. Exploring genetic regulatory networks in metazoan development: methods and models. //Physiol Genomics. 2002. Vol.10 P.131–43.
143. Hannenhalli S., Pevzner P.A. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. //Journal of the ACM. 1999. Vol. 46, P. 1-27.
144. Harbison C.T., Gordon B., Lee T.I., Rinaldi N.J., Macisaac K.D., Danford T., Hannett N.M., Tagne J.B., Reynolds D.B., Yoo J., Jennings E.G., Zeitlinger J., Pokholok D.K., Kellis M., Rolfe P.A., Takusagawa K.T., Lander E.S., Gifford D.K., Fraenkel E.,

Young R.A. Transcriptional regulatory code of a eukaryotic genome. //Nature. 2004. Vol.431, P.99–104.
145. Henikoff S., Henikoff J.G. Amino acid substitution matrices from protein blocks // Proc. Natl. Acad. Sci. USA. 1992. Vol. 89, P. 10915–10919.
146. Henikoff S., Henikoff J.G. Amino acid substitution matrices. // Adv. Protein Chem. 2000. Vol.54, P.73-97.
147. Hermelin D., Landau G.M., Roytberg M. Pair-Wise Sequence-Structure RNA Alignment with Affine Gap Penalty. Algorithms in Bioinformatics. //Proceedings of the International Workshop (Moscow, July, 2006). Moscow, MCCME, Laboratoire J.V.Poncele . 2006. P.34.
148. Hirschberg D.S. A linear space algorithm for computing maximal common subsequence // CACM. 1975. Vol. 18, N. 6. P.341-343.
149. Hirschberg D.S. Algorithms for the Longest Common Subsequence Problem. // Journal of the ACM . 1977. Vol. 24 , N.4. P. 664 – 675.
150. Hofacker, I..L., et al. Fast folding and comparison of RNA secondary structures. //Monatsh. Chemie. 1994. Vol. 125 , P.167–188.
151. Hofacker I..L., et al. Conserved RNA secondary structures in viral genomes: a survey. //Bioinformatics. 2004. Vol. 20, P.1495–1499.
152. Hopcroft J.E., Ullman J.D. Formal languages and their relation to automata. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA 1969. P. 262.
153. Hu YJ, Sandmeyer S, McLaughlin C, Kibler D. Combinatorial motif analysis and hypothesis generation on a genomic scale. //Bioinformatics. 2000. Vol.16, P.222–232.
154. Hunt J.W., Szymanski T.G. A fast algorithm for computing longest common subsequences, //Comm. Assoc. Comput. 1977. Vol. 20, P.350-353.
155. Hwa T., Lassig M. Similarity Detection and Localization. //Phys. Rev. Lett. 1996. Vol. 76, P. 2591- 2594:
156. Ilie L., Ilie S. Long spaced seeds for finding similarities between biological sequences. // Proceedings of the 2nd International Conference on Bioinformatics & Computational Biology (BIOCOMP). 2007. P. 3–8.
157. Iliopoulos C.S., Kubica M., Rahman M.S., Walen T. Algorithms for Computing the Longest Parameterized Common Subsequence // Combinatorial Pattern Matching. LNCS. 2007. Vol. 4580. P. 265-273.
158. Izing, E. Beitrag zut Theorie des Ferromagnetizmus. //Zeitschr. Phys.1925. Vol. 31, P.253-258.
159. Jaeger, J.A., et al. Improved predictions of secondary structures for RNA. //Proc. Natl Acad. Sci. USA. 1989. Vol. 20, P.7706–7710.
160. Jareborg N., Birney E., Durbin R. Comparative analysis of noncoding regions of 77 orthologous mouse and human gene pairs. // Genome Res. 1999. Vol.9, P. 815-824.
161. Jegga A.G, Sherwood S.P, Carman J.W, Pinski A.T, Phillips J.L, Pestian J.P, Aronow B.J. Detection and visualization of compositionally similar cis-regulatory element clusters in orthologous and coordinately controlled genes. //Genome Res. 2002. Vol.12, P1408–1417.
162. Jones D. Protein secondary structure prediction based on position-specific scoring matrices. // J. Mol. Biol. 1999. Vol. 292, P.195–202.

163. Jones D.T., Taylor W.R., Thornton J.M. A new approach to protein fold recognition. //Nature. 1992. Vol. 358, P.86–89.
164. Jun S., Desplan C. Cooperative interactions between paired domain and homeodomain. //Development. 1996. Vol.122, P.2639–50.
165. Kabsch W., Sander C. Dictionary of protein secondary structure: pattern recognition of hydrogenbonded and geometrical features. //Biopolymers. 1983. Vol.22, P.2577–2637.
166. Kanehisa M. Use of statistical criteria for screening potential homologies in nucleic acid sequences. //Nucleic Acids Res. 1984. Vol.12. P. 203-213.
167. Karlin S., Altschul S.F. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. // Proc. Natl. Acad. Sci. USA. 1990. Vol. 87, P.2264-2268.
