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

Современная биоинформатика — это молодая, бурно развивающаяся наука, возникшая в 1976-1978 годах и окончательно оформившаяся в 1980 году со специальным выпуском журнала «Nucleic Acid Research» (NAR) [1]. 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.

Основу биоинформатики составляют сравнения. Одна из ключевых задач — поиск сходства последовательностей. Ее решение позволяет понять функциональное назначение частей геномов, оценить эволюционное расстояние между ними. Кроме этого, различия в генотипах могут объяснить различия в фенотипах.

Для того чтобы определить, насколько две последовательности «похожи», используют алгоритмы выравнивания. Они основаны на размещении исходных последовательностей мономеров ДНК, РНК или белков друг под другом таким образом, чтобы было легко увидеть их сходные участки [2]. Качество выравнивания оценивают, назначая штрафы за несовпадение букв и за наличие пробелов (когда приходится раздвигать 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 сравнении ищется такой вариант выравнивания, чтобы итоговый счет был максимален. В такой постановке задача называется поиском «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 парного 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 дуплицированные 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-кодонов 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 поставлены 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 последовательностей 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.

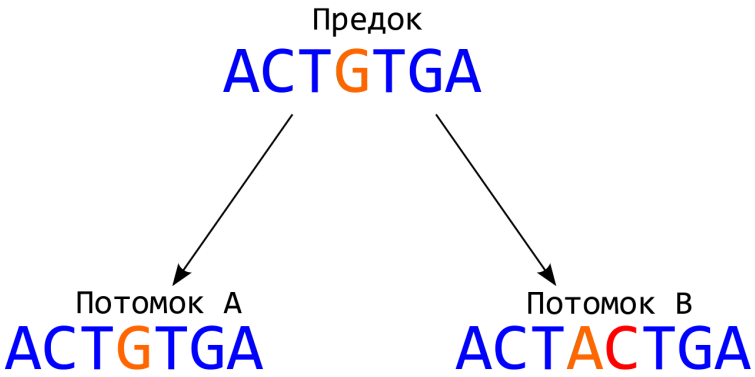


Рисунок 1 – Пример мутации

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 длинны 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.

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

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 $(i, j) \to (i-1, j-1)$ означает, что i-му символу в первой строке соответствует j-й символ во второй строке. Переход вида $(i, j) \to (i-1, j)$ означает, что i-му символу первой строки ничего не соответствует, а переход $(i, j) \to (i, j-1)$ — 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.

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

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 осуществля-

ется 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 $m - k^*$.

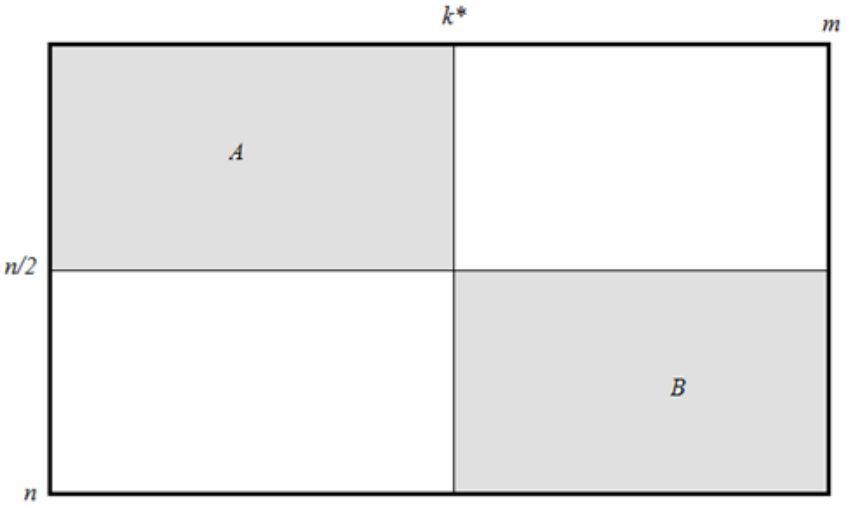


Рисунок 3 – Разделение задачи выравнивания на две подзадачи

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

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.

