

Государственное образовательное учреждение высшего профессионального
образования

«Московский государственный технический
университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКИ И СИСТЕМ УПРАВЛЕНИЯ
КАФЕДРА ТЕОРЕТИЧЕСКОЙ ИНФОРМАТИКИ И КОМПЬЮТЕРНЫХ
ТЕХНОЛОГИЙ

Пояснительная записка
к дипломному проекту на тему:

МНОЖЕСТВЕННОЕ ВЫРАВНИВАНИЕ
КОДИРУЮЩИХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ ДНК С
УЧЕТОМ СДВИГОВ РАМКИ СЧИТЫВАНИЯ

Студент–дипломник _____ Батусов П. В.

Научный руководитель _____ Страшнов П. В.

Москва 2015

Аннотация

Данная работа посвящена изучению различных методов построения выравниваний кодирующих последовательностей ДНК. Отдельное внимание уделено подходу «двухуровневого» выравнивания для получения биологически обоснованного результата. На его основе был разработан и протестирован собственный алгоритм построения множественного выравнивания с учетом сдвигов рамки считывания. Итогом работы является кроссплатформенное приложение для выравнивания кодирующих последовательностей, которое по сравнению с существующими аналогами имеет схожее качество результата и более высокую производительность.

Содержание

ВВЕДЕНИЕ	4
1 Обзор предметной области	7
1.1 Существующие методы поиска гомологий в биологических последовательностях	7
1.1.1 Алгоритм Нидлмана-Вунша	8
1.1.2 Алгоритм Смита-Ватермана	9
1.1.3 Алгоритм Хиршберга	10
1.2 Алгоритмы множественного выравнивания	12
1.2.1 Выравнивание в «кубе»	12
1.2.2 Выравнивание выравниваний. Алгоритм Clustal	13
1.3 Выравнивание с учетом открытых рамок считывания	15
1.3.1 Трехэтапный подход	15
1.3.2 Двухуровневое выравнивание	16
1.3.3 MACSE	16
2 Алгоритм выравнивания	18
2.1 Принятые обозначения	18
2.2 Построение парного выравнивания	20
2.3 Алгоритм кластеризации	24
2.4 Построение множественного выравнивания	26
3 Программная реализация	28
3.1 Структуры данных	28
3.1.1 Представление биологических последовательностей	28
3.1.2 Класс построения парного выравнивания	30
3.1.3 Профили	32
3.2 Общая схема работы	33

3.2.1	Чтение входных данных	34
3.2.2	Определение порядка выравниваний	34
3.2.3	Объединение профилей	35
3.3	Руководство пользователя	37
3.3.1	Опции компиляции	37
3.3.2	Параметры запуска	38
3.3.3	Использование собранных библиотек	41
4	Тестирование	43
4.1	Оценка качества выравнивания	43
4.2	Оценка производительности	46
	ВЫВОДЫ	51
	ЗАКЛЮЧЕНИЕ	52
	СПИСОК ЛИТЕРАТУРЫ	53

ВВЕДЕНИЕ

Биоинформатика — это молодая область науки, возникшая в 1976-1978 годах и сформировавшаяся в 1980 году. В настоящее время она переживает бурный рост, связанный с развитием техники получения и обработки биологической информации. По сути, это собрание различных математических моделей и методов в помощь биологам для решения биологических задач, таких как предсказание пространственной структуры белков, расшифровка структуры ДНК, хранение, поиск и аннотация биологической информации [1].

Одной из ключевых задач в биоинформатике является оценка сходства последовательностей. Ее решение позволяет, например, описывать эволюционный путь генетических элементов, предсказывать связь между структурой генетического материала и его функциональностью.

Для того чтобы определить, насколько две биологические последовательности «похожи», их представляют в виде строк символов из алфавита соответствующего нуклеотидам и аминокислотам. К ним применяют алгоритмы выравнивания, основанные на размещении исходных последовательностей мономеров ДНК, РНК или белков друг под другом таким образом, чтобы максимизировать некоторую функцию оценки результата [2]. Качество выравнивания определяют, назначая штрафы за несовпадение букв и за наличие пробелов (когда приходится разрывать одну последовательность для того, чтобы получить наибольшее число совпадающих позиций), например, через расстояние Левенштейна — минимальное число элементарных операций (вставка, удаление или замена символа в строке), чтобы превратить одну строку в другую [3]. При построении выравнивания решается задача максимизации выбранной функции оценки. В такой постановке рассматривается задача получения «глобального выравнивания». Необходимо отметить, что для полных геномов глобальное выравнивание не работает, так как при мутации, помимо вставок, удалений и замен, бывают нелинейные перестройки, которые могут менять порядок и ориентацию целых геномных блоков. Для решения, аналогично задаче поиска глобального выравнивания, формулируют задачу поиска «локального выравнивания»: для двух произвольных строк A и B найти две самые похожие подстроки и их выравнивание.

Алгоритмы множественного выравнивания, аналогично алгоритмам пар-

ного выравнивания, представляют собой инструмент для установления функциональных, структурных или эволюционных взаимосвязей между биологическими последовательностями. Несмотря на то что задача множественного выравнивания была сформулирована более 20 лет назад [4], она до сих пор не теряет своей актуальности. Если говорить о множественном глобальном выравнивании, то, по сравнению с парным выравниванием, практически ничего не меняется: необходимо расставить разрывы в выравниваемых строках таким образом, чтобы «счет по столбцам» был максимален. Счет по столбцу можно вести, перебирая все пары символов. Множественное локальное выравнивание обобщить на многомерный случай не так просто. Во-первых, какие-то подстроки могут быть не во всех последовательностях. Во-вторых, последовательности могут содержать дублицированные участки. Поэтому для решения такой задачи необходимо более точно сформулировать условия выравнивания.

Таким образом, две главные составляющие автоматических методов выравнивания — это непосредственно алгоритм и функция оценки качества полученного результата. На сегодняшний день можно выделить два основных алгоритма выравнивания биологических последовательностей: алгоритм Нидлмана-Вунша и алгоритм Смита-Ватермана. Существуют различные их модификации, использующие эвристики для уменьшения количества шагов алгоритма или требуемого объема памяти, однако, эти методы строят выравнивание без проверки биологического смысла результата. В погоне за лучшим счетом происходит потеря качества: множественные разрывы на нуклеотидном и появление стоп-кодона на аминокислотном уровнях.

Построение качественного, биологически обоснованного выравнивания нуклеотидных последовательностей с сохранением открытых рамок считывания является очень важной и пока что нерешенной задачей биоинформатики. Для получения парного выравнивания на данный момент существуют программные утилиты, выдающие хороший результат, однако, имеющие столь высокую вычислительную сложность, что не могут быть расширены для построения множественных выравниваний.

Цель настоящей работы — разработать и реализовать программу построения множественного выравнивания кодирующих последовательностей ДНК с учетом сдвигов рамки считывания. Для ее достижения нами были постав-

лены и решены следующие задачи:

- разработать и реализовать алгоритм парного выравнивания с учетом сдвигов рамки считывания
- реализовать алгоритм кластеризации для перехода от задачи парного к задаче множественного выравнивания
- оценить производительность и качество результата созданной программы

1 Обзор предметной области

Выравнивание аминокислотных или нуклеотидных последовательностей — это процесс сопоставления сравниваемых строк для такого их взаиморасположения, при котором наблюдается максимальное количество совпадений аминокислотных остатков или нуклеотидов [5]. Различают два вида выравнивания: парное (выравнивание двух последовательностей ДНК, РНК или белков) и множественное (выравнивание трех и более последовательностей).

1.1 Существующие методы поиска гомологий в биологических последовательностях

В генетике под гомологиями понимаются участки белков или ДНК, имеющие сходную последовательность аминокислот или нуклеотидов. Обычно существа, у которых есть гомологичные участки белков или ДНК, имеют общего предка, от которого они и получили такой участок. Поскольку в процессе эволюции ДНК подвергается мутациям, эти участки не обязательно идентичны. В них могут быть случайно заменены, добавлены или удалены нуклеотиды или аминокислоты (рисунок 1). Некоторые мутации, такие как транслокации и инверсии, приводят к изменениям, затрагивающим большие участки генома. Такие мутации сложно учитывать, поскольку локальное сходство проверять легче, чем глобальное, а в результате глобальных мутаций участки ДНК могут быть соединены в непредсказуемом порядке.

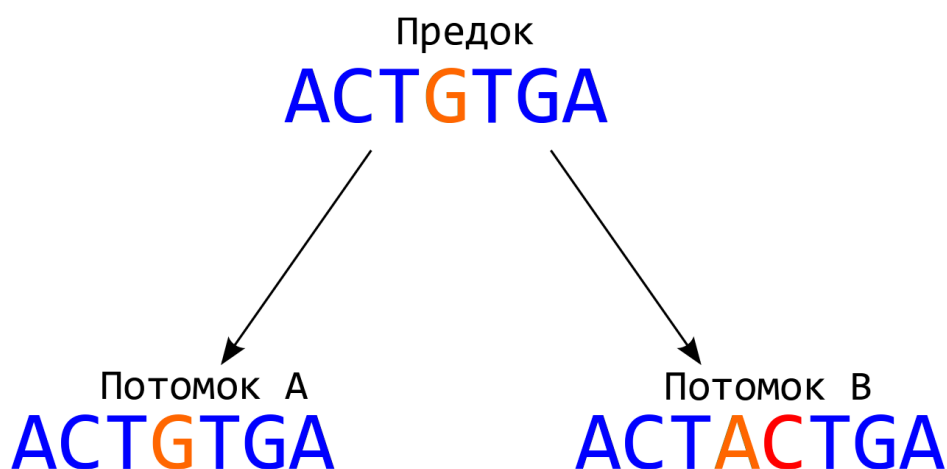


Рисунок 1 – Пример мутации

1.1.1 Алгоритм Нидлмана-Вунша

Одним из наиболее распространенных алгоритмов выравнивания является алгоритм Нидлмана-Вунша [6], основанный на двумерном динамическом программировании. Для своей работы алгоритм использует матрицу сходства, которая указывает, насколько схожими можно считать разные нуклеотиды. Использование матрицы позволяет придавать разный вес разным заменам нуклеотидов. Например, поскольку транзиции более вероятны, чем трансверсии, логично считать последовательности, отличающиеся заменой пурина на пурин или пиримидина на пиримидин, более схожими, чем те, которые отличаются заменой пурина на пиримидин или наоборот. Обычно используется симметричная матрица, однако, применение несимметричной матрицы позволяет различать замены в одну и в другую стороны. На рисунке 2 представлен пример матрицы сходства. Здесь А, Г, Т и Ц обозначают, соответственно, аденин, гуанин, тимин и цитозин, а числа в матрице указывают степень сходства между двумя нуклеотидами.

	А	Г	Т	Ц
А	10	-1	-4	-3
Г	-1	7	-3	-5
Т	-4	-3	8	0
Ц	-3	-5	0	9

Рисунок 2 – Пример матрицы сходства

Еще один параметр алгоритма — штраф за разрыв последовательности. Он может выражаться произвольной функцией от длины и/или направления разрыва. Для определенности будем рассматривать линейный штраф за разрыв, определяющийся параметром d (за разрыв длины n будет начислен штраф $d \cdot n$).

На вход алгоритм получает матрицу сходства S , параметр штрафа d и две последовательности (строки), которые необходимо выровнять. Для получения результата выполняется построение матрицы $F_{i,j}$, где i и j изменяются от нуля до длины, соответственно, первой и второй строк. Вначале алгоритм инициализирует $F_{i,0}$ и $F_{0,j}$ равными, соответственно, $d \cdot i$ и $d \cdot j$ для всех i и j . Затем происходит вычисление оставшихся элементов матрицы по формуле 1.

$$F_{i,j} = \max \begin{cases} F_{i-1,j-1} + S_{A_i,B_j} \\ F_{i-1,j} + d \\ F_{i,j-1} + d \end{cases} \quad (1)$$

После того как матрица посчитана, необходимо определить, каким путем появилось значение в правом нижнем углу. Например, если $F_{i,j} = F_{i-1,j-1} + S_{A_{i-1},B_{j-1}}$, то элемент (i, j) появился из элемента $(i-1, j-1)$, и т. д. Элементы в верхней строке произошли из элементов левее себя, элементы из левого столбца — из элементов выше себя. Переход вида $(i, j) \rightarrow (i-1, j-1)$ означает, что i -му символу в первой строке соответствует j -й символ во второй строке. Переход вида $(i, j) \rightarrow (i-1, j)$ означает, что i -му символу первой строки ничего не соответствует, а переход $(i, j) \rightarrow (i, j-1)$ — что j -му символу второй строки ничего не соответствует. Путь в матрице от левого верхнего угла к правому нижнему даст искомое выравнивание последовательностей.

Очевидно, что алгоритм всегда ищет выравнивание с максимальным счетом, так как, строя матрицу F , он рассматривает всевозможные варианты размещения одной строки относительно другой. Время работы и количество используемой памяти пропорционально произведению длин последовательностей.

1.1.2 Алгоритм Смита-Ватермана

Алгоритм Смита-Ватермана [7] аналогичен алгоритму Нидлмана-Вунша, но решает задачу локального выравнивания: находит подстроки первой и второй строк, обладающие максимальным сходством.