168. Kasami T. An efficient recognition and syntax-analysis algorithm for context-free languages. //Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.1965.
169. Kato M, Hata N, Banerjee N, Futcher B, Zhang MQ. Identifying combinatorial regulation of transcription factors and binding motifs.// Genome Biol. 2004. 5:R56. doi: 10.1186/gb-2004-5-8-r56. Epub 2004 Jul 28.
170. Katoh K, Kuma K, Toh H, Miyata T: MAFFT version 5: Improvement in Accuracy of Multiple Sequence Alignment. // Nucleic Acids Research 2005, Vol. 3. P. 511-518.
171. Keich U., Li M., Ma B., Tromp J. On spaced seeds for similarity search. // Discrete Applied Mathematics. 2004. Vol. 138, No. 3. P. 253–263.
172. Kent, W.J.: BLAT–the BLAST-like alignment tool.// Genome Research. 2002. Vol. 12, P. 656–664.
173. Kent W.J. Zahler A.M. Conservation, regulation, synteny, and introns in large-scale C.briggsae - C.elegans genomic alignment. //Genome Res. 2000. Vol. 10, P. 1115-1125.
174. Kister A.E, Roytberg M.A, Chothia C., Gelfand I.M.. The sequence determinants of cadherin molecules. // Protein Science. 2001. Vol.10, P. 1801-1810.
175. Kleene, S.C., Representation of events in nerve nets and finite automata. // Shannon, C.E., McCarthy, J. (Eds.), Automata Studies, Princeton University Press. 1956. P. 3-41.
176. Klein P.N.. Computing the edit-distance between unrooted ordered trees. // Proceedings of the 6th European Symposium on Algorithms (ESA). 1998. P. 91-102.
177. Klingenhoff A, Frech K, Werner T. Regulatory modules shared within gene classes as well as across gene classes can be detected by the same in silico approach. // Silico Biol. 2002. Vol.2, P.17–26.
178. Knut D.E. Art of Computer Programming: Fundamental Algorithms. //Addison-Wesley, Reading, MA, USA. 1968. Vol. 1, P. 658.
179. Knut D.E. Art of Computer Programming: Seminumerical Algorithms. // Addison-Wesley, Reading, MA, USA. 1969. Vol. 2. P. 638.
180. Knut D.E. Art of Computer Programming: Sorting and Searching. // Addison-Wesley, Reading, MA, USA. 1973. Vol. 3, P. 736.
181. Korn L.J., Queen C.L., Wegman M.N.. Computer analysis of nucleic acid regulatory sequences. // Proc Natl Acad Sci U S A. 1977. Vol.74, N.10. P.4401–4405.

182. Kramers, H.A., Wannier, G.H. Statistics of the one-dimensional ferromagnet.// Phys. Rev. 1941. Vol. 60, P.252-276.
183. Kruskal J.B. An Overview of Sequence Comparison: Time Warps. String Edit and Macromoleculas. // SIAM Rev. 1983. Vol.25, N. 26. P.201-238.
184. Kschischo M., Lassig M. Finite-temperature Sequence Alignment. // Pacific Symposium on Biocomputing. 2000. Vol. 5, P.621-632.
185. Kucherov G., Noe L., Ponty Y. Estimating seed sensitivity on homogeneous alignments. // Proceedings of the IEEE 4th Symposium on Bioinformatics and Bioengineering, IEEE Computer Society Press. 2004. P. 387–394.
186. Kucherov G., Noe L. Roytberg, M. Multi-seed lossless filtration. // IEEE/ACM Trans-actions on Computational Biology and Bioinformatics. 2005. Vol. 2, No. 1, 51-61.
187. Kucherov G. , Noe L., Roytberg M. A unifying framework for seed sensitivity and its application to subset seeds. //Journal of Bioinformatics and Computational Biology. 2006. Vol. 4, N. 2. P. 553–570, preliminary version in WABI 2005.
188. Kucherov G., Noe L., Roytberg M. Subset seed automaton. // Implementation and Application of Automata. Lecture Notes in Computer Science. Springer. Berlin-Heidelberg. 2007. Vol. 4783, P. 180-191
189. Kung, H.T., Luccio, F., Preparata F.P.On Finding the Maxima of a Set of Vectors // J. ACM. 1975. Vol. 22, P. 469–476.
190. Latchman D. S. Eukaryotic Transcription Factors. //Elsevier Academic Press, New York, 4-th edition. 2004. P. 360.
191. Lee T.I., Young R.A. Transcription of eukaryotic protein-coding genes. //Annu. Rev. Genet. 2000. Vol.34, P. 77–137.
192. Lesk A.M. Introduction to protein architecture. //Oxford, N.Y.: Oxford Univ. press. 2001. P.360.
193. Lesk, A. M., M. Levitt, C. Chothia. Alignment of the Amino Acid Sequences of Distantly Related Proteins Using Variable Gap Penalties. //Protein Engineering. 1986. Vol.1, P.77-78
194. Li H., Rhodius V., Gross C., Siggia E.D. Identification of the binding sites of regulatory proteins in bacterial genomes. //Proc Natl Acad Sci USA. 2002. Vol.99, P.11772–7. doi: 10.1073/pnas.112341999. Epub 2002 Aug 14.