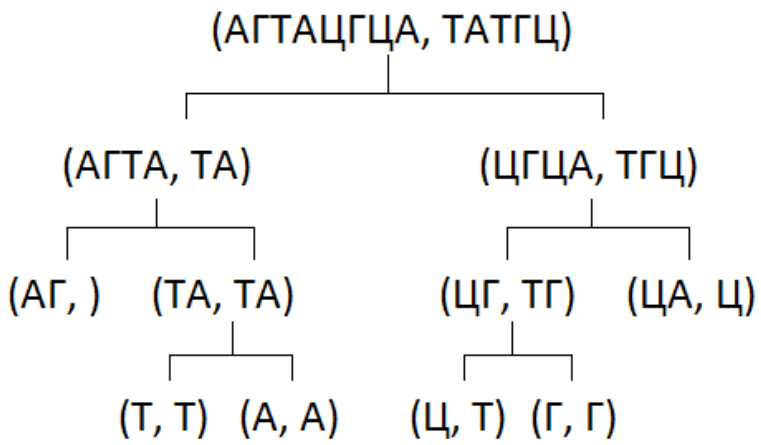


Рисунок 4 – Дерево подзадач для алгоритма Хиршберга

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.

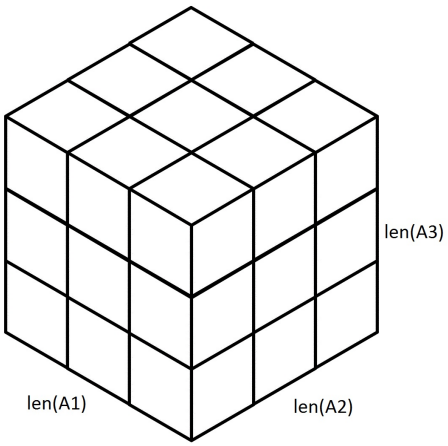


Рисунок 5 – Матрица F для выравнивания трех последовательностей

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

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 $A_1, \ldots, A_n$ — выравниваемые последовательности, то алгоритм можно условно записать формулой 4.

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

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 третей, 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.

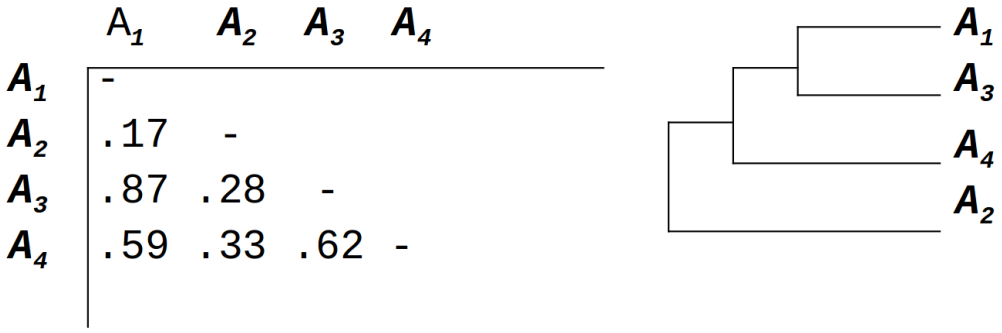


Рисунок 6 – Построение дерева-подсказки для алгоритма Clustal

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 программа для множественного выравнивания кодирующих последовательностей с учетом существующих рамок считывания и стоп-кодонов [17]. Кроме задачи выравнивания, она может быть применена 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 $O(len(S_1) \cdot len(S_2))$. 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:

- $len(S_k)$ — 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
- $cost(\mathcal{A}(S_i, S_j))$ — численная характеристика полученного выравнивания, вычисляемая по рекурсивной формуле
- $cost('-')$ — штраф (число) за разрыв последовательности
- $cost('!')$ — штраф за разрыв рамки считывания
- $cost('*')$ — 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 $cost(\mathcal{A}(S_1, S_2))$ для классического алгоритма Нидлмана-Вунша (5).

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

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-кодоны 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.