На вход алгоритм получает матрицу сходства S , две последовательности и два вектора I и D , вектор стоимостей добавления и вектор стоимостей удаления, соответственно. Элементы матрицы $F_{i,0}$ и $F_{0,j}$ инициализируются нулями. Вычисление оставшихся элементов происходит по формуле 2.

$$F_{i,j} = \max \begin{cases} F_{i-1,j-1} + S_{A_i,B_j} \\ F_{i-1,j} + D_{A_i} \\ F_{i,j-1} + I_{B_j} \\ 0 \end{cases} \quad (2)$$

Для получения выравнивания необходимо найти максимальный элемент в матрице. Если переходить от этого элемента по цепочке предыдущих, то путь закончится в каком-то нулевом элементе. Индексы этих двух элементов равны индексам начал и концов подстрок: первые индексы — в первой строке, вторые — во второй. Путь интерпретируется так же, как и в алгоритме Нидлмана-Вунша.

Видно, что оба алгоритма похожи друг на друга. Они имеют одинаковую сложность и затраты по памяти, что делает такие алгоритмы неприемлемыми для работы с большим количеством генетического материала.

1.1.3 Алгоритм Хиршберга

Оба предыдущих алгоритма требуют объем памяти, пропорциональный произведению длин выравниваемых последовательностей, что затрудняет обработку больших строк, поэтому очень важно иметь методы, уменьшающие затраты памяти без критического увеличения времени счета. В 1975 году был предложен алгоритм Хиршберга, значительно сокращающий затраты памяти [8]. Он позволяет вычислять оптимальное выравнивание строк длины n и m , используя $O(n + m)$ количество памяти, но примерно вдвое большее времени счета по сравнению с алгоритмом Нидлмана-Вунша.

Идея алгоритма состоит в том, что одна из двух входных последовательностей разбивается на две части, и исходная задача сводится к двум, меньшим, задачам выравнивания второй входной последовательности с каждой из частей. Решение подзадач осуществляется путем аналогичного сведения к подзадачам. На рисунке 3 показана схема разбивки задачи на две подзадачи: верхнюю, которая решается в прямоугольнике A исходной таблицы, и нижнюю — в прямоугольнике B . Последовательности имеют длины n и m , соответственно. Для разбиения каждой задачи на подзадачи необходимо вычислить значение k^* . При этом используется объем памяти, линейно зависящий от m . Верхняя задача заключается в выравнивании строки с длинами не больше $n/2$ и k^* , а нижняя — с длинами не больше $n/2$ и $m - k^*$.

Для представления задач в алгоритме Хиршберга можно использовать бинарные деревья [9]. Узлам дерева соответствуют подзадачи, которые заключаются в выравнивании меньших подпоследовательностей. Каждый узел

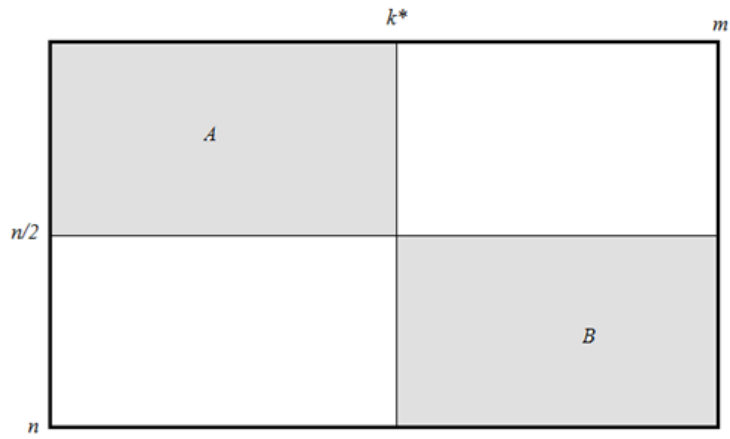


Рисунок 3 – Разделение задачи выравнивания на две подзадачи

дерева хранит в памяти границу прямоугольной области, в которой решается соответствующая задача динамического программирования. Дерево в процессе работы алгоритма строится по уровням. Сначала оно состоит только из корневого узла, который соответствует прямоугольнику $[0, 0] \times [n, m]$. Создание двух узлов эквивалентно разбиению задачи на две подзадачи и разделению области решения на две, меньшего размера.

Алгоритм Хиршберга заключается в обходе полного дерева всех подзадач. Результат выравнивания можно будет получить, если пройти по листьям построенного дерева (рисунок 4). Для оптимизации вычислений можно выполнять обход (решение подзадач) только части вершин дерева: тех, которые удалены от корня на величину, не превосходящую заранее заданную константу h — максимальную глубину обхода дерева. При достижении глубины дерева h или минимального размера прямоугольника применяется алгоритм Нидлмана-Вунша, который работает вдвое быстрее алгоритма Хиршберга.

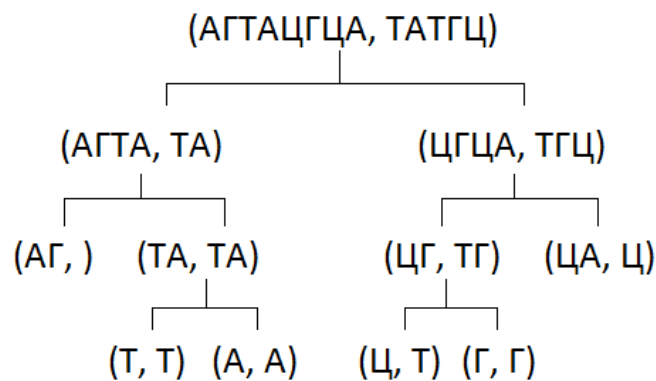


Рисунок 4 – Дерево подзадач для алгоритма Хиршберга

Дополнительное ускорение можно получить за счет распараллеливания. Заметим, что на каждом шаге алгоритма полученные подзадачи никак не связаны между собой, и, следовательно, их решения могут вычисляться в отдельных потоках.

1.2 Алгоритмы множественного выравнивания

В пункте 1.1 были рассмотрены основные подходы для получения парного выравнивания. Для некоторых областей биоинформатики задачу поиска выравнивания необходимо переложить на многомерный случай, например, при реконструкции эволюционной последовательности (получение филогенетических деревьев) или при выявлении шаблона функциональных семейств и сигналов ДНК.

1.2.1 Выравнивание в «кубе»

Рассмотрим задачу выравнивания трех последовательностей: A_1 , A_2 и A_3 . Построим трехмерную матрицу F (рисунок 5) с длинами сторон $len(A_i)$, $i = 1, 2, 3$, где $len(A_i)$ — длина i -ой строки. Аналогично алгоритму Нидлмана-Вунша (пункт 1.1.1) определим значение в ячейке $F_{i,j,k}$ $i = 1 \dots len(A_1)$, $j = 1 \dots len(A_2)$, $k = 1 \dots len(A_3)$ по формуле 3.

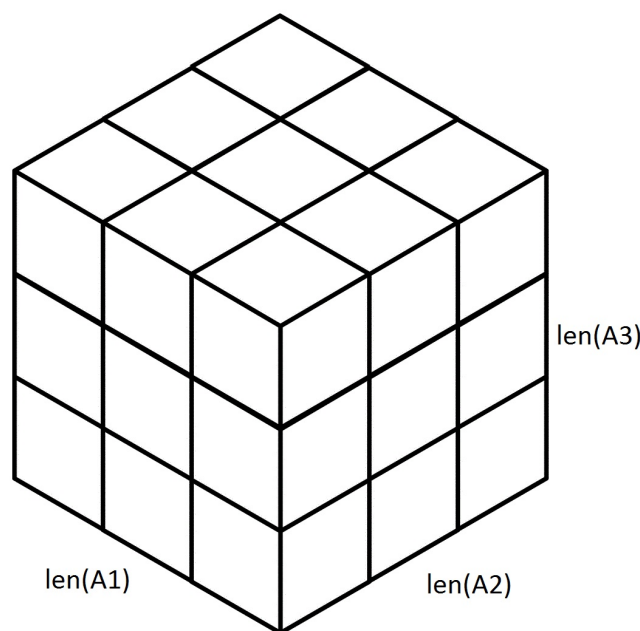


Рисунок 5 – Матрица F для выравнивания трех последовательностей

$$F_{i,j,k} = \max \begin{cases} F_{i-1,j-1,k-1} + S(A_{1_i}, A_{2_j}) + S(A_{1_i}, A_{3_k}) + S(A_{2_j}, A_{3_k}) \\ F_{i-1,j-1,k} + S(A_{1_i}, A_{2_j}) + 2d \\ F_{i-1,j,k-1} + S(A_{1_i}, A_{3_k}) + 2d \\ F_{i,j-1,k-1} + S(A_{2_j}, A_{3_k}) + 2d \\ F_{i-1,j,k} + 3d \\ F_{i,j-1,k} + 3d \\ F_{i,j,k-1} + 3d \end{cases} \quad (3)$$

Можно заметить, что каждая грань куба — это парное выравнивание двух последовательностей с учетом некоторой части третьей, что и дает в итоге полный перебор всех возможных вариантов. Нулевые грани куба $F_{0,j,k}$, $F_{i,0,k}$ и $F_{i,j,0}$ заполняются аналогично алгоритму Нидлмана-Вунша.

Чтобы получить ответ, необходимо найти путь от ячейки $F_{len(A_1), len(A_2), len(A_3)}$, где записан итоговый счет за выравнивание, до $F_{0,0,0}$. Так как имеется всего семь возможных перемещений в кубе и $len(A_1) \cdot len(A_2) \cdot len(A_3)$ ячеек, то сложность алгоритма можно оценить как $O(7 \prod_{i=1}^3 len(A_i))$.

Не составляет большого труда «продлить» аналогичным образом это решение на n -мерный случай и получить «честное» многомерное выравнивание. Под словом «честное» подразумевается, что рассмотрены все возможные варианты выравнивания последовательностей, и полученный результат всегда имеет максимальный счет. Единственный недостаток — слишком большая вычислительная сложность алгоритма: $O((2^n - 1) \prod_{i=1}^n len(A_i))$, что делает такой подход совершенно неприменимым для выравнивания большого числа и/или длинных последовательностей.

1.2.2 Выравнивание выравниваний. Алгоритм Clustal

Другой подход заключается в получении парного выравнивания между первыми двумя последовательностями, после чего полученный результат выравнивается с третьей и так далее. То есть, если f — функция вычисления парного выравнивания, а A_1, \dots, A_n — выравниваемые последовательности,

то алгоритм можно условно записать формулой 4.

$$f(f(f(\dots f(f(A_1, A_2), A_3) \dots), A_{n-1}), A_n) \quad (4)$$

Очевидно, что результат алгоритма будет зависеть от порядка исходных последовательностей. Существуют различные соображения по поводу наиболее правильного выбора этого порядка. Можно не ограничиваться выравниваниями типа «последовательность против выравнивания», но также производить выравнивание «выравнивание против выравнивания». Например, если есть четыре последовательности, из которых первая очень похожа на четвертую, вторая — на третью, а гомология между остальными парами (1-2, 1-3, 2-4, 3-4) более слабая, то разумно сначала сделать два парных выравнивания: первой последовательности с четвертой и второй с третьей, а затем уже выровнять эти два выравнивания друг с другом.

Похожим образом работает Clustal — один из самых популярных алгоритмов множественного выравнивания. По сути, это жадный алгоритм с «умным» способом выбора пар. Сначала происходит построение всех парных выравниваний, после чего по полученным результатам строится «дерево-подсказка». На рисунке 6 представлен пример возможного дерева. Для четырех последовательностей A_1, A_2, A_3 и A_4 строится таблица (на рисунке слева), числа в которой обозначают их схожесть друг с другом. Видно, что самые близкие последовательности — A_1 и A_3 , и их выравнивание будет первым, затем оно выравнивается с A_4 , а последнее — с A_2 .

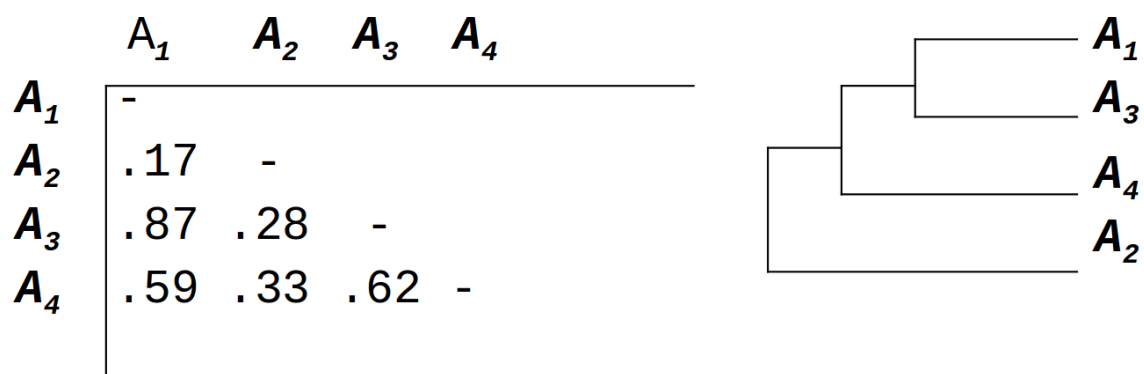


Рисунок 6 – Построение дерева-подсказки для алгоритма Clustal

1.3 Выравнивание с учетом открытых рамок считывания

Изменению числа нуклеотидных пар в цепи ДНК способствуют воздействия на генетический материал некоторых химических веществ, например акридиновых соединений [10]. Деформируя структуру двойной спирали ДНК, они приводят к вставке дополнительных оснований или их выпадению при репликации. Однако, куда более вероятно возникновение ошибки на этапе секвенирования.