195. Li M., Ma B., Kisman D., Tromp J. PatternHunter II: Highly Sensitive and Fast Homology Search. J. Bioinformatics and Computational Biology. 2004. Vol. 2, N. 3. P. 417-440.
196. Li W.H. Molecular evolution. //Sunderland: Sinauer Associates. 1997. P.443.
197. Liaw G.J, Lengyel J.A. Control of tailless expression by bicoid, dorsal and synergistically interacting terminal system regulatory elements. //Mech Dev. 1993. Vol.40, P.47–61. doi: 10.1016/0925-4773(93)90087-E.
198. Lifanov A., Makeev V., Nazina A., Papatsenko D. Uniform clusters in Drosophila. //Genome Res. 2003. Vol.13, P.579–588. doi: 10.1101/gr.668403
199. Lim V.I. Algorithms for prediction of alpha-helical and beta-structural regions in globular proteins.// J. Mol. Biol. 1974. Vol. 88, P. 857–872.
200. Lipman D.J, Pearson W.R. Rapid and sensitive protein similarity searches. //Science. 1985. Vol.227, P.1435–1441.

201. Litvinov I.I., Finkelstein A.V., Roytberg M.A. Optimization of accurancy and confidence for alignment algorithms exploiting data on secondary structure. //Proceedings of the fifth international conference on bioinformatics of genome regulation. BGRS'2006. 2006. P.289-293.
202. Lothaire. Applied Combinatorics on Words. //Cambridge University Press, Reading, Mass. 2005.
203. Luthy R., McLachlan A.D., Eisenberg D. Secondary structure-based profiles: use of structure-conserving scoring tables in searching protein sequence databases for structural similarities. //Proteins. 1991. Vol.10, P.229–239.
204. Lyngso R.B., et al. Fast evaluation of internal loops in RNA secondary structure prediction. // Bioinformatics. 1999 Vol.15, P. 440–445.
205. Ma B., Li W, Zhang K. . Amino Acid Classification and Hash Seeds for Homology Search. // In: Lecture Notes in Computer Science. 2009. Vol. 5462. P. 44-51.
206. Ma B., Tromp J., Li M., PatternHunter: Faster and More Sensitive Homology Search. // Bioinformatics. 2002. Vol. 18, N. 3. P. 440-445.
207. Ma B., Zhang K.. The similarity metric and the distance metric.// In Proceedings of the 6th Atlantic Symposium on Computational Biology and Genome Informatics. 2005. P. 1239-1242.
208. MacIsaac K.D., Fraenkel E. Practical strategies for discovering regulatory DNA sequence motifs. //PloS Comput Biol. 2006;2:e36. doi: 10.1371/journal.pcbi.0020036.
209. Mak D., Gelfand Y., Benson G. Indel seeds for homology search. // Bioinformatics. 2006. Vol. 22, N. 14. P. 341–349.
210. Makeev V., Lifanov A., Nazina A., Papatsenko D. Distance preferences in distribution of binding motifs and hierarchical levels in organization of transcription regulatory information. //Nucleic Acids Res. 2003.Vol.31, P.6016–26. doi:10.1093..
211. Markstein M., Markstein P., Markstein V., Levine M. Genome-wide Analysis of Clustered Dorsal Binding Sites Identifies Putative Target Genes in the Drosophila Embryo. //PNAS. 2002. Vol.99, P.763–768. doi: 10.1073/pnas.012591199.
212. Markstein M., Zinzen R., Markstein P., Yee K.P., Erives A., Stathopoulos A., Levine M. A regulatory code for neurogenic gene expression in the Drosophila embryo. //Development. 2004. Vol.131, P.2387–94. doi: 10.1242/dev.01124.
213. Marsden, R. L., McGuffin, L. J., Jones, D. T. Rapid protein domain assignment from amino acid sequence using predicted secondary structure. // Protein Sci.2002.Vol. 11, P.2814-2824.
214. Masek W.J. Paterson M.S. A Faster Algorithm for Computing String Edit Distances. // J. of Computer and Systems Sciences. 1980. Vol.20, N.1. 18-31.
215. Mathews D.H., et al. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. //J. Mol. Biol . 1999. Vol. 288, P. 911–940.
216. Mayr G., Domingues F.S., Lackner P. Comparative Analysis of Protein Structure Alignments. // BMC Structural Biology. 2007. Vol.7, P.50.
217. McCaskill, J.S. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. // Biopolymers. 1990. Vol. 29, P.1105–1119.
218. McNaughton R., Yamada H. Regular expressions and state graphs for automata. // IRE Transactions on Electronic Computer 1960. Vol.9, N.1. P. 39–47.

219. Miller W. Comparison of genomic DNA sequences: solved and unsolved problems. //Bioinformatics. 2001. Vol.17, P. 391-397.