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

где $AA_1 = \pi(S_1[3i-2:3i])$ и $AA_2 = \pi(S_2[3j-2:3j])$

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

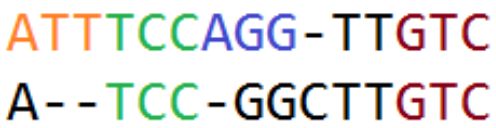
Таким образом, за разрыв длины l будет начислен штраф $l \cdot cost('-')$ при линейном или $cost(gap_open) + l \cdot cost(gap_extension)$ — при аффинном подходе.

2.2 Построение парного выравнивания

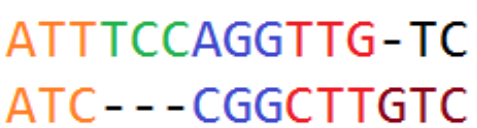
В отличие от алгоритма Нидлмана-Вунша, учитывающего в своей рекурсивной формуле лишь три варианта перехода (инсерция, делеция или совпадение нуклеотидов), разработанное решение рассматривает дополнительные возможности построения выравнивания с учетом образующихся аминокислот и сдвигов рамки считывания. 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.



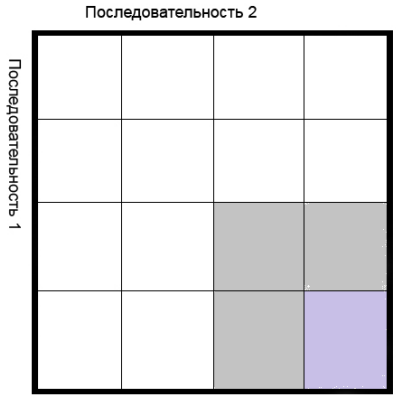
а) Выравнивание по алгоритму Нидлмана-Вунша



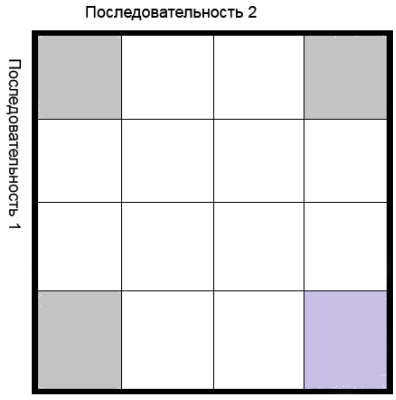
б) Выравнивание по разработанному алгоритму

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.

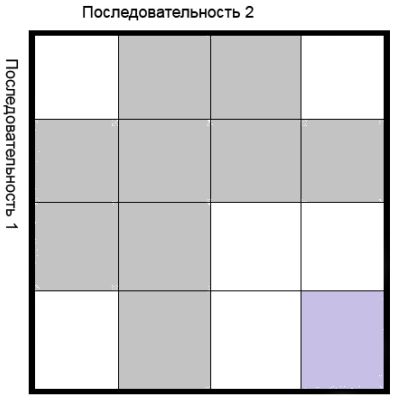
Созданный алгоритм двухуровневого выравнивания на каждом этапе выбора рассматривает все возможные варианты сопоставления от нуля до трех нуклеотидов из каждой последовательности. Таким образом, общее число возможных переходов: $\sum_{i=1}^{3} 2 \cdot 2^i - 1 = 25$. Для наглядности эти варианты представлены 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, потомучто до рассматриваемых клеток перехода существует несколько возможных 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** ▷ 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
  $AA_1 \leftarrow \pi(S_1[i-2:i])$ ▷ Трансляция триплетов
  $AA_2 \leftarrow \pi(S_2[j-2:j])$
  $stopS_1 \leftarrow 0$
  $stopS_2 \leftarrow 0$ ▷ 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
  ▷ Перебор вариантов: сопоставление аминокислот

$$score \leftarrow max \begin{cases} CalcAlign(S_1, S_2, i-3, j-3) + \sigma(AA_1, AA_2) \\ CalcAlign(S_1, S_2, i-3, j) + stopS_1 + cost('-') \\ CalcAlign(S_1, S_2, i, j-3) + stopS_2 + cost('-') \end{cases}$$