Рассмотренные в пунктах 1.1 и 1.2 алгоритмы множественного и парного выравниваний применимы для любых, не обязательно биологических, последовательностей, например, текстов статей или исходных кодов программ на предмет поиска плагиата. Изложенные выше методы подходят к задаче выравнивания исключительно на математическом уровне, в том плане, что они производят поиск выравнивания с максимальным счетом, совершенно не опираясь на логический смысл входных данных. Возвращаясь непосредственно к задачам биоинформатики, для поиска «правильного» выравнивания последовательностей необходимо использовать более сложные алгоритмы, учитывающие трансляцию полученного результата на уровень аминокислот.

1.3.1 Трехэтапный подход

Один из самых простых способов построения «правильного» выравнивания — произвести трансляцию нуклеотидной последовательности в аминокислотную по всем возможным рамкам считывания, после чего произвести выравнивание «классическими» алгоритмами, и, в завершение, транслировать полученный белок обратно в последовательность нуклеотидов. Для автоматизации выполнения этих трех шагов были разработаны программы revTrans [11], transAlign [12], TranslatorX [13] и PAL2NAL [14], которая, по сравнению с остальными, дополнительно позволяет указать позиции известных рамок считывания.

Основным недостатком этого трехступенчатого подхода является его неспособность справляться с неожиданной заменой рамки считывания. Все последующие этапы алгоритма после неправильной первой трансляции уже никак

не смогут это исправить. В лучшем случае этот ошибочный перевод быстро приведет к появлению стоп-кодона, который будет сигналом для предупреждения пользователя о неправильной трансляции. В худшем случае программа построит выравнивание, которое будет очень сильно расходиться с действительностью.

1.3.2 Двухуровневое выравнивание

В 1994 году был предложен еще один подход для решения этой задачи. Автором была предложена модель, по которой штраф за выравнивание являлся сочетанием двух штрафов: на аминокислотном и нуклеотидном уровнях [15]. Он рассмотрел частный случай идеализированной эволюции исходных последовательностей, при котором инсерции допустимы только на аминокислотном уровне (запрет на сдвиг рамки считывания), а штраф за выравнивание вычислялся просто как сумма штрафов на обоих уровнях. Предложенный алгоритм выравнивания двух последовательностей длины n и m имел сложность $O(n^2m^2)$.

Позже этот алгоритм был оптимизирован и перенесен на модель с аффинными штрафами и итоговой сложностью $O(nm)$ [16]. Эти улучшения казались многообещающими, так как асимптотическая сложность алгоритма получилась точно такая же, как и у классических методов выравнивания. Однако, необходимо отметить, что постоянный множитель, спрятанный в оценке сложности O , ограничивает применение этого алгоритма на практике. Для получения парного выравнивания алгоритму необходимо вычислить примерно $400nm$ значений, что, к сожалению, делает его неприменимым для задачи множественного выравнивания.

1.3.3 MACSE

MACSE (Multiple Alignment of Coding SEquences Accounting for Frameshifts and Stop Codons) — это программный комплекс, позволяющий строить множественные выравнивания кодирующих последовательностей с учетом существующих рамок считывания и стоп-кодонов [17]. Кроме задачи выравнивания, он может быть применен для обнаружения недокументированных рамок

считывания в публичных базах данных.

Алгоритм MACSE основывается на идее двухуровневого выравнивания, но требует меньше времени на вычисление парного выравнивания, благодаря чему возможно его расширение на многомерный случай. Для получения многомерного выравнивания n строк $S_1 \dots S_n$ MACSE производит выравнивание выравниваний, выбирая порядок через дерево-подсказку, как и алгоритм Clustal.

2 Алгоритм выравнивания

За основу был взят алгоритм Нидлмана-Вунша и произведена его модификация, благодаря которой заполнение очередной ячейки матрицы происходит с учетом трансляции текущего триплета нуклеотидов в аминокислоту. Таким образом, полученное решение строит двухуровневое выравнивание последовательностей S_1 и S_2 за время $O(\text{len}(S_1) \cdot \text{len}(S_2))$. Константа, спрятанная в оценке сложности O , равна 25, что позволяет продлить алгоритм до задачи множественного выравнивания.

2.1 Принятые обозначения

Пусть S_1 и S_2 — некоторые последовательности нуклеотидов. Введем следующие обозначения:

- $\text{len}(S_k)$ — как и в предыдущих пунктах, длина последовательности S_k
- $S_k[i : j]$ — подпоследовательность S_k с i -го по j -ый нуклеотид. Запись $S_k[i : i]$, или просто $S_k[i]$, обозначает i -ый нуклеотид S_k , а в случае $j < i$ $S_k[i : j]$ является пустой последовательностью
- $\mathcal{A}(S_i, S_j)$ — оптимальное выравнивание последовательностей S_i и S_j
- $\text{cost}(\mathcal{A}(S_i, S_j))$ — численная характеристика полученного выравнивания, вычисляемая по рекурсивной формуле
- $\text{cost}(' -')$ — штраф (число) за разрыв последовательности
- $\text{cost}('!')$ — штраф за разрыв рамки считывания
- $\text{cost}('*')$ — штраф за появление стоп-кодона не в конце последовательности
- $\sigma(X, Y)$ — оценка за сопоставление нуклеотидов (или аминокислот) X и Y

Перепишем в новых обозначениях рекурсивную формулу $\text{cost}(\mathcal{A}(S_1, S_2))$ для классического алгоритма Нидлмана-Вунша (5).

$$cost(\mathcal{A}(S_1[1 : i], S_2[1 : j])) = \max \left\{ \begin{array}{l} cost(\mathcal{A}(S_1[1 : i - 1], S_2[1 : j - 1])) \\ \quad + \sigma(S_1[i], S_2[j]) \\ cost(\mathcal{A}(S_1[1 : i - 1], S_2[1 : j])) \\ \quad + cost(' -') \\ cost(\mathcal{A}(S_1[1 : i], S_2[1 : j - 1])) \\ \quad + cost(' -') \end{array} \right. \quad (5)$$

На каждом шаге рекурсии происходит уменьшение i и/или j на единицу. Условие продолжения рекурсии: $i > 0$ и $j > 0$. Граничные значения при $i = 0$ и $j = 0$ заполняются по формулам 6.

$$\begin{aligned} cost(\mathcal{A}(-, S_2[1 : j])) &= j \cdot cost(' -') \\ cost(\mathcal{A}(S_1[1 : i], -)) &= i \cdot cost(' -') \end{aligned} \quad (6)$$

Для получения выравнивания необходимо запомнить оптимальный выбор на каждом шаге рекурсии $cost(\mathcal{A}(S_1, S_2)) = cost(\mathcal{A}(S_1[1 : len(S_1)], S_2[1 : len(S_2)]))$ и выполнить восстановление ответа (пункт 1.1.1).

При построении выравнивания с учетом трансляции нуклеотидов необходимо связать уровни нуклеотидов и аминокислот. Введем дополнительную функцию $\pi(S)$, которая по входной последовательности нуклеотидов S строит ее трансляцию AA_S . Трансляция происходит по первой рамке считывания (начиная с первого нуклеотида). Для обозначения неполных кодонов используется символ '!', а стоп-кодона переводятся в символы '*' без остановки трансляции.

Запишем в новых обозначениях рекурсивную формулу $cost(\mathcal{A}(S_1, S_2))$ для алгоритма двухуровневого выравнивания (7), рассмотренном в пункте 1.3.2, где $AA_1 = \pi(S_1[3i - 2 : 3i])$ и $AA_2 = \pi(S_2[3j - 2 : 3j])$.

$$cost(\mathcal{A}(S_1[1 : 3i], S_2[1 : 3j])) = max \begin{cases} cost(\mathcal{A}(S_1[1 : 3i - 3], \\ S_2[1 : 3j - 3])) + \sigma(AA_1, AA_2) \\ cost(\mathcal{A}(S_1[1 : 3i - 3], S_2[1 : j])) \\ + cost(' - ') \\ cost(\mathcal{A}(S_1[1 : i], S_2[1 : 3j - 3])) \\ + cost(' - ') \end{cases} \quad (7)$$

В рассмотренных выше алгоритмах используется линейный штраф за разрыв последовательности. Для использования аффинного штрафа введем еще два параметра:

- $cost(gap_open)$ — штраф за открытие разрыва
- $cost(gap_extension)$ — штраф за продолжение разрыва

Таким образом, за разрыв длины l будет начислен штраф $l \cdot cost(' - ')$ при линейном или $cost(gap_open) + l \cdot cost(gap_extension)$ — при аффинном подходе.

2.2 Построение парного выравнивания

В отличие от алгоритма Нидлмана-Вунша, учитывающего в своей рекурсивной формуле лишь три варианта перехода (инсерция, делеция или совпадение нуклеотидов), разработанное решение рассматривает дополнительные возможности построения выравнивания с учетом образующихся аминокислот и сдвигов рамки считывания. В некоторых случаях становится выгоднее оставлять друг напротив друга различающиеся нуклеотиды, чтобы в итоге получить одинаковые аминокислоты и не допускать излишних разрывов в последовательности. Кроме этого, сигналом о построении неправильного выравнивания может служить внезапное появление стоп-кодона, на что классические алгоритмы не обращают внимания. На рисунке 7 показан пример двух выравниваний, построенных с помощью алгоритма Нидлмана-Вунша и собственного решения.

Необходимо отметить, что, регулируя параметры штрафа за открытие и продолжение разрыва в последовательности, а также подобрав определенную

матрицу замен нуклеотидов, можно добиться более качественного выравнивания для алгоритма Нидлмана-Вунша на этом тесте, однако, это никак не решает проблем одноуровневого подхода. Для одного набора входных данных выбранные значения будут давать хороший результат, а для другого — нет. Постоянный подбор оптимальных параметров крайне неэффективен.

ATTTCAGG-TTGTC
A--TCC-GGCTTGTC

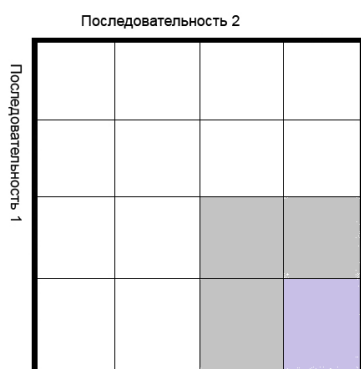
а) Выравнивание по алгоритму
Нидлмана-Вунша

ATTTCAGGTTG-TC
ATC---CGGCTTGTC

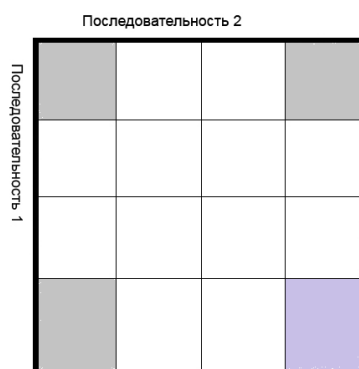
б) Выравнивание по разработанному
алгоритму

Рисунок 7 – Пример выравнивания двух последовательностей без учета аминокислотного уровня (а), и с его учетом (б). Одинаковым цветом отмечены одинаковые аминокислоты; неполные кодоны отмечены черным.

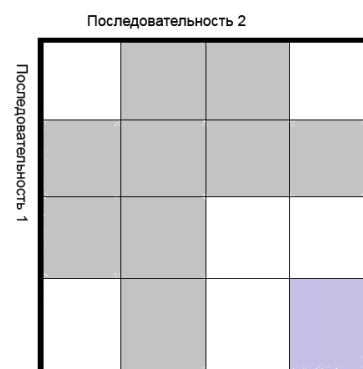
Созданный алгоритм двухуровневого выравнивания на каждом этапе выбора рассматривает все возможные варианты сопоставления от нуля до трех нуклеотидов из каждой последовательности. Таким образом, общее число возможных переходов: $\sum_{i=1}^3 2 \cdot 2^i - 1 = 25$. Для наглядности эти варианты представлены на рисунке 8.



а) Сопоставление
нуклеотидов
(3 возможных
перехода)



б) Сопоставление
аминокислот
(3 возможных
перехода)



в) Сопоставление
неполных кодонов
(19 возможных
переходов)

Рисунок 8 – Выбор оптимального шага. Правый нижний угол таблицы представляет собой текущую позицию. Серым выделены рассматриваемые клетки для перехода.

При сопоставлении неполных кодонов выходит так много вариантов, потому что до рассматриваемых клеток перехода существует несколько возмож-

ных путей. Алгоритм учитывает их все, чтобы гарантировать оптимальное выравнивание на выходе. Ниже представлен псевдокод функции расчета $cost(\mathcal{A}(S_1, S_2))$ (листинг 1).