220. Miller W., Myers E.W. Sequence comparison with concave weighting functions. //Bulletin of Mathematical Biology. 1988. Vol.50, P. 97-120.
221. Mironov A.A., Koonin E.V., Roytberg M.A., Gelfand M.S. Computer analysis of transcription regulatory patterns in completely sequenced bacterial genomes. //Nucleic Acids Res. 1999. Vol.15, N.27. P. 2981-9.
222. Mironov A.A., Roytberg M.A. Pevzner P.A., Gelfand M.S. Performance guarantee gene predictions via spliced alignment. //Genomics. 1998, Vol. 51, P. 332-339.
223. Mitashev V.I, Koussoulakos S., Zinov'eva R.D., Ozerniuk N.D., Mikaelian A.S., Shmukler E., Smirnova L. A. Constructive synergism of regulatory genes expressed in the course of the eye and muscle development and regeneration.// Izv Akad Nauk Ser Biol. 2001. P.261–75.
224. Moore P.B. Structural motifs in RNA. //Annual review of biochemistry. 1999. Vol. 68, P.287-300.
225. Mott R. Maximum-likelihood estimation of the statistical distribution of Smith-Waterman local sequence similarity scores. //Bull. Math. Biol. 1992. Vol. 54. P.59-75.
226. Mott R. Accurate formula for p-values of gapped local sequence and profile alignments. //J. Mol Biol. 2000. Vol.300, P. 649-659.
227. Mount D.W. Bioinformatics. Sequence and genome analysis. //Cold Spring Harbor Laboratory Press,U.S. 2001. P. 564.
228. Myers E.W. An O(N D) difference algorithm and its variations. // Algorithmica. 1986. Vol. 1, P. 251-266.
229. Myers E.W, Miller W. Optimal alignments in linear space.// Comput Appl Biosci. 1988a. Vol.4, N.1. P.11-7.
230. Myers E.W., Miller W. Chaining multiple-alignment fragments in sub-quadratic time. // Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms. San Francisco, SA. 1995. P. 38 - 47.
231. Naor D., Brutlag D. On near optimal a;ignmentsin biological sequences. // J. Comp.Biol. 1994. Vol.1, P. 349-366.
232. Navarro G,. Raffinot M. Flexible Pattern Matching in Strings //Practical On-Line Search Algorithms for Texts and Biological Sequences. Cambridge Univ. Press. 2002.
233. Nazipova N.N., Shabalina S.A., Ogurtsov A.Yu., Kondrashov A.S., Roytberg M.A., Buryakov G.V., Vernoslov S.E. SAMSON: a software package for the biopolymer primary structure analyses. //Comput. Appl. Biosci. 1995. Vol. 11, N. 4. P.423-426.
234. Needleman S.B., Wunsch C.D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. // J.Mol. Biol. 1970. Vol.48, P.443-453.
235. Nicodeme P. Regexpcount, a symbolic package for counting problems on regular expressions and words. //Fundamenta Informaticae. 2003. Vol. 56, P. 71–88.
236. Nicodeme P., Salvy B., Flajolet P. Motif Statistics. //Theoretical Computer Science. 2002. Vol. 287, P. 593–618.
237. Noe L., Kucherov G. Improved Hit Criteria for DNA Local Alignment. // BMC Bioinformatics. 2004. Vol. 5, N. 149. Oct. 2004.

238. Noe L. and Kucherov G. YASS: enhancing the sensitivity of DNA similarity search. // Nucleic Acid Research, 2005. Vol. 33, P. 540–543.
239. Notredame C., Higgins D.G., Heringa J. T-Coffee: a novel method for fast and accurate multiple sequence alignment. //J Mol Biol. 2000. Vol. 302, P.205–217.
240. Nozaki Y., Bellgard M. Statistical evaluation and comparison of a pairwise alignment algorithm that a priori assigns the number of gaps rather than employing gap penalties // BIOINFORMATICS . 2005. Vol.21, N.8. P. 1421-1428.
241. Nussinov R., Jacobson, A.B. Fast algorithm for predicting the secondary structure of single-stranded RNA. // Proc. Natl Acad. Sci. USA. 1980. Vol. 77, P. 6309–6313.
242. Ogurtsov A.Y., Roytberg M.A., Shabalina S.A., Kondrashov A.S. OWEN: aligning long collinear regions of genomes. //Bioinformatics. 2002 . Vol. 18, P. 1703-1704.
243. Ogurtsov, A.Yu, Shabalina, S.A., Kondrashov, A.S., Roytberg M.A. Analysis of internal loops within the RNA secondary structure in almost quadratic time. // Bioinformatics. 2006, Vol. 22, No. 11, P. 1317-1324
244. Ogurtsov A.Yu., Vasilchenko A.N., Vlasov P.K., Shabalina S.A., Kondrashov A.S., Roytberg M.A. OWEN-Script – extended tool for pairwise genome alignment. //Proceedings of the fifth international conference on bioinformatics of genome regulation. 2006. P.122-125.