  ▷ Перебор вариантов: сопоставление неполных триплетов

$$score \leftarrow max \begin{cases} CalcAlign(S_1, S_2, i-2, j-2) + \sigma(S_1[i-1], S_2[j-1]) \\ \qquad + \sigma(S_1[i], S_2[j]) + 2 \cdot cost('!') \\ CalcAlign(S_1, S_2, i-2, j-1) + \sigma(S_1[i-1], S_2[j]) + 2 \cdot cost('!') \\ CalcAlign(S_1, S_2, i-2, j-1) + \sigma(S_1[i], S_2[j]) + 2 \cdot cost('!') \\ CalcAlign(S_1, S_2, i-2, j) + cost('!') + cost('-') \\ CalcAlign(S_1, S_2, i-1, j-2) + \sigma(S_1[i], S_2[j-1]) + 2 \cdot cost('!') \\ CalcAlign(S_1, S_2, i-1, j-2) + \sigma(S_1[i], S_2[j]) + 2 \cdot cost('!') \\ CalcAlign(S_1, S_2, i, j-2) + cost('!') + cost('-') \\ CalcAlign(S_1, S_2, i-3, j-2) + \sigma(S_1[i-2], S_2[j-1]) \\ \qquad + \sigma(S_1[i-1], S_2[j]) + stopS_1 + cost('!') \\ CalcAlign(S_1, S_2, i-3, j-2) + \sigma(S_1[i-1], S_2[j-1]) \\ \qquad + \sigma(S_1[i], S_2[j]) + stopS_1 + cost('!') \\ CalcAlign(S_1, S_2, i-3, j-2) + \sigma(S_1[i-2], S_2[j-1]) \\ \qquad + \sigma(S_1[i], S_2[j]) + stopS_1 + cost('!') \\ CalcAlign(S_1, S_2, i-3, j-1) + \sigma(S_1[i-2], S_2[j]) + stopS_1 + cost('!') \\ CalcAlign(S_1, S_2, i-3, j-1) + \sigma(S_1[i-1], S_2[j]) + stopS_1 + cost('!') \\ score \end{cases}$$

$$
score \leftarrow max \begin{cases}
CalcAlign(S_1, S_2, i-3, j-1) + \sigma(S_1[i], S_2[j]) + stopS_1 + cost('!') \\
CalcAlign(S_1, S_2, i-2, j-3) + \sigma(S_1[i-1], S_2[j-2]) \\
\qquad + \sigma(S_1[i], S_2[j-1]) + stopS_2 + cost('!') \\
CalcAlign(S_1, S_2, i-2, j-3) + \sigma(S_1[i-1], S_2[j-1]) \\
\qquad + \sigma(S_1[i], S_2[j]) + stopS_2 + cost('!') \\
CalcAlign(S_1, S_2, i-2, j-3) + \sigma(S_1[i-1], S_2[j-2]) \\
\qquad + \sigma(S_1[i], S_2[j]) + stopS_2 + cost('!') \\
CalcAlign(S_1, S_2, i-1, j-3) + \sigma(S_1[i], S_2[j-2]) + stopS_2 + cost('!') \\
CalcAlign(S_1, S_2, i-1, j-3) + \sigma(S_1[i], S_2[j-1]) + stopS_2 + cost('!') \\
CalcAlign(S_1, S_2, i-1, j-3) + \sigma(S_1[i], S_2[j]) + stopS_2 + cost('!') \\
score
\end{cases}
$$

▷ Перебор вариантов: сопоставление нуклеотидов

$$
score \leftarrow max \begin{cases}
CalcAlign(S_1, S_2, i-1, j-1) + \sigma(S_1[i], S_2[j]) + 2 \cdot cost('!') \\
CalcAlign(S_1, S_2, i-1, j) + cost('!') + cost('-') \\
CalcAlign(S_1, S_2, i, j-1) + cost('!') + cost('-') \\
score
\end{cases}
$$

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).