Листинг 1 Рекурсивный алгоритм построения оптимального парного выравнивания двух кодирующих последовательностей нуклеотидов

```

procedure CalcAlign( $S_1, S_2, i, j$ )
    if  $i = 0$  AND  $j = 0$  then                                 $\triangleright$  Проверка граничных условий
        return 0
    end if
    if  $i = 0$  OR  $j = 0$  then
        return  $(i + j - 1) * cost(gap\_extension) + cost(gap\_open)$ 
    end if
     $AA_1 \leftarrow \pi(S_1[i - 2 : i])$                                  $\triangleright$  Трансляция триплетов
     $AA_2 \leftarrow \pi(S_2[j - 2 : j])$ 
     $stopS_1 \leftarrow 0$ 
     $stopS_2 \leftarrow 0$      $\triangleright$  Проверка на появление преждевременных стоп-кодонов
    if  $i \neq len(S_1)$  AND  $AA_1 = *$  then
         $stopS_1 \leftarrow cost('*')$ 
    end if
    if  $j \neq len(S_2)$  AND  $AA_2 = *$  then
         $stopS_2 \leftarrow cost('*')$ 
    end if                                 $\triangleright$  Перебор вариантов: сопоставление аминокислот
     $score \leftarrow \max \left\{ \begin{array}{l} CalcAlign(S_1, S_2, i - 3, j - 3) + \sigma(AA_1, AA_2) \\ CalcAlign(S_1, S_2, i - 3, j) + stopS_1 + cost('-') \\ CalcAlign(S_1, S_2, i, j - 3) + stopS_2 + cost('-') \end{array} \right.$ 
     $\triangleright$  Перебор вариантов: сопоставление неполных триплетов
     $score \leftarrow \max \left\{ \begin{array}{l} CalcAlign(S_1, S_2, i - 2, j - 2) + \sigma(S_1[i - 1], S_2[j - 1]) \\ \quad + \sigma(S_1[i], S_2[j]) + 2 \cdot cost('!') \\ CalcAlign(S_1, S_2, i - 2, j - 1) \\ \quad + \sigma(S_1[i - 1], S_2[j]) + 2 \cdot cost('!') \\ CalcAlign(S_1, S_2, i - 2, j - 1) \\ \quad + \sigma(S_1[i], S_2[j]) + 2 \cdot cost('!') \\ CalcAlign(S_1, S_2, i - 2, j) + cost('!') + cost('-') \\ CalcAlign(S_1, S_2, i - 1, j - 2) \\ \quad + \sigma(S_1[i], S_2[j - 1]) + 2 \cdot cost('!') \\ CalcAlign(S_1, S_2, i - 1, j - 2) \\ \quad + \sigma(S_1[i], S_2[j]) + 2 \cdot cost('!') \\ score \end{array} \right.$ 

```

```

    
$$score \leftarrow \max \left\{ \begin{array}{l}
        CalcAlign(S_1, S_2, i, j - 2) + cost('!') + cost(' - ') \\
        CalcAlign(S_1, S_2, i - 3, j - 2) + \sigma(S_1[i - 2], S_2[j - 1]) \\
        \quad + \sigma(S_1[i - 1], S_2[j]) + stopS_1 + cost('!') \\
        CalcAlign(S_1, S_2, i - 3, j - 2) + \sigma(S_1[i - 1], S_2[j - 1]) \\
        \quad + \sigma(S_1[i], S_2[j]) + stopS_1 + cost('!') \\
        CalcAlign(S_1, S_2, i - 3, j - 2) + \sigma(S_1[i - 2], S_2[j - 1]) \\
        \quad + \sigma(S_1[i], S_2[j]) + stopS_1 + cost('!') \\
        CalcAlign(S_1, S_2, i - 3, j - 1) + \sigma(S_1[i - 2], S_2[j]) \\
        \quad + stopS_1 + cost('!') \\
        CalcAlign(S_1, S_2, i - 3, j - 1) + \sigma(S_1[i - 1], S_2[j]) \\
        \quad + stopS_1 + cost('!') \\
        CalcAlign(S_1, S_2, i - 3, j - 1) + \sigma(S_1[i], S_2[j]) \\
        \quad + stopS_1 + cost('!') \\
        CalcAlign(S_1, S_2, i - 2, j - 3) + \sigma(S_1[i - 1], S_2[j - 2]) \\
        \quad + \sigma(S_1[i], S_2[j - 1]) + stopS_2 + cost('!') \\
        CalcAlign(S_1, S_2, i - 2, j - 3) + \sigma(S_1[i - 1], S_2[j - 1]) \\
        \quad + \sigma(S_1[i], S_2[j]) + stopS_2 + cost('!') \\
        CalcAlign(S_1, S_2, i - 2, j - 3) + \sigma(S_1[i - 1], S_2[j - 2]) \\
        \quad + \sigma(S_1[i], S_2[j]) + stopS_2 + cost('!') \\
        CalcAlign(S_1, S_2, i - 1, j - 3) + \sigma(S_1[i], S_2[j - 2]) \\
        \quad + stopS_2 + cost('!') \\
        CalcAlign(S_1, S_2, i - 1, j - 3) + \sigma(S_1[i], S_2[j - 1]) \\
        \quad + stopS_2 + cost('!') \\
        CalcAlign(S_1, S_2, i - 1, j - 3) + \sigma(S_1[i], S_2[j]) \\
        \quad + stopS_2 + cost('!') \\
        score
    \end{array} \right.$$

    
$$score \leftarrow \max \left\{ \begin{array}{l}
        \triangleright \text{Перебор вариантов: сопоставление нуклеотидов} \\
        CalcAlign(S_1, S_2, i - 1, j - 1) + \sigma(S_1[i], S_2[j]) \\
        \quad + 2 \cdot cost('!') \\
        CalcAlign(S_1, S_2, i - 1, j) + cost('!') + cost(' - ') \\
        CalcAlign(S_1, S_2, i, j - 1) + cost('!') + cost(' - ') \\
        score
    \end{array} \right.$$

    return score
end procedure

```

Необходимо отметить, что при построении ответа нужно делать восстановление рамки в тех случаях, когда алгоритм выбрал в качестве оптимального шага сопоставление неполных триплетов (рисунок 9).

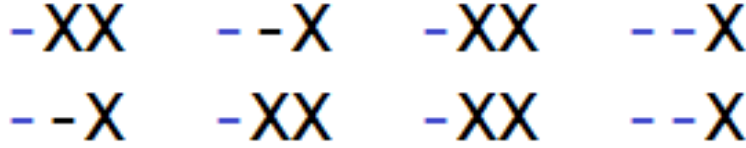


Рисунок 9 – Сопоставление неполных триплетов. Цветом указаны разрывы, добавленные для сохранения текущей рамки считывания.

2.3 Алгоритм кластеризации

Для построения множественного выравнивания используется идея выравнивания выравниваний (пункт 1.2.2). Порядок объединения последовательностей определяется по методу невзвешенного попарного среднего (UPGMA — Unweighted Pair Group Method with Arithmetic mean) [18]. Для использования этого алгоритма кластеризации необходимо определить функцию расстояния между выравниваемыми строками: $dist(S_i, S_j) \rightarrow \mathbb{R}$. Последовательность S_1 ближе к последовательности S_2 , чем к S_3 , если $dist(S_1, S_2) < dist(S_1, S_3)$. Чем выше значение $dist(S_i, S_j)$, тем более похожими считаются строки S_i и S_j .

В качестве функции расстояния можно использовать $cost(\mathcal{A}(S_i, S_j))$, однако, ее вычисление требует больших вычислительных ресурсов. Альтернативный подход к оценке схожести двух строк — сравнение их подстрок фиксированной длины.

Пусть имеются две последовательности S_1 и S_2 . Рассмотрим все их подстроки длины k : $S_1[i : i+k-1] \in A_1, i = 1 \dots len(S_1) - k + 1$ и $S_2[i : i+k-1] \in A_2, i = 1 \dots len(S_2) - k + 1$. Расстояние между последовательностями — это количество одинаковых подпоследовательностей $dist(S_1, S_2) = |A_1 \cap A_2|$. Данный метод имеет меньшую точность, по сравнению с построением оптимального парного выравнивания, однако, он проще и требует меньше операций вычисления при небольших значениях k .

На начальном этапе алгоритма кластеризации происходит вычисление расстояния между всеми последовательностями, заполняется таблица $D = dist(S_i, S_j) \ i, j = 1 \dots n$. Далее, на каждом шаге из матрицы D выбирается

максимальное значение, которое определяет текущие кластеры i и j для объединения, после чего происходит пересчет расстояний от нового кластера до всех остальных по формуле 8

$$D((i, j), w) = \frac{T_i D(i, w) + T_j D(j, w)}{T_i + T_j} \quad (8)$$

где T_k — количество последовательностей в кластере k . Старые i и j столбцы и строки таблицы D становятся недействительными. Таким образом, с каждой новой итерацией алгоритма происходит уменьшение числа кластеров на единицу: два кластера собираются в один. Финальное объединение двух последних кластеров даст итоговое выравнивание, содержащее все исходные последовательности нуклеотидов. На рисунке 10 представлен пример работы алгоритма.

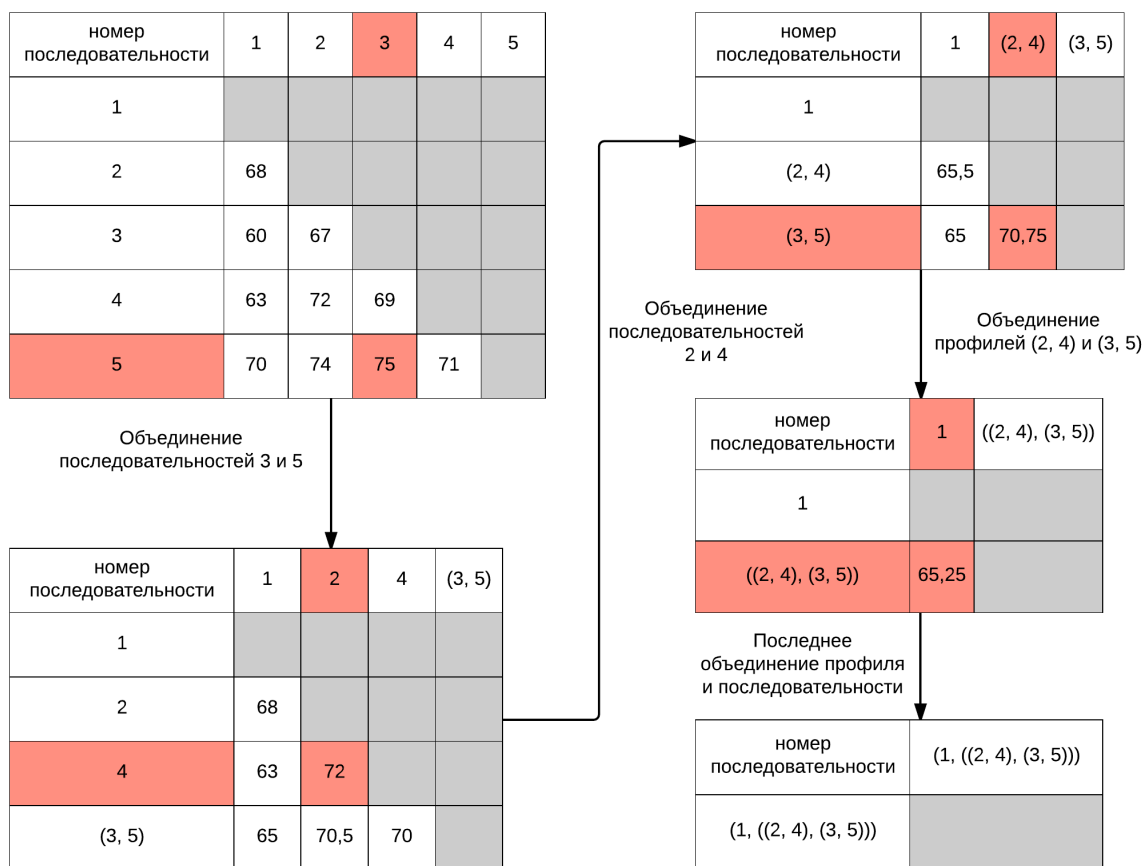


Рисунок 10 – Изменение матрицы расстояний в процессе кластеризации. Красным цветом обозначены объединяемые кластеры.

Необходимо отметить, что процесс кластеризации необязательно продолжаться до объединения всех кластеров в один. Если найденный в таблице D

максимум чересчур мал, то это свидетельствует о слабом родстве последовательностей, и, чтобы не испортить построенное выравнивание, алгоритм можно остановить.

2.4 Построение множественного выравнивания

Введем обозначение «профиль» P как набор выравненных строк. Используя рассмотренный выше алгоритм парного выравнивания, можно объединять две последовательности в один профиль $P_{12} = \mathcal{A}(S_1, S_2)$. Для построения множественного выравнивания необходимо ввести операции сложения профилей $P_i + P_j$ и профиля со строкой $P_i + S_j$.

При переходе от последовательностей к профилям достаточно определиться с интерпретацией $\sigma(P_1[i], P_2[j])/\sigma(P_1[i], S_2[j])$ и $\sigma(\pi(P_1[i-2:i]), \pi(P_2[j-2:j]))/\sigma(\pi(P_1[i-2:i]), \pi(S_2[j-2:j]))$, чтобы использовать алгоритм парного выравнивания для операций объединения. В отличие от строки, у профиля по i -ой позиции находится набор нуклеотидов из всех входящих в него последовательностей по i -ому индексу (рисунок 11). Для каждого такого столбца $i = 1 \dots \text{len}(P)$, где $\text{len}(P)$ — длина профиля, можно рассчитать частоты встречающихся символов.



Рисунок 11 – Профиль из четырех последовательностей и распределение частот для i -го столбца

Тогда определим $\sigma(P[i], S[j])$ по формуле 9, где $\lambda_i[N]$ — частота нуклеотида N в i -ом столбце профиля P .

$$\sigma(P[i], S[j]) = \sigma('A', S[j])\lambda_i['A'] + \sigma('C', S[j])\lambda_i['C'] + \dots + \lambda_i['-']cost(' -') \quad (9)$$

Аналогичным образом вычисляется $\sigma(\pi(P[i-2:i]), \pi(S[j-2:j]))$, только в данном случае $\lambda_{i-2:i}$ содержит частоты распределения аминокислот в профиле.