245. Papatsenko D, Makeev V, Lifanov A, Regnier M, Nazina A, Desplan C. Extraction of Functional Binding Sites from Unique Regulatory Regions: The Drosophila Early Developmental Enhancers.// Genome Research. 2002. Vol. 12, P.470–481.
246. Papatsenko D. ClusterDraw web server: a tool to identify and visualize clusters of binding motifs for transcription factors. //Bioinformatics. 2007. Vol.23, P.1032–1034. doi: 10.1093/bioinformatics/btm047.
247. Pareto V. Manual of political economy. New York: A.M. Kelley. 1972. 516 p.
248. Pascarella S., Milpetz F., Argos P. A databank (3D-ali) collecting related protein sequences and structures. // Protein Eng. 1996. Vol. 9, P.249–251.
249. Pearson W.R. Empirical statistical estimates for sequence similarity searches.// J.Mol.Biol . 1998. Vol.276, P. 71-84.
250. Pearson W.R. Rapid and Sensitive Sequence Comparison with FASTP and FASTA. // Methods Enzymol. 1990. Vol. 183. P. 63-98.
251. Pearson W.R., Lipman D.J. Improved tools for biological sequence comparison. //Proc Natl Acad Sci U S A. 1988. Vol. 85, N.8. P. 2444-8.
252. Pearson W.R. Comparison of methods for searching protein sequence databases. //Protein Sci. 1995. Vol.4, N.6. P.1145-60.
253. Pearson W.R. Flexible sequence similarity searching with the FASTA3 program package. //Methods Mol Biol. 2000. Vol. 132, P.185-219.
254. Pearson W.R. Uva FASTA Downloads: http://fasta.bioch.virginia.edu/fasta_www2/fasta_down.shtml
255. Peterlongo P., Noe L., Lavenier D., Georges G., Jacques J., Kucherov G., Giraud M. Protein similarity search ith subset seeds on a dedicated reconfigurable hardware. // Proceedings of the 2nd Workshop on Parallel Computational Biology, Gdansk (Poland), ser. Lecture Notes in Computer Science. 2007. Vol. 4967.
256. Pevzner P., Waterman M. “Multiple Filtration and Approximate Pattern Matching. Algorithmica. 1995. Vol. 13, P. 135-154.

257. Pirovano W., Feenstra K.A., Heringa J. The meaning of alignment: lessons from structural diversity. // BMC Bioinformatics 2008,V. 9 P. 556 doi:10.1186/1471-2105-9-556
258. Polyanovsky V., Roytberg M.A., Tumanyan V.G. Reconstruction of genuine pair-wise sequence alignment. //Comput Biol. 2008. Vol.15, N. 4. P. 379-91.
259. Ptitsyn O.B., Finkelstein A.V. Theory of protein secondary structure and algorithm of its prediction. //Biopolymers. 1983 . Vol. 22, P.15–25.
260. Ramensky VE, Makeev VJu, Roytberg MA, Tumanyan VG. DNA segmentation through the Bayesian approach. //J Comput Biol. 2000. Vol.7, N.1-2. P. 215-31.
261. Ramensky V.E., Makeev V.Y., Roytberg M.A., Tumanyan V.G. Segmentation of long genomic sequences into domains with homogeneous composition with BASIO software. // Bioinformatics. 2001. Vol.17, N.11. P. 1065-1066.
262. Rebeiz M., Reeves N.L., Posakony J.W. SCORE: a computational approach to the identification of cis-regulatory modules and target genes in whole-genome sequence data. Site clustering over random expectation. //Proc Natl Acad Sci USA.2002. Vol.99, P.9888–93. doi: 10.1073/pnas.152320899. Epub 2002 Jul 09.
263. Regnier, M., Kirakosyan, Z., Furletova E., Roytberg, M. A Word Counting Graph.// In. London Algorithmics 2009. Theory and Practice (Chan, J., Daykin, J.W. and Rahman M.S., eds). 2009. P. 10-43.
264. Resch A. M. , L. Carmel L. Marino-Ramirez A. Y. Ogurtsov S. A. Shabalina I. B. Rogozin E. V. Koonin. Widespread Positive Selection in Synonymous Sites of Mammalian. //Genes Mol. Biol. Evol. 2007. Vol. 24, N.8. P. 1821 - 1831.
265. Rice D., Eisenberg D. A 3D-1D substitution matrix for protein fold recognition that includes predicted secondary structure of the sequence. //J Mol. Biol. 1997. Vol. 267, P.1026–1038.
266. Rice,P. Longden,I, Bleasby,A. *EMBOSS: The European Molecular Biology Open Software Suite* Trends in Genetics 2000. Vol.16, N 6. P. 276--277
267. Rochkind M.J. The Source Code Control System. // IEEE Transactions on Software Engineering. 1975. Vol.1, N. 4. P. 364-370.