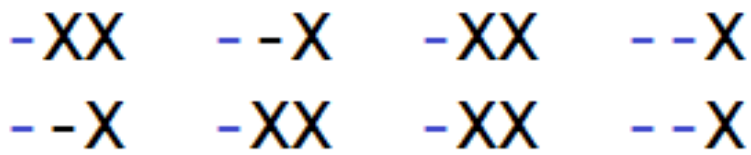


Рисунок 9 – Сопоставление неполных триплетов. Цветом указаны разрывы, добавленные для сохранения текущей рамки считывания.

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: $dist(S_i, S_j) \to \mathbb{R}$. 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 похожести 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

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

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.

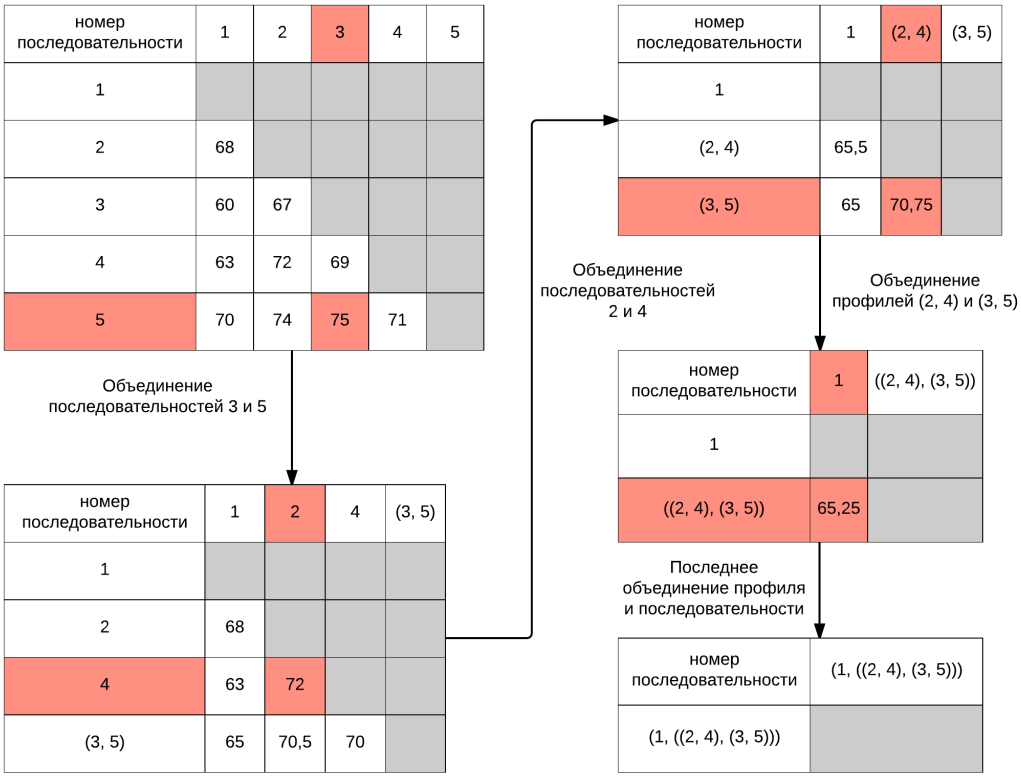


Рисунок 10 – Изменение матрицы расстояний в процессе кластеризации. Красным цветом обозначены объединяемые кластеры.

Необходимо отметить, что процесс кластеризации необязательно продолжать 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

Определим новое понятие «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 $i = 1 \dots len(P)$, где $len(P)$ — 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.

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

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 запущен на всех основных операционных системах: Windows, Linux, Mac OS. Кроме основного приложения, был создан веб-интерфейс, через который также можно получить выравнивание. Так как задача требует существенных вычислительных ресурсов, практично запускать программу на мощной вычислительной платформе, а осуществлять взаимодействие с ней через веб-интерфейс. Кроме этого, для возможности внедрения отдельных компонент программы в другие проекты, были собраны статические библиотеки построения парного и множественного выравниваний.