В случае объединения двух профилей необходимо рассчитать частоты для каждого из них λ_i и λ_j , после чего произвести полный перебор по всем возможным вариантам сопоставления, как в формуле 9. Результат вычислений нормируется, чтобы счет за сопоставление столбцов никак не зависел от количества последовательностей в профилях.

Таким образом, итоговый алгоритм множественного выравнивания производит объединение исходных строк в профили, определяя порядок по алгоритму кластеризации. Для сложения профилей и строк используется алгоритм парного двухуровневого выравнивания.

3 Программная реализация

Разработанный алгоритм реализован на языке программирования C++ без использования сторонних библиотек и может быть использован на всех основных операционных системах: Windows, Linux, Mac OS. Для удобства использования основного приложения был создан веб-интерфейс, позволяющий осуществлять взаимодействие с программой через графическую оболочку. Поскольку задача множественного выравнивания может требовать существенных вычислительных ресурсов, практически решать ее, используя мощную вычислительную платформу, а взаимодействие с ней осуществлять через веб-интерфейс. Кроме того, для возможности внедрения функций построения парного и множественного выравниваний в другие программы они были реализованы в виде статических библиотек.

3.1 Структуры данных

В данном разделе описаны используемые в программе структуры данных для хранения и обработки кодирующих последовательностей ДНК. Всего разработано и реализовано три объекта: структура *BioSeq*, описывающая биологические последовательности, класс построения парного выравнивания *PairwiseAlign* и структура *Profile* для работы с набором последовательностей. Все они входят в состав собранных статических библиотек и могут быть внедрены в другие проекты.

3.1.1 Представление биологических последовательностей

Для представления биологических последовательностей используется структура *BioSeq* (листинг 2) с двумя строковыми полями:

- *name* — идентификатор последовательности
- *nt_seq* — последовательность нуклеотидов

Ради удобства использования структуры *BioSeq* был определен оператор индексирования `[]` и функция *Length*, возвращающая текущую длину последовательности. Чтобы иметь связь нуклеотидного и аминокислотного

уровней, реализованы методы, позволяющие получить как трансляцию всей строки по первой рамке считывания *std :: string GetAAseq() const*, так и конкретного триплета, начинающегося с *i*-го индекса *char TranslateNTtoAA(int i) const*. Также необходимо иметь возможность вставлять разрывы в последовательность, для чего были добавлены методы *InsertGap(int pos)* и *InsertGap(int pos, int count)*, добавляющие 1 или *count* разрывов по индексу *pos*. Печать реализована через перегруженную операцию *<<*. Независимый вывод результатов работы программы как на нуклеотидном, так и на аминокислотном уровнях, осуществляется с помощью методы *void PrintNT(std :: ostream& out)* и *void PrintAA(std :: ostream& out)*, которые отправляют результат выравнивания на обоих уровнях в указанный поток *out*.

Листинг 2 Структура представления биологических последовательностей BioSeq

```
struct BioSeq {
    std::string name;
    std::string nt_seq;
    // constructor
    BioSeq(std::string n, std::string s): name(n), nt_seq(s)
    { }
    // destructor
    ~BioSeq() { }
    // accessors
    int Length() { return nt_seq.length(); }
    char operator [] (int index) { return nt_seq[index]; }
    void InsertGap(int pos, int count);
    void InsertGap(int pos);
    // translator
    std::string GetAAseq() const;
    char TranslateNTtoAA(int i) const;
    // printer
    void PrintAA(std::ostream& out);
    void PrintNT(std::ostream& out);
    friend std::ostream& operator << (std::ostream& stream,
                                      const BioSeq& data);
};
```

3.1.2 Класс построения парного выравнивания

Вычисление оптимального выравнивания двух последовательностей реализовано в виде класса *PairwiseAlign* (листинг 3). Таким образом, в одном объекте собираются и данные, и методы их обработки, что упрощает экспорт алгоритма парного выравнивания в другой проект.

Работа с классом подразумевает его использование для повторного решения задачи построения парного выравнивания на различных входных данных. Поэтому класс не содержит полей с исходными последовательностями, в нем хранятся только структуры для построения решения и параметры выравнивания, в частности, штрафы за разрывы, смещение рамки, появление преждевременного стоп-кодона, а также матрицы замен нуклеотидов и аминокислот. Входные данные передаются через метод *Align(const BioSeq* seq1, const BioSeq* seq2)*, реализующий алгоритм парного выравнивания (2.2). Для увеличения производительности программы вместо использования рекурсивной функции *CalcAlign* (листинг 1) таблицы оптимальных ходов заполняются как в классическом алгоритме Нидлмана-Вунша.

Для хранения матриц замен используются два массива: *nt_score_matrix* (нуклеотиды) и *aa_score_matrix* (аминокислоты), размерностью $128 \cdot 128$ элементов, поскольку символы строк представлены одним байтом, принимающим значения от 0 до 127 (рисунок 12).

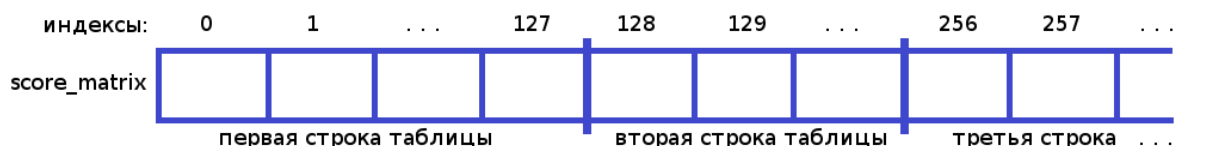


Рисунок 12 – Представление матрицы замен в памяти компьютера

Используя данный подход, на вход программе выравнивания можно подавать последовательности с символами из произвольного алфавита. Для этого необходимо лишь определить соответствующие матрицы замен и предоставить функцию для трансляции кодонов. Адресация по массиву определена следующим образом:

$$nt_score_matrix[(unsigned\ char)'A' * 128 + (unsigned\ char)'C']$$

Пример отражает цену за сопоставление аденина (A) и цитозина (C). В общем случае вместо конкретных нуклеотидов в формулу подставляются i -ый и j -ый символы строк: $seq1[i]$, $seq2[j]$.

Листинг 3 Класс построения оптимального выравнивания двух последовательностей

```
class PairwiseAlign {
private:
    int gap_open, gap_extension, gap_frame, stop_cost;
    int nt_score_matrix[128*128];
    int aa_score_matrix[128*128];
    int seq1_length, seq2_length;
    int matrix_size = 0;
    int* F = NULL; // score matrix
    int* W = NULL; // way matrix
    void NewScoreMatrix(const char* file_name,
                        int* score_matrix);

public:
    PairwiseAlign(const char* nt_score_matrix,
                  const char* aa_score_matrix, int stop_cost,
                  int gap_open, int gap_extension, int frame_gap);
    ~PairwiseAlign() {
        if (F) delete[] F;
        if (W) delete[] W;
    }

    // calc align
    int Align(const BioSeq* seq1, const BioSeq* seq2);
    std::pair<std::string, std::string> GetAlign();

    // change substitution matrices
    void ChangeNTscoreMatrix(const char* file_name) {
        return NewScoreMatrix(file_name, nt_score_matrix);
    }
    void ChangeAAscoreMatrix(const char* file_name) {
        return NewScoreMatrix(file_name, aa_score_matrix);
    }

    // accessors
    int GetGapOpen() { return gap_open; }
```

```

int GetGapExtension() { return gap_extension; }
int GetGapFrame() { return gap_frame; }
int GetStopCost() { return stop_cost; }
};

```

3.1.3 Профили

Выравненные строки сохраняются в структуру *Profile* (листинг 4). Для экономии памяти и времени в профили заносятся не копии последовательностей, а только указатели на них. Функция объединения записана в виде перегруженной операции $+$. Кроме этого, профили имеют методы для вычисления оценки за сопоставление двух столбцов нуклеотидов и аминокислот. Так же, как и для структуры *BioSeq*, реализована возможность вставки разрывов по указанному индексу заданной длины. Дополнительно у профилей имеется массив *frequency*, содержащий частоты распределения нуклеотидов и аминокислот в конкретной позиции. Для его заполнения реализованы методы *CalcFrequenciesNT* и *CalcFrequenciesAA*.

Листинг 4 Структура профилей

```

struct Profile {
    Profile();
    ~Profile();
    // data
    std::vector<BioSeq*> sequences;
    int frequency[128];
    // alignment of profiles and sequence
    Profile& operator + (Profile& another);
    Profile& operator + (BioSeq* sequence);
    // get score in column
    float ColumnNTscore(BioSeq* seq, int index1,
                        int index2);
    float ColumnAAscore(BioSeq* seq, int index1,
                        int index2);
    float ColumnNTscore(Profile& another, int index1,
                        int index2);
    float ColumnAAscore(Profile& another, int index1,
                        int index2);

```

```

// fill an array of frequency
void CalcFrequenciesNT(int position);
void CalcFrequenciesAA(int position);
// insert gaps in sequences
void InsertGap(int pos, int count);
void InsertGap(int pos);
};

```

Предложенное нами программное решение обеспечивает защиту последовательностей, входящих в состав профиля, от изменения или удаления до конца построения выравнивания. В противном случае возможно было бы появление ошибок выполнения или ухудшение качества результата. При экспорте разработчик должен уделить внимание этому аспекту.

3.2 Общая схема работы

На рисунке 13 изображена блок-схема работы программы. Основные этапы реализованы в виде отдельных, независимых друг от друга блоков, что дает возможность модифицировать их или заменить новыми.

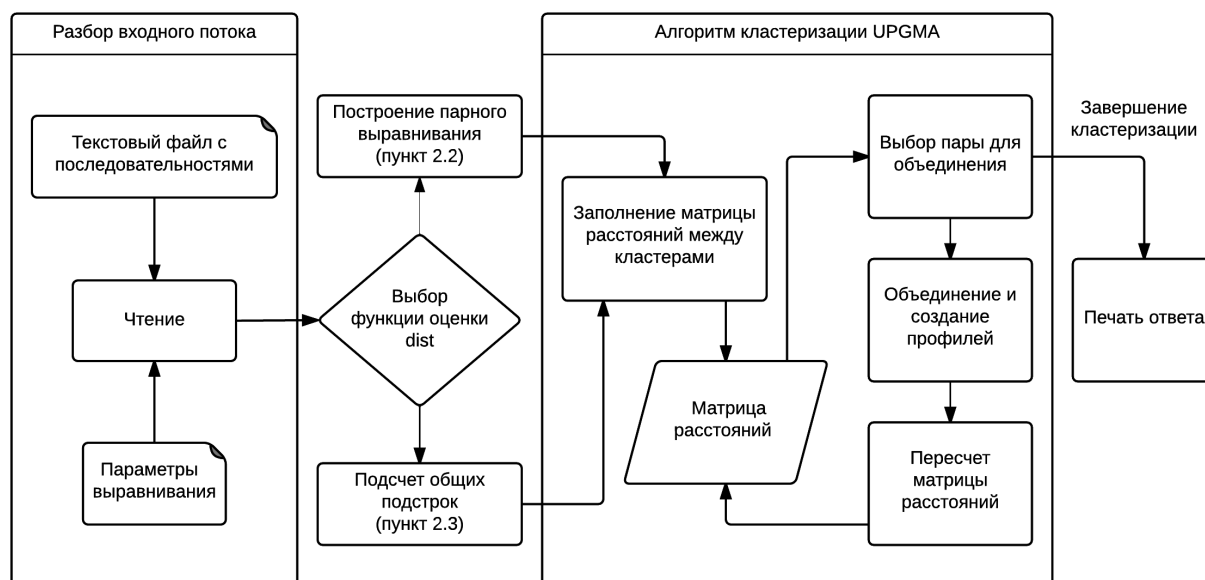


Рисунок 13 – Блок-схема работы программы

3.2.1 Чтение входных данных

На вход программе подается текстовый файл с последовательностями в формате FASTA [19]. Строка, начинающаяся с символа '>', называется строкой описания. Она содержит имя последовательности и некоторую дополнительную информацию, предназначенную для идентификации. Другие строки, начинающиеся с символа ';', являются комментариями и игнорируются. За строкой описания следует код последовательности. При кодировании нуклеотидов буквами А, С, G и Т кодируют, соответственно, аденин, цитозин, гуанин и тимин. На рисунке 14 представлен пример тестового файла в формате FASTA.

```
; Austrostipa
>AUSG51910|Austrostipa exilis|matK|JF769104
TTTATTGCGATTCTTTCTCAACTACTATTCTGAATTGGAATACTCTTATTACTTCAATGAAATCGATTTTTATTTTGAAAA
AAGAAAATAAAAGACTATTTCAATTCCTATACAACCTCTTATGTATCAGAATATGAATTTTTCTTGTTGCTTCTTCGTA
CAATCTTCTTGGTTATCATTAACATCTTCTGGAACCTTTCTAGAACGAATCCACTTTTCTAGGAAGATGGAACATTTTGG
GATAATGTACCCAGGTTTTTTTCGGAAAACAGTATGGTTCTTTATGGATCCTCTTATGCATTATGTTTCGATATCAAGGAA
AGGCAATTCTTGCATCAAAAGGAACCTTTTTTTGAAGAAGAAATGGAAATGTTAC
>AUSG47510|Austrostipa geoffrey|matK|JF769107
ACTATTCGAATTGGAATACTCTTATTACTTCAATGAAATCGATTTTTCTTTGAAAAAAGAAAATAAAAGACTATTTCAA
TTCCTATACAACCTCTTATGTATCAGAATATGAATTTTTCTTGTTGCTTCTTCGTAAACAATCTTCTTGGTTATCATTAAC
ATCTTCTGGAACCTTTCTAGAACGAATCCACTTTTCTAGGAAGATGGAACATTTTGGGATAATGTACCCAGGTTTTTTTC
GGAAAACAGTATGGTTCTTTATGGATCCTCTTATGCATTATGTTTCGATATCAAGGAAAGGCA
```

Рисунок 14 – Пример файла в формате FASTA

Программа может использовать для вычисления очков за сопоставление нуклеотидов и аминокислот пользовательские матрицы замен. Их необходимо представить в таком же формате, как и на рисунке 15. Строки, начинающиеся с символа '#', являются комментариями и игнорируются, а при задании таблицы первые строка и столбец определяют символы сопоставления. Количество пробелов может быть любым.