268. Rossberg M., Theres K., Acarkan A., Herrero R., Schmitt T., Schumacher K., Schmitz G., Schmidt R. Comparative sequence analysis reveals extensive microcolinearity in the Lateral suppressor regions of the tomato, Arabidopsis, and Capsella genomes. //Plant Cell. 2001. Vol. **13**, P.979- 988.
269. Rost B., Schneider R., Sander C. Protein fold recognition by prediction-based threading. //J. Mol. Biol. 1997.Vol. 270, P.471–480.
270. Roytberg M.A. (a) Fast algorithm for optimal aligning of symbol sequences. // Mathematical methods of the analysis of biopolymer sequences. AMS, Providence. 1992. P. 103-117.
271. Roytberg M.A. (b) A Search for Common Patterns in many Sequences. // CABIOS. 1992. Vol.8, N.1. P. 57 – 64.
272. Roytberg M.A. Similarity Search in Biological Sequences. // In:"Modelling and Computer Methods in Molecular Biology and Genetics" (eds. V.A.Ratner and N.A. Kolchanov). Nova Science Publishers, Inc., NY .1992, P. 81-86.
273. Roytberg M.A. Pareto-optimal alignments of symbol sequences. // Preprint, ONTI NTsBI, Pushchino. 1994.

274. Roytberg M.A., Astahova T.V., Gelfand M.S. Combinatorial approaches to gene recognition. // Computers and Chemistry. 1998. Vol.1. N.21. P. 229-236.
275. Roytberg M., Gambin A., Noe L., Lasota S., Furletova E., Szczurek E., Kucherov G. On subset seeds for protein alignment. // Transactions on Computational Biology and Bioinformatics. 2009. Vol. 6, N. 3. P. 483-494.
276. Roytberg, M.A., Ogurtsov A.Y., Shabalina S.A., Kondrashov A.S. A hierarchical approach to aligning collinear regions of genomes. Bioinformatics. 2002. Vol. 18, P.1673–1680.
277. Russell D., Out H., Sayood K. Grammar-based distance in progressive multiple sequence alignment // BMC Bioinformatics. 2008. V. 9. P. 306 doi:10.1186/1471-2105-9-30
278. Russell R.B., Barton G.J. Structural features can be unconserved in proteins with similar folds. An analysis of side-chain to side-chain contacts secondary structure and accessibility.// J. Mol. Biol. 1994. Vol.244, P.332–350.
279. Russell R.B., Copley R.R., Barton G.J. Protein fold recognition by mapping predicted secondary structures. //J. Mol. Biol. 1996. Vol. 259, P.349–365.
280. Sanchez R, Sali A. Comparative protein structure modeling. Introduction and practical examples with modeller. //Methods Mol Biol. 2000. Vol.143, P.97–129.
281. Sander C., Schneider R. Database of homology-derived protein structures and the structural meaning of sequence alignment. //Proteins. 1991. Vol. 9, P.56-68.
282. Sankoff D. Simultaneous Solution of the RNA Folding, Alignment, and Protosequence Problems. // SIAM J Appl Math. 1985. Vol, 45, P.810-825.
283. Sankoff D., Kruskal J.B. (eds) Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. // Reading, MA: Addison-Wesley. 1983. P.426
284. Schaffer A.A, Aravind L., Madden T.L., Shavirin S., Spouge J.L., Wolf Y.I., Koonin E.V. Altschul SF. Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements. //Nucleic Acids Res. 2001. Jul1.
285. Schwartz S., Kent J., Smit A., Zhang Z., Baertsch R., R. Hardison R., Haussler D., Miller W. Human—Mouse Alignments with BLASTZ. // Genome Research. 2003. Vol. 13, P. 103-107.
286. Schwartz S., Zhang Z., Frazer K.A., Smit A., Riemer C., Bouck J., Gibbs R., Hardison R., Miller W. PipMaker - A Web server for aligning two genomic DNA sequences. //Genome Res. 2000. Vol.10, P.577-586.
287. Sellers P.H. On the Theory and Computation of Evolutionary Distance, SIAM // J. Appl. Math. 1974. Vol. 26, P. 787-793.
288. Sellers, P. H. (1979), Pattern recognition in genetic sequences. //Proc. Natl. Acad. Sci. USA. 1979. Vol. 76, P. 3041.
289. Sellers P.H. The theory and computation of evolutionary distances: pattern recognition. // J.of Algorithms. 1980. Vol.1, P.359-373.
290. Shabalina S.A., Kondrashov A.S. Pattern of selective constraint in C.elegans and C.briggsae genomes. //Genet. Res. 1999. Vol. 74, P. 23-30.

291. Shabalina S.A., Ogurtsov A.Y., Kondrashov V.A., Kondrashov A.S. (2001) Selective constraint in intergenic regions of human and mouse genomes. //Trends in Genet. 2001. Vol.17, P.373-376.
292. Sheridan R.P., Dixon J.S., Venkataraghavan R., Kuntz I.D., Scott K.P. Amino acid composition and hydrophobicity patterns of protein domains correlate with their structures. //Biopolymers. 1985. Vol. 24, P.1995–2023.