3.1 Структуры данных

Ниже описанны используемые в программе структуры для хранения и обработки кодирующих последовательностей. 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 легко подключенны 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 у нее 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 <<. В качестве результат работы программы желательно получить выравнивание не только на нуклеотидном, но также и на аминокислотном уровне. Для наглядного разделения этих двух опций, в структуру были добавлены методы $void\ PrintNT(std :: ostream\&\ out)$ и $void\ PrintAA(std :: ostream\&\ out)$, которые выводят результат выравнивания и его трансляцию соответственно 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);
  // translater
  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 через класс *PairwiseAlign* (листинг 3). Таким образом, в одном объекте собираются и данные, и методы их обработки, что упрощает импорт алгоритма парного выравнивания в другой проект.

Класс не содержит полей с исходными последовательностями, в нем хранятся только структуры для построения ответа и параметры выравнивания: штрафы за разрывы, смещение рамки, появление преждевременного стоп-кодона, матрицы замен нуклеотидов и аминокислот. Данные для выравнивания передаются через метод *Align*(*const BioSeq* seq*1, *const BioSeq* seq*2), реализующий алгоритм из пункта 2.2. Однако, для увеличения производительности вместо рекурсии происходит заполнение таблицы оптимальных ходов, как в классическом алгоритме Нидлмана-Вунша.

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

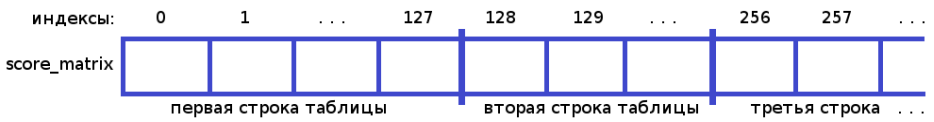


Рисунок 12 – Представление матрицы замен в памяти компьютера

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

$$nt\_score\_matrix[(unsigned\ char)'A' * 128 + (unsigned\ char)'C']$$

Полученное значение — цена 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 two profiles
  Profile& operator + (Profile& another);
  // alignment of profile and sequence
  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) {
    for_each(sequences.begin(), sequences.end(),
      [pos] (BioSeq* s) {
        s->nt_seq.insert(pos, 1, '-');
      }
    );
  }
};
```

При работе с профилями необходимо гарантировать, что ни одна из входящих в его состав последовательностей не будет модифицирована или удалена до конца построения выравнивания. В противном случае возможно появление ошибок выполнения или ухудшение качества результата.

3.2 Общая схема работы

На рисунке 13 показана общая блок-схема работы программы. Основные этапы реализованы в виде отдельных, независимых друг от друга блоков, которые впоследствии можно модифицировать или заменить новыми.

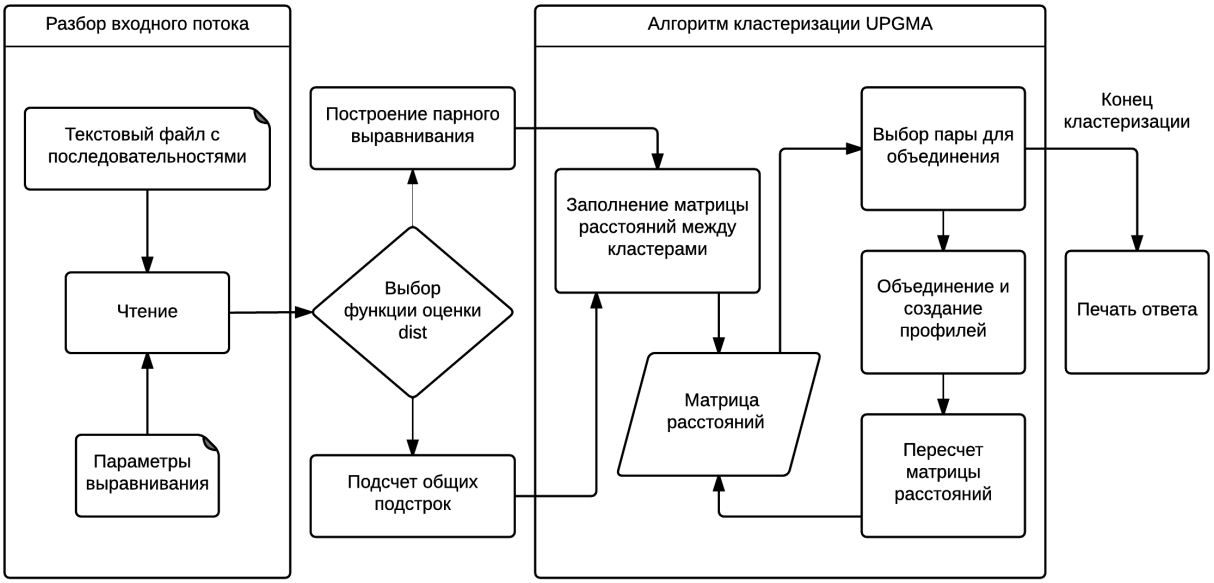


Рисунок 13 – Блок-схема работы программы

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 A, C, G и T кодируют, соответственно, аденин, цитозин, гуанин и тимин. На рисунке 14 представлен пример тестового файла в формате FASTA.