3.2.2 Определение порядка выравниваний

Алгоритм кластеризации реализован в виде процедуры, принимающей на вход массив последовательностей и функцию оценки расстояния между ними — *dist*. Пользователь может выбрать, что использовать в качестве оценки *dist*: метод построения парного выравнивания *Align* из класса *PairwiseAlign* или подсчет количества одинаковых подпоследовательностей заданной длины *k*. Существует возможность использования своей соб-

#	Матрица замен аминокислот BLOSUM-62																							
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0	-4
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1	-4
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1	-4
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1	-4
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2	-4
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1	-4
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1	-4
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1	-4
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1	-4
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1	-4
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1	-4
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1	-4
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2	-4
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1	-4
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1	-4
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1	-4
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1	-4
*	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	1

Рисунок 15 – Пример файла с матрицей замен

ственной функции расстояния. Для этого ее необходимо определить как *std :: function < int (BioSeq*, BioSeq*) >*.

На каждом шаге алгоритма не происходит выделение новой памяти для перехода от старой таблицы размерности n к новой $n - 1$. Используется дополнительный массив, в котором хранится информация о том, какие строки и столбцы матрицы более неактивны и пропускаются при переборе. На рисунке 16 показан пример пересчета таблицы расстояний с указанием неактивных строк и обновляемых на каждом этапе элементов.

Такой подход позволяет экономить время выполнения программы за счет избавления от операций выделения новой таблице на каждом шаге алгоритма кластеризации. Кроме того, алгоритм экономно использует память компьютера.

3.2.3 Объединение профилей

При объединении профилей P_1 и P_2 происходит расчет оптимального выравнивания с заполнением матрицы размерности $len(P_1) \cdot len(P_2)$. Для то-

номер последовательности	1	2	3	4	5
1					
2	68				
3	60	67			
4	63	72	69		
5	70	74	75	71	

номер последовательности	1	(2, 4)	(3, 5)	4	5
1					
(2, 4)	65,5				
(3, 5)	65	70,75			
4					
5					

номер последовательности	1	2	(3, 5)	4	5
1					
2	68				
(3, 5)	65	70,5			
4	63	72	70		
5					

номер последовательности	1	((2, 4), (3, 5))	(3, 5)	4	5
1					
((2, 4), (3, 5))	65,25				
(3, 5)					
4					
5					

Рисунок 16 – Пересчет матрицы расстояний. Красным отмечены элементы для объединения, зеленым — обновленные значения, серым — неактивные ячейки.

го чтобы сократить количество выделений памяти, было решено сохранять заполненную таблицу после операции объединения. Таким образом, новая память будет выделена только в том случае, если текущая размерность матрицы, оставшейся с предыдущей итерации, не удовлетворяет размерности очередной пары профилей. На листинге 5 представлен алгоритм объединения.

Листинг 5 Алгоритм объединения профилей

```

 $F \leftarrow []$  ▷ Матрица оптимальных ходов
procedure ProfileMerge( $P_1, P_2$ )
▷ Проверка размерности матрицы
if  $size(F) < len(P_1)$  OR  $size(F) < len(P_2)$  then
     $new\_size \leftarrow \max(len(P_1), len(P_2))$ 
     $F \leftarrow [new\_size \times new\_size]$  ▷ Выделение новой памяти
end if

```

```

for  $i = 1 \dots \text{len}(P_1)$  do                                ▷ Цикл заполнения таблицы переходов
    for  $j = 1 \dots \text{len}(P_2)$  do
                                                ▷ Позиция текущей ячейки —  $i, j$ 
        Расчет оптимального перехода,
        перебор 25 возможных вариантов
    end for
end for
while ответ не построен do                                ▷ Цикл построения ответа
    Определить переход на  $i$ -ом шаге
    Изменить последовательности в профилях
    Перейти по таблице на следующую ячейку
end while
end procedure
 $P_1 \leftarrow$  добавить все последовательности из  $P_2$ 
return  $P_1$ 

```

3.3 Руководство пользователя

3.3.1 Опции компиляции

Ниже приведены команды сборки разработанного приложения для компилятора gcc [20] версии 4.8.4. Подразумевается, что в директории, где происходит компиляция, находятся только исходные коды описанной программы. При сборке необходимо указать стандарт языка c++11, а также для повышения производительности добавить опцию оптимизации -O3:

```
g++ -std=c++11 -O3 *.cpp -o multy
```

Для получения версии с печатью отладочной информации в процессе построения выравнивания (вывод матрицы оптимальных ходов, текущих результатов объединения профилей и таблицы расстояний на каждом шаге кластеризации) нужно воспользоваться следующей командой:

```
g++ -std=c++11 -DDEBUG *.cpp -o multy
```

Использование такой сборки предполагает поиск и устранение ошибок, возникающих в процессе выравнивания. В таком случае к команде компиляции добавляют опции -g и -O0, отвечающие за предоставление отладочных

символов в исполняемом файле и отсутствие оптимизации. После устранения всех ошибок и переходу к этапу тестирования можно воспользоваться сборкой, предоставляющей информацию о времени выполнения различных этапов алгоритма:

```
g++ -std=c++11 -DTIME -O3 *.cpp -o multy
```

Также для удобства использования был написан Makefile — набор инструкций для программы make [21], благодаря которому сборка всего решения производится одной командой: make, или make DEBUG — для получения версии с отладочной информацией.

Используемая в предыдущих командах компиляции опция -o определяет имя полученной программы. Для конкретизации обозначения разработанного приложения множественного выравнивания было выбрано название multy (от английского multiple — множественный).

3.3.2 Параметры запуска

Работа с программой представлена в двух вариантах: через консольное приложение или веб-интерфейс. В таблице 1 перечислены все возможные опции для запуска исполняемого файла.

Таблица 1: Опции программы

Сокращенные опции	Полные опции	Значение	Комментарий
<code>-h</code>	<code>--help</code>	без аргумента	вывод справочной информации
<code>-i</code>	<code>--input</code>	путь к файл с последовательностями	файл должен быть представлен в формате FASTA
<code>-n</code>	<code>--NT_subst</code>	путь к файлу с матрицей замен для нуклеотидов	по умолчанию: +4 за сопоставление одинаковых нуклеотидов и -5 в противном случае
<code>-a</code>	<code>--AA_subst</code>	путь к файлу с матрицей замен для аминокислот	по умолчанию используется матрица замен BLOSUM62 [22]

Сокращенные опции	Полные опции	Значение	Комментарий
$-g$	$--gap_open$	штраф за открытие разрыва	значение по умолчанию: -10
$-e$	$--gap_extension$	штраф за продолжение разрыва	значение по умолчанию: -3
$-f$	$--gap_frame$	штраф за разрыв рамки	значение по умолчанию: -15
$-s$	$--stop_cost$	штраф за преждевременное появление стоп-кодона	значение по умолчанию: -50
$-d$	$--dimension$	начальная размерность матрицы для объединения профилей	значение по умолчанию: 1024×1024
$-k$	$--k-mers$	необязательный числовой аргумент k	использовать в качестве функции <i>dist</i> алгоритм подсчета общих подстрок длины k ; значение по умолчанию для k : 10
$-p$	$--pairwise$	без аргумента	использовать в качестве функции <i>dist</i> алгоритм построения парного выравнивания

При запуске приложения без указания входного файла или с использованием неверных опций будет выдано соответствующее сообщение об ошибке. В случае корректного задания входных параметров произойдет чтение и разбор указанного набора последовательностей, после чего будет напечатана вся информация по заданной задаче выравнивания: количество полученных последовательностей, матрицы замен и параметры штрафов (листинг 6).

Листинг 6 Примеры запуска программы

```
$ ./multy -s -40 -f 20
nothing to align

$ ./multy -i ENAM_genes.fasta --AA_substitution BLOSUM60
./multy: unrecognized option '--AA_substitution'

$ ./multy -i ENAM_genes.fasta
Reading sequences ...
7 sequences were obtained
Input parameters:
NT substitution matrix  NUC-45
```

AA substitution matrix	BLOSUM62
Gap open cost	−10
Gap extension cost	−3
Gap frame cost	−15
Stop codon cost	−50

```
$ ./multy -i ENAM_genes.fasta -s -40 --AA_subst BLOSUM60
Reading sequences ...
7 sequences were obtained
Input parameters:
NT substitution matrix  NUC-45
AA substitution matrix  BLOSUM60
Gap open cost           -10
Gap extension cost      -3
Gap frame cost          -15
Stop codon cost         -40
```

На рисунке 17 изображен графический интерфейс для работы с программой. Форму можно условно разделить на две части: поле ввода данных и область задания параметров выравнивания.

The screenshot shows the web interface for the 'multy' program. At the top, the title is 'multy: множественное выравнивание кодирующих последовательностей'. Below the title are three links: 'главная', 'документация', and 'использование'. The interface is divided into two main sections. On the left, under the heading 'Введите последовательности (FASTA формат)', there is a large text input area for pasting sequences. Below this input area is a button labeled 'Построить выравнивание'. On the right, under the heading 'Входные параметры', there are several settings. Two input fields are shown: 'Штраф за начало разрыва' with a value of '-10' and 'Штраф за продолжение разрыва' with a value of '-3'. Below these are three radio button options for the algorithm: 'k-mers', 'парные выравнивания', and 'в зависимости от объема данных' (which is currently selected). The text 'Алгоритм вычисления расстояния для UPGMA' is positioned above the radio buttons.

Рисунок 17 – Графический интерфейс для построения выравнивания

После ввода последовательностей и указания необходимых параметров необходимо нажать на кнопку «построить выравнивание». По окончании работы будет загружена страница с результатами выравнивания (рисунок 18).

multy: множественное выравнивание кодирующих последовательностей

ГЛАВНАЯ ДОКУМЕНТАЦИЯ ИСПОЛЬЗОВАНИЕ

```
-----TTT--ATTGCGATTCTTTCTCAACTACTATTGGAATTGGAATCTTATTACTTCAATGAAATCGATTTTCTTTTGAAAAAGAAAAATAA
>AUSG47510|Austrostipa geoffrey|matK|JF769107
-----AAC---TATTGGAATTGGAATCTTATTACTTCAATGAAATCGATTTTCTTTTGAAAAAGAAAAATAA
>AUSG56010|Austrostipa oligostachya|matK|JF769120
-----TTT--ATTGCGATTCTTTCTCAACTACTATTGGAATTGGAATCTTATTACTTCAATGAAATCGATTTTCTTTTGAAAAAGAAAAATAA
>AUSG56710|Austrostipa puberula|matK|JF769122
----CCATCA--T---TTTTATTGCGATTCTTTCTCAACTACTATTGGAATTGGAATCTTATTACTTCAATGAAATCGATTTTCTTTTGAAAAAGAAAAATAA
>AUSG48910|Austrostipa aristiglumis|matK|JF769096
GTTCCATCATTTGCATTT-TATTGCGATTCTTTCTCAACTACTATTGGAATTGGAATCTTATTACTTCAATGAAATCGATTTTCTTTTGAAAAAGAAAAATAA

>AUSG51910|Austrostipa exilis|matK|JF769104
-----F!LRFFLNYYSNWNTLITSMKSIFFLLKKENKRLFQFLYNSYVSEYEFFLLLRKQSSWLSLTSSGTFLEIRHFSRKMEHFGIMYPGFFRKTWVFFMDPLMHYVI
>AUSG47510|Austrostipa geoffrey|matK|JF769107
-----N-YSNWNLTITSMKSIFFLLKKENKRLFQFLYNSYVSEYEFFLLLRKQSSWLSLTSSGTFLEIRHFSRKMEHFGIMYPGFFRKTWVFFMDPLMHYVI
>AUSG56010|Austrostipa oligostachya|matK|JF769120
-----F!LRFFLNYYSNWNTLITSMKSIFFLLKKENKRLFQFLYNSYVSEYEFFLLLRKQSSWLSLTSSGTFLEIRHFSRKMEHFGIMYPGFFRKTWVFFMDPLMHYVI
>AUSG56710|Austrostipa puberula|matK|JF769122
-PS!-FLLRFFLNYYSNWNTLITSMKSIFFLLKKENKRLFQFLYNSYVSEYEFFLLLRKQSSWLSLTSSGTFLEIRHFSRKMEHFGIMYPGFFRKTWVFFMDPLMHYVI
>AUSG48910|Austrostipa aristiglumis|matK|JF769096
VPSFAF!LRFFLNYYSNWNTLITSMKSIFFLLKKENKRLFQFLYNSYVSEYEFFLLLRKQSSWLSLTSSGTFLEIRHFSRKMEHFGIMYPGFFRKTWVFFMDPLMHYVI
```

Рисунок 18 – Форма с результатами выравнивания

3.3.3 Использование собранных библиотек

В данном разделе описана работа с собранными статическими библиотеками построения парного (*libpairwise.a*) и множественного (*libmulty.a*) выравниваний. В таблице 2 представлена информация о входящих в состав библиотек структур данных и функциях.