293. Shi, J., Blundell, T. L., Mizuguchi, K. FUGUE: sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties.// J. Mol. Biol. 2001. Vol.310, P.243-257.
294. Smith T. F. , Waterman M. S.,Fitch W. M.. Comparative biosequence metrics. // J. Mol. Evol. 1981. Vol. 18, P. 38–46
295. Smith T.F., Waterman M.S. Identification of common molecular subsequences. //J. Mol. Biol. 1981. Vol.147, P.195-197.
296. Sosinsky A., Bonin C., Mann R., Honig B. Target Explorer: an automated tool for the identification of new target genes for a specified set of transcription factors. //Nucleic Acids Research. 2003. Vol.31, P.3589–3592. doi:10.1093/nar/gkg5.
297. Stephen G.A. String Searching Algorithms. // WORLD SCIENTIFIC PUBLISHING COMPANY. Singapore. 1994. P.256.
298. Sun Y., Buhler J. Designing Multiple Simultaneous Seeds for DNA Similarity Search.// Proc. Eighth Ann. Int'l Conf. Research in Computational Molecular Biology (RECOMB 2004). 2004. P 76-84.
299. Sunyaev S.R., Bogopolsky G.A., Oleynikova N.V., Vlasov P.K., Finkelstein A.V., Roytberg M.A. From analysis of protein structural alignments toward a novel approach to align protein sequences. // Proteins. 2004. Vol. 54, P.569–582.
300. Sze S.H, Roytberg M.A. , Gelfand M.S., Mironov A.A., Astakhova T.V., Pevzner P.A. Algorithms and software for support of gene identification experiments. //Bioinformatics. 1998. Vol.1, N. 14. P. 14-19.
301. Szymanski M, Barciszewska MZ, Erdmann VA, Barciszewski J: 5S Ribosomal RNA Database. //Nucleic Acids Res. 2002. Vol. 30, P.176-178.
302. Thompson J.D., Higgins D.G., Gibson T.J. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence eighting, position-specific gap penalties and weight matrix choice. // Nucleic Acids Res. 1994. Vol. 22, P.473-480.
303. Thompson J.D, Koehl P., Ripp R., Poch O. BAliBASE 3.0: latest developments of the multiple sequence alignment benchmark. //Proteins. 2005. Vol.61, N.1. P.127-36
304. Tinoco,I.Jr., Uhlenbeck,O.C., and Levine,M.D. Estimation of secondary structure in ribonucleic acids. // Nature.1971. V. 230**.** P. 3362-3367
305. Tinoco I. Jr, et al. Improved estimation of secondary structure in ribonucleic acids. // Nat. New Biol . 1973. Vol. 246, P.40–41.
306. Touzet H. Tree edit distance with gaps.// Information Processing Letters. 2003. Vol.85, N.3. P.123-129.
307. Ukkonen E. Algorithms for approximate string matching. // Information and Control. 1985. Vol.64, P.100-118.
308. Ulam S. M. Some Ideas and Prospects in Biomathematics. // Annu Rev Biophys Bioeng. 1972. Vol.1, P.277-92.

309. Venkatesh B., GilliganP., Brenner S. Fugu: a compact vertebrate reference genome. //FEBS Lett. 2000. Vol. 476, P.3-7.
310. Vingron M, Argos P. Determination of reliable regions in protein sequence alignments. //Protein Engineering. 1990. Vol. 3, N.7. P.565-569.
311. Vingron M. Near-optimal sequence alignment // Current Opinion in Structural Biology Volume 6, Issue 3, June 1996, Pages 346-352
312. Vingron M., Waterman M.S. Sequence alignment and penalty choice: Review of concepts, case studies and implications // Journal of Molecular Biology. 1994. Vol.235, N. 1. P. 1-12
313. Vogt, G., Etzold, T., Argos P. An assessment of amino acid exchange matrices in aligning protein sequences: the twilight zone revisited. //J. Mol. Biol. 1995. Vol.249, P.816-831.
314. Wagner A. Genes regulated cooperatively by one or more transcription factors and their identification in whole eukaryotic genomes. //Bioinformatics. 1999. Vol.15, P.776–784. doi: 10.1093/bioinformatics/15.10.776.
315. Wagner R.A., Fischer M.J. The String-to-String Correction Problem. // Journal of ACM. 1974. Vol. 21, N. 1. P. 168-173.
316. Wallqvist A., Fukunishi Y., Murphy L.R., Fadel A., Levy R.M. Iterative sequence/secondary structure search for protein homologs: comparison with amino acid sequence alignments and application to fold recognition in genome databases.//Bioi.
317. Waterman M.S. Sequence alignments in the neighborhood of the optimum with general application to dynamic programming. // PNAS. 1983. Vol. 80, P.10 3123-3124.