```
; Austrostipa
>AUSG51910|Austrostipa exilis|matK|JF769104
TTTATTGCGATTCTTTCTCAACTACTATTCGAATTGGAATACTCTTATTACTTCAATGAAATCGATTTTTATTTTGAAAA
AAGAAAATAAAAGACTATTTCAATTCCTATACAACTCTTATGTATCAGAATATGAATTTTTCTTGTTGCTTCTTCGTAAA
CAATCTTCTTGGTTATCATTAACATCTTCTGGAACCTTTCTAGAACGAATCCACTTTTCTAGGAAGATGGAACATTTTGG
GATAATGTACCCAGGTTTTTTTCGGAAAACAGTATGGTTCTTTATGGATCCTCTTATGCATTATGTTCGATATCAAGGAA
AGGCAATTCTTGCATCAAAAGGAACTCTTTTTTTGAAGAAGAAATGGAAATGTTAC
>AUSG47510|Austrostipa geoffrey|matK|JF769107
ACTATTCGAATTGGAATACTCTTATTACTTCAATGAAATCGATTTTTCTTTTGAAAAAAGAAAATAAAAGACTATTTCAA
TTCCTATACAACTCTTATGTATCAGAATATGAATTTTTCTTGTTGCTTCTTCGTAAACAATCTTCTTGGTTATCATTAAC
ATCTTCTGGAACCTTTCTAGAACGAATCCACTTTTCTAGGAAGATGGAACATTTTGGGATAATGTACCCAGGTTTTTTTC
GGAAAACAGTATGGTTCTTTATGGATCCTCTTATGCATTATGTTCGATATCAAGGAAAGGCA
```

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.

```
#  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

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. Для использования своей собственной функции расстояния 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 не активны и пропускаются при переборе. На рисунке 16 показан пример пересчета таблицы расстояний с указанием неактивных строк и обновляемых на каждом этапе элементов.