Таблица 2: Содержание статических библиотек

Библиотека	Содержание
libpairwise.a	структура BioSeq
	класс PairwiseAlign
	функция чтения файла в формате FASTA
libmulty.a	структура BioSeq
	структура Profile
	функция чтения файла в формате FASTA
	функция кластеризации по алгоритму UPGMA

В проект необходимо подключить заголовочные файлы с используемыми структурами и функциями (листинг 7). Функция *ReadFastaFile* осуществляет чтение последовательностей из указанного файла *input_fasta* и сохраняет их в массив *buf*. В случае возникновения ошибки она вернет значение 1, иначе — 0. Алгоритм кластеризации реализован через функцию *UPGMA*, принимающей на вход набор последовательностей *sequences*, функцию оценки расстояния *dist* и параметры выравнивания для объединения кластеров (названия параметров соответствуют именам коротких опций программы).

Вызов данной функции приведет к изменению данных в массиве *sequences*, он будет содержать результаты выравнивания.

Листинг 7 Прототипы библиотечных функций

```
int ReadFastaFile(char* input_fasta ,
                  std::vector<BioSeq*>& buf);

// distance function type
typedef std::function<int (BioSeq* seq1 ,
                          BioSeq* seq2)> func;

// UPGMA function change input sequences!
void UPGMA(std::vector<BioSeq*>& sequences , func dist ,
           // input parameters
           int g, int e, int f, int s, const int* n,
           const int* a);
```

При сборке приложения с импортированными методами и структурами построения выравниваний необходимо указать компилятору используемую библиотеку. Ниже представлен пример компиляции новой программы из исходного файла *new_program.c++* с библиотекой *libmulty.a*:

```
g++ -std=c++11 -L./ -lmulty new_program.c++
```

4 Тестирование

Для всесторонней оценки работы программы результат выравнивания и производительность сравнивали с единственным существующим ее аналогом — MACSE (пункт 1.3.3), который является признанным программным решением.

4.1 Оценка качества выравнивания

В качестве тестовых задач для оценки получаемых выравниваний использовали наборы последовательностей, которые применяли авторы MACSE в работе [17]. Для каждой тестовой задачи из набора строили множественные выравнивания с помощью программ MACSE и multy. Полученные результаты попарно сравнивались функцией оценки (листинг 8). Идея метода заключена в сопоставлении выравненных блоков последовательностей сравниваемых результатов и вычислении среднего арифметического долей совпадений в каждом из выравниваний:

$$\frac{\frac{count}{len(P_1)} + \frac{count}{len(P_2)}}{2}$$

где *count* — число совпадений, а *len(P_i)* — длина *i*-го выравнивания. Совпадением считается сопоставление одинаковых нуклеотидов или аминокислот, лишние разрывы пропускаются до следующего символа. Алгоритм оценки реализован на языке программирования Python 2.7.8 (листинг 8).

Листинг 8 Реализация алгоритма сравнения двух выравниваний

```
score = 0.
while index1 < len(seq1) and index2 < len(seq2):
    if (seq1[index1] == seq2[index2]):
        score += 1
        index1 += 1
        index2 += 1
    else:
        if (seq1[index1] == '-'):
            index1 += 1
        else:
            if (seq2[index2] == '-'):

```

```

        index2 += 1
    else :
        index1 += 1
        index2 += 1
total += (score / len(seq1) + score / len(seq2)) / 2.

```

Здесь *seq1* и *seq2* — сравниваемые последовательности, а *total* — оценка схожести выравниваний. При сравнении множественных выравниваний вычисляется среднее арифметическое полученных парных оценок.

В качестве исходных данных для тестирования был взят проверенный тестовый набор последовательностей MACSE. В связи с тем, что решение MACSE строит выравнивание с использованием дополнительных функций обновления и улучшения результатов при выравнивании псевдогенов и не имеет возможности их отключения, некоторые тесты были пропущены.

Обеим программам на вход подавались одинаковые параметры выравнивания. Согласно нашей функции оценки, мы получили среднее значение совпадений 90,5% на нуклеотидном и 92,1% на аминокислотном уровне. Наличие различий в выдаваемых выравниваниях естественно, так как алгоритмы имеют общую идею, но совершенно разную реализацию. В разработанном решении используются две матрицы замен: для нуклеотидного и аминокислотного уровней, в отличие от решения MACSE, ограничивающегося только оценкой уровня аминокислот. Более высокое качество выравнивания на аминокислотном уровне доказывает правильность работы созданного алгоритма. Полученные результаты являются удовлетворительными.

На рисунках 19 и 20 показаны построенные выравнивания для одного из блоков тестов MACSE. Представлены только различающиеся участки полученных выравниваний. Общая длина на нуклеотидном уровне — 426 остатков.

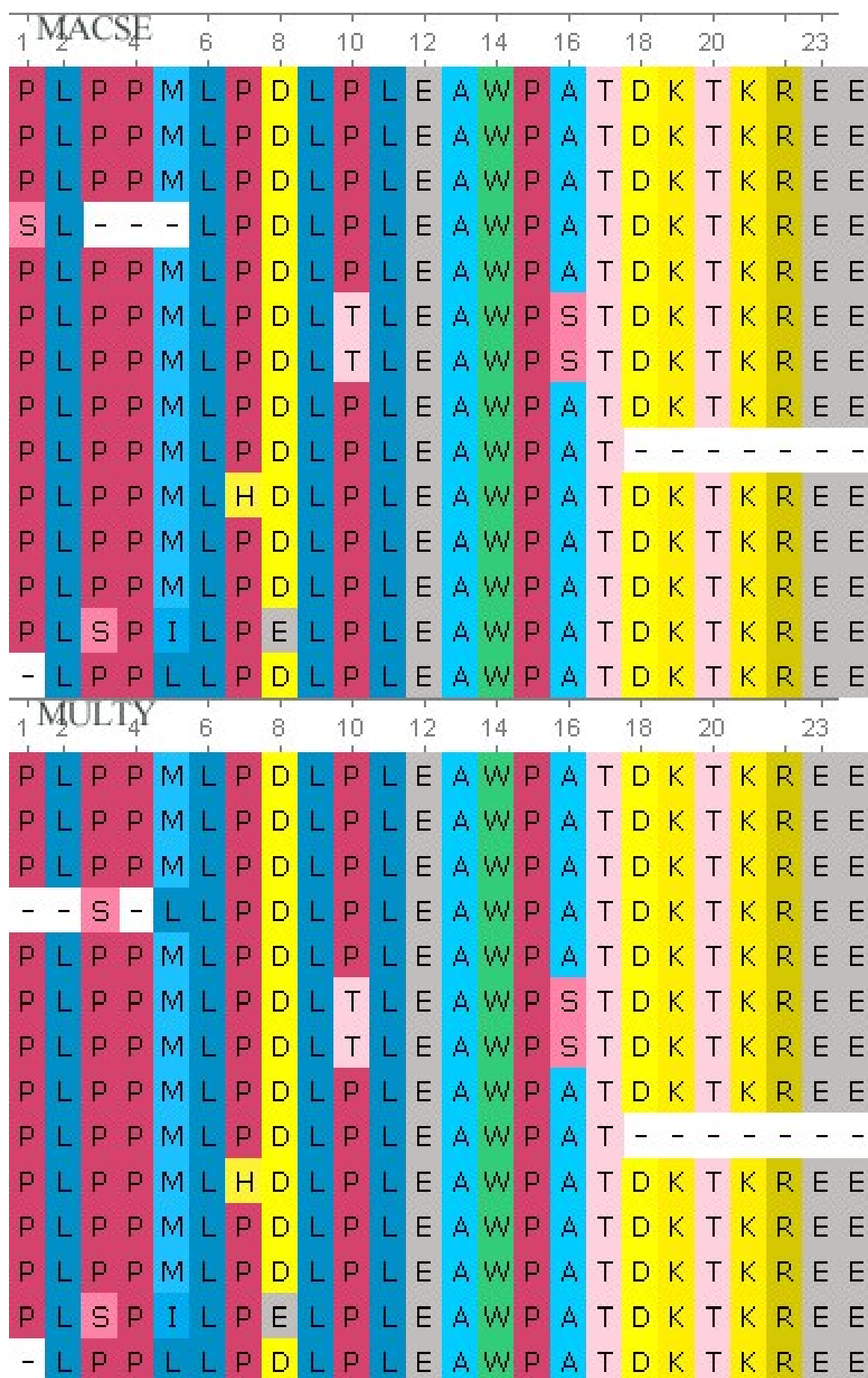


Рисунок 19 – Сравнение множественных выравниваний последовательностей аминокислот

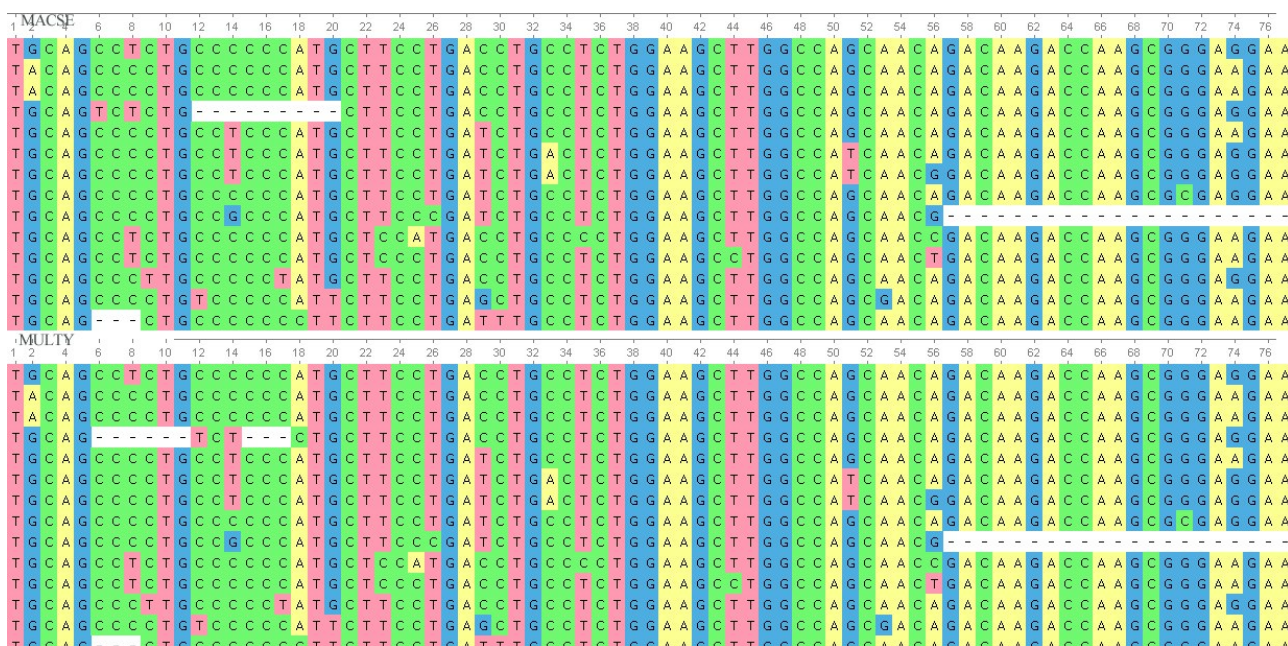


Рисунок 20 – Сравнение множественных выравниваний последовательностей нуклеотидов

4.2 Оценка производительности

Производительность разработанного алгоритма против существующей программы парного выравнивания MACSE оценивалась на группах синтетических тестов — наборы созданных случайных последовательностей нуклеотидов заданной длины (на листинге 9 представлен код программы генерации тестовых данных). Тестирование происходило на машине с двумя шестиядерными процессорами Intel Xeon E5645 под управлением ОС Linux Debian 7 (wheezy) и объемом оперативной памяти 24 ГБ. Результаты тестирования представлены в таблице и на гистограмме (рисунок 22).