318. Waterman M.S. Efficient sequence alignment algorithm. // J.Theor.Biol. 1984. Vol.108, 333-337.
319. Waterman M.S.(ed.) Mathematical methods for DNA sequences. // CRC Press, Boca Raton, FL. 1989. 293 p.
320. Waterman M.S. Introduction to Computational Biology. // London: Chapman and Hall Press. 1995.
321. Waterman M.S., Eggert M., Lander E. Parametric sequence comparisons. // Proc.Nat. Acad. Sci. USA. 1992. Vol. 89, P. 6090–6093
322. Waterman, M. S., Smith, T.F., Beyer, W.A. Some Biological Sequence Metrics // Advances in mathematics. 1976. Vol.20, P.367 – 387.
323. Waterman M. S., Smith T. F. RNA secondary structure: a complete mathematical analysis. // Mathematical Biosciences. 1978. Vol. 42, P. 257–266.
324. Wilbur W.Y., Lipman D.J. Rapid similarity searches of nucleic acid and protein data banks. // PNAS USA. 1983. Vol.80, P.726-730.
325. Wilm A, Mainz I, Steger G. An enhanced RNA alignment benchmark for sequence alignment programs. // Algorithms Mol Biol. 2006. Vol 24, P.1:19.
326. Wolf Y.I., Rogozin I.B., Kondrashov A.S., Koonin E.V. Genome alignment, evolution of prokaryotic genome organization, and prediction of gene function using genomic context. //Genome Research. 2001. Vol.11, P. 356-372.

327. Wuchty S., Fontana W., Hofacker I.L., Schuster P. Complete suboptimal folding of RNA and the stability of secondary structures. // Biopolymers 1999. Vol. 49, P.145–165.
328. Xia T., SantaLucia J. Jr., Burkard M.E., Kierzek R., Schroeder S.J., Jiao X., Cox C., Turner D.H. Thermodynamic parameters for an expanded nearest-neighbor model for formation of RNA duplexes with Watson–Crick base pairs. //Biochemistry. 1998. Vol.37 4735.
329. Xu J. Brown D., Li M., Ma B.. Optimizing Multiple Spaced Seeds for Homology Search. //Proc. 15th Symp. Combinatorial Pattern Matching. 2004. P. 47-58.
330. Yang I.H., Wang S.H., Chen Y.H., Huang P.H., Ye L., Huang X., Chao K.M. Efficient methods for generating optimal single and multiple spaced seeds. // Proceedings of the 4th IEEE Symposium on Bioinformatics and Bioengineering. 2004. P. 411
331. Younger D. Recognition and parsing of context-free languages in time n3. // Information and Control. 1967. Vol. 10, N.2. P. 189–208.
332. Zafar N., Mazumder R., Seto D. Comparisons of gene colinearity in genomes using GeneOrder2.0. //Trends. Biochem. Sci. 2001. Vol. 26, P.514-516.
333. Zhang J, Jiang B, Li M, Tromp J, Zhang X, Zhang M. Computing exact P-values for DNA motifs. //Bioinformatics. 2007. Vol.23, P.531–537. doi: 10.1093/bioinformatics/btl662.
334. Zhang K., Shasha D. Simple fast algorithms for the editing distance between trees and related problems. //SIAM Journal on Computing. 1989. Vol. 18, N.6. P.1245-1262.
335. Zhang Z., Berman P., Miller W. Alignments without low-scoring regions. //J. Comput. Biol. 1998. Vol.5, P.197-210.
336. Zhang Z., Raghavachari B., Hardison R.C., Miller W. Chaining multiple-alignment blocks. //J. Comput. Biol. 1994. Vol. 1, P.217-226.
337. Zhang Z., Scheffer A., , Miller W., Madden T.,Lipman D., Koonin E., Altschul S. Protein sequence similarity searches using patterns as seeds // Nucleic Acids Research, 1998, Vol. 26, No. 17. P. 3986–3990
338. Zhu Z, Shendure J, Church GM. Discovering functional transcription-factor combinations in the human cell cycle.// Genome Res. 2005. Vol.15, P.848–55. doi: 0.1101/gr.3394405.
339. Zuker M. On finding all suboptimal foldings of an RNA molecule. //Science. 1989. 244, 48–52.
340. Zuker M. Suboptimal sequence alignment in molecular biology. Alignment with error analysis.// J Mol Biol. 1991 Vol.221, N.2. P.403-20.
341. Zuker M. Mfold web server for nucleic acid folding and hybridization prediction. //Nucleic Acids Res. 2003. Vol.31, P.3406–3415.
342. Zuker M., Mathews D.H., Turner D.H., et al. Algorithms and thermodynamics for RNA secondary structure prediction: a practical guide. // *RNA Biochemistry and Biotechnology*, J. Barciszewski & B.F.C. Clark, eds., NATO ASI Series, Kluwer Academic Publishers, 1999.
343. Zuker M., Sankoff D. RNA secondary structures and their prediction. //Bull. Math. Biol. 1984. Vol.46, P. 591–621.

344. Zuker M., Stiegler P. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. //Nucleic Acids. Res. 1981. Vol. 9, P.133–148.
345. Zvelebil, M., Baum, J.O. Understanding bioinformatics. London. Garland Science. 2007. P. 800