Листинг 9 Реализованные на Python 2.7.8 функции создания набора тестовых данных

```
import random, string

def CreateSeq(length = 1000):
    return ''.join(random.choice("ACGT") \
                    for _ in range(length))

def CreateData(seq_count, seq_length):
    data = []
```

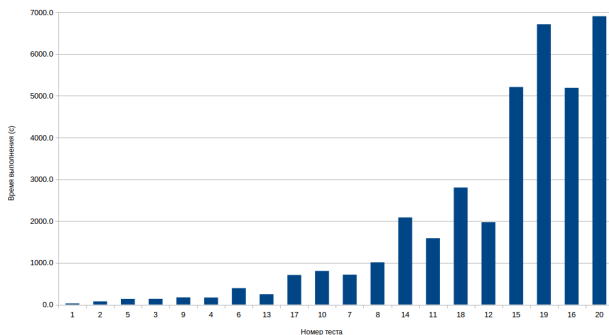
```

for i in range(seq_count):
    data.append(">seq"+str(i))
    data.append(CreateSeq(seq_length))
return data

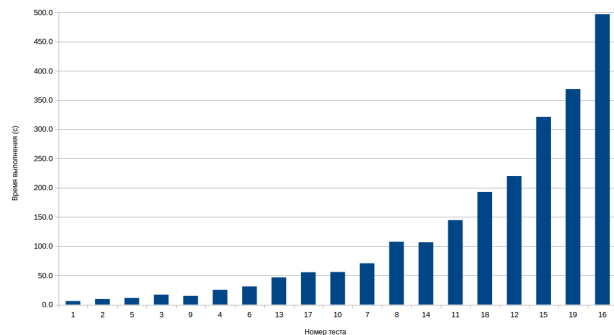
```

Из графиков и таблицы хорошо видно серьезное замедление времени выполнения на больших тестах для обоих алгоритмов (рисунок 21). При тестировании было рассмотрено несколько сборок разработанной программы: используя компилятор gcc версии 4.8.4 с флагом -O3 и компилятор intel icpc версии 15.0.3 с различными флагами оптимизации. Одна из выбранных опций -fast, включающая в себя следующие ключи icpc:

```
-xHOST -O3 -ipo -no-prec-div -static -fp-model fast=2
```



а) MACSE



б) multy

Рисунок 21 – График роста времени выполнения при увеличении объема входных данных

Также компилятор intel предоставляет возможность скомпилировать пробную версию программы, запустить ее на заданном наборе тестов и по результатам выполнения произвести повторную сборку, что потенциально может дать прирост скорости. Для этого необходимо воспользоваться опцией -prof-gen:

```
-O3 -no-prec-div -fp-model fast=2 -xHost -prof-gen
```

После компиляции производится запуск программы на тестовом наборе данных для генерации профилей производительности *.dyn и *.dpi, а затем выполняется повторная сборка. Аналогичным образом используется ключ ав-

томатической параллелизации -parallel:

```
-O3 -no-prec-div -fp-model fast=2 -xHost -prof-gen -parallel
```

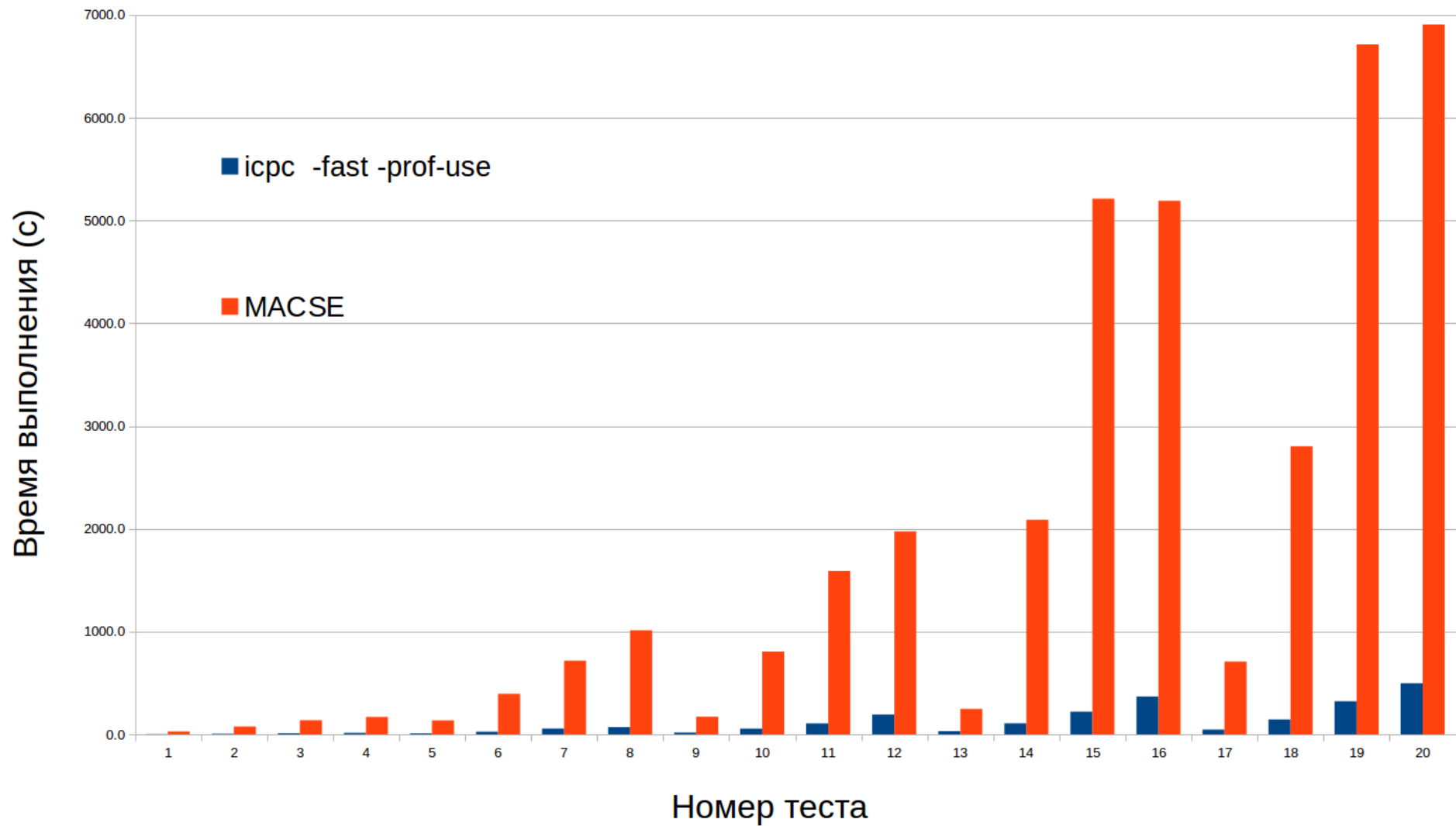


Рисунок 22 – Время выполнения тестов разработанного алгоритма и MACSE

Таблица 3: Результаты тестирования

Номер теста	Длина поledo- вательностей	Количество последователь- ностей	Время выполнения (с)						
			Опции компиляции						MACSE
			icpc -O0	icpc -fast	icpc -fast -prof-use	icpc -fast -parallel	icpc -fast -prof-use -parallel	gcc -O3	
1	500	5	11,9	2,4	2,3	2,3	2,2	2,8	27,3
2	500	10	40,2	6,7	6,1	7,2	6,1	7,7	76,3
3	500	15	76,0	12,5	11,4	12,0	11,1	14,0	137,8
4	500	20	115,0	18,7	15,0	16,0	15,7	18,5	169,3
5	1000	5	48,8	9,1	9,6	8,8	8,1	10,8	136,3
6	1000	10	165,3	27,4	25,2	28,5	25,3	34,7	394,3
7	1000	15	396,3	62,4	55,8	62,9	60,9	69,9	716,9
8	1000	20	556,7	75,1	70,7	86,7	74,8	87,4	1012,1
9	1500	5	100,5	18,1	17,0	18,1	16,6	22,0	171,6
10	1500	10	384,5	60,4	55,4	65,2	57,0	78,1	805,8
11	1500	15	830,0	117,5	106,7	115,6	106,0	136,0	1589,4
12	1500	20	1551,4	205,2	192,7	203,8	196,3	252,4	1974,1
13	2000	5	184,9	33,2	31,1	40,8	29,3	39,9	248,2
14	2000	10	669,4	106,7	107,6	103,8	101,8	124,4	2087,5
15	2000	15	1540,5	219,7	219,9	274,0	203,2	257,5	5212,5
16	2000	20	2909,8	397,5	368,8	401,8	375,6	529,8	5191,9
17	2500	5	290,3	53,4	46,5	50,9	46,2	62,1	708,8
18	2500	10	1006,8	156,9	144,6	166,8	143,6	187,9	2803,2
19	2500	15	2417,3	344,1	321,2	352,0	334,8	408,6	6714,3
20	2500	20	3944,9	596,8	496,9	572,0	511,3	665,7	6907,3

ВЫВОДЫ

- Предложен и реализован новый метод парного выравнивания последовательностей ДНК с учетом сдвигов рамки считывания.
- На основе этого метода создана программа *multy* для решения задачи множественного выравнивания, которая позволяет получать ответ как для самих нуклеотидных последовательностей, так и для продуктов их трансляции.
- На демонстрационном наборе задач проведено сравнение программы *multy* с единственным существующим аналогом - программой MACSE. Показано, что результаты множественного выравнивания хорошо согласуются с эталонными, и что программа *multy* превосходит MACSE по производительности в 20,9 раз на наборе из 15 последовательностей длины 2500 нуклеотидов.

ЗАКЛЮЧЕНИЕ

В настоящей работе предложен и реализован новый метод построения множественных выравниваний кодирующих последовательностей ДНК с учетом сдвигов рамки считывания. Отличительной особенностью представленного решения является то, что, в отличие от классического алгоритма Нидлмана-Вунша, при построении парных выравниваний используются не только сами нуклеотидные последовательности, но также их возможные продукты трансляции. Такой подход позволяет не просто решать математическую задачу выравнивания строк алфавита генетического кода, но и учитывать ее биологический контекст. Представленный в работе метод позволяет снизить влияние ошибок при секвенировании нуклеотидных последовательностей, «преждевременных» стоп-кодонов, вызванных ошибками или мутациями, и других артефактов.

Разработанный алгоритм построения множественных выравниваний реализован в виде кросс-платформенной компьютерной программы на языке C++, а также в виде набора статических библиотек, которые можно включать в состав сторонних программ. Важной особенностью представленной программы является ее модульность: функции каждого из этапов построения выравнивания реализованы независимо и могут быть заменены на другие. В частности, в текущей версии для построения матрицы сходства для набора последовательностей можно использовать либо метод попарных выравниваний, либо метод подсчета общих подстрок заданной длины. В плане дальнейшей разработки программы стоит добавление альтернатив алгоритму кластеризации UPGMA, а также ряда методов, позволяющих улучшить промежуточные результаты выравнивания.

Список литературы

- [1] Миронов Андрей Александрович. Лекция "Введение в биоинформатику". [Электронный ресурс] // URL: http://mipt.ru/dbmp/student/files/bioinformatics/public_lecture/ (дата обращения 02.03.2015).
- [2] Mount David W. Bioinformatics: sequence and genome analysis. New York: 2004.
- [3] Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР. 163.4:845-848. 1965.
- [4] Humberto Carrillo, David Lipman "The Multiple Sequence Alignment Problem in Biology" on Applied Mathematics Vol. 48, No. 5. (Oct., 1988).
- [5] А.В. Бутвиловский Е.В. Барковский В.Э. Бутвиловский. Выравнивание аминокислотных и нуклеотидных последовательностей. [Электронный ресурс] // Белорусский государственный медицинский университет. URL: <http://www.bsmu.by/medicaljournal/bca1983ad3b6898219bca6eb399ae488/> (дата обращения 17.03.2015).
- [6] Needleman S. B., Wunsch C. D. "A general method applicable to the search for similarities in the amino acid sequence of two proteins" on Journal of Molecular Biology Vol. 48, no. 3. 1970.
- [7] Smith T. F., Waterman M. S. "Identification of common molecular subsequences" on Journal of Molecular Biology Vol. 147, no. 1. 1981.
- [8] Hirschberg D. S. A linear space algorithm for computing maximal common subsequences. New York: 1975. P. 341–343.
- [9] Параллельный алгоритм глобального выравнивания с оптимальным использованием памяти [Электронный ресурс] // <http://www.science-education.ru/107-8139> (дата обращения 18.03.2015).
- [10] Ярыгин В. Н. Биология / В. Н. Ярыгин, В. И. Васильева, И. Н. Волков [и др.] // Биология. 2010. Т. 8. с. 12.

- [11] Wernersson R, Pedersen AG (2003) RevTrans: Multiple alignment of coding DNA from aligned amino acid sequences. *Nucleic Acids Res* 31: 3537–3539.
- [12] Bininda-Emonds OR (2005) transAlign: using amino acids to facilitate the multiple alignment of protein-coding DNA sequences. *BMC Bioinformatics* 6: 156.
- [13] Abascal F, Zardoya R, Telford MJ (2010) TranslatorX: multiple alignment of nucleotide sequences guided by amino acid translations. *Nucleic Acids Res* 38: W7–13.
- [14] Suyama M, Torrents D, Bork P (2006) PAL2NAL: robust conversion of protein sequence alignments into the corresponding codon alignments. *Nucleic Acids Res* 34: W609–612.
- [15] Hein J (1994) An algorithm combining DNA and protein alignment. *J Theor Biol* 167: 169–174.
- [16] Arvestad L (1997) Aligning coding DNA in the presence of frame-shift errors. pp. 180–190.
- [17] MACSE: Multiple Alignment of Coding SEquences Accounting for Frameshifts and Stop Codons [Электронный ресурс] // URL: <http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0022594> (дата обращения 23.03.2015).
- [18] Legendre Pierre, Legendre Louis. Numerical ecology: second English edition // Developments in environmental modelling. 1998. Т. 20.
- [19] What is FASTA format? [Электронный ресурс] // URL: <http://zhanglab.ccmb.med.umich.edu/FASTA/> (дата обращения 04.04.2015).
- [20] The GNU Compiler Collection [Электронный ресурс] // GNU Operating System. URL: <http://gcc.gnu.org/> (дата обращения 07.05.2015).
- [21] GNU Make [Электронный ресурс] // GNU Operating System. URL: <http://www.gnu.org/software/make/> (дата обращения 07.05.2015).

- [22] BLOSUM Clustered Scoring Matrix [Электронный ресурс] // URL: <http://www.ncbi.nlm.nih.gov/Class/FieldGuide/BLOSUM62> (дата обращения 21.05.2015).