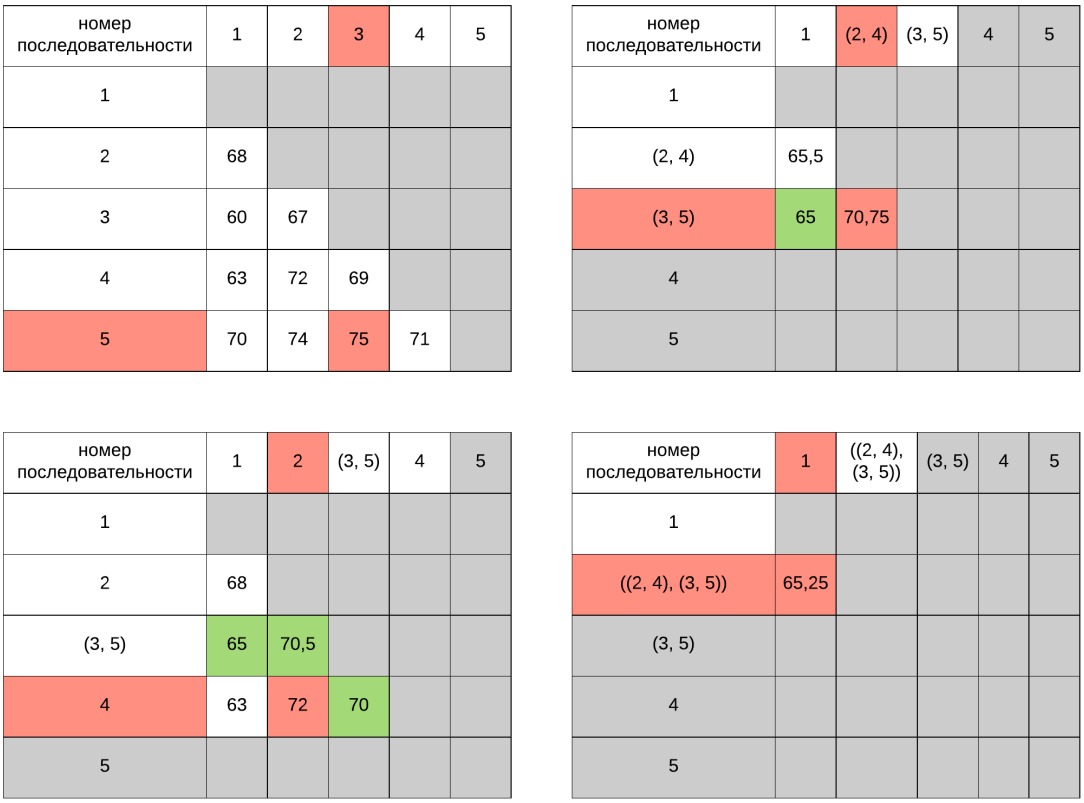


Рисунок 16 – Пересчет матрицы расстояний. Красным отмечены элементы для объединения, зеленым — обновленные значения, серым — неактивные ячейки.

3.2.3 Объединение профилей

При объединении профилей P_1 и P_2 происходит рассчет оптимального выравнивания с заполнением матрицы размерности $len(P_1) \cdot len(P_2)$. Для того 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 ← [ ]                                              ▷ Матрица оптимальных ходов
procedure ProfileMerge(P1, P2)
    if size(F) < len(P1) OR size(F) < len(P2) then   ▷ Проверка размерности матрицы
        new_size ← max(len(P1), len(P2))
        F ← [new_size × new_size]                    ▷ Выделение новой памяти
    end if
    for i = 1 ··· len(P1) do                         ▷ Цикл заполнения таблицы переходов
        for j = 1 ··· len(P2) do
            Рассчет оптимального перехода,           ▷ Позиция текущей ячейки — i, j
            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 *.c++ -o multy
```

Для получения версии с печатью отладочной информацией 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 *.c++ -o multy
```

Использование такой сборки предполагает поиск и устранение ошибок, возникающих в процессе выравнивания. В таком случае, обычно, 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 *.c++ -o multy
```

Также, для удобства использования, был написан Makefile — набор инструкций для программы make [21], благодаря которой 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

Короткие опции	Длинные опции	Значение	Комментарий
$-h$	$--help$	без аргумента	вывод справочной информации
$-i$	$--input$	путь к файл с последовательностями	файл должен быть представлен в формате FASTA
$-n$	$--NT\_subst$	путь к файлу с матрицей замен для нуклеотидов	по умолчанию: +4 за сопоставление одинаковых нуклеотидов и -5 в противном случае
$-a$	$--AA\_subst$	путь к файлу с матрицей замен для аминокислот	по умолчанию используется матрица замен BLOSUM62 [22]
$-g$	$--gap\_open$	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 *dist* алгоритм подсчета общих подстрок длины k; значение по умолчанию для k: 10
$-p$	$--pairwise$	без аргумента	использовать в качестве функции *dist* 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.txt
./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 -f -20 --AA_subst BLOSUM60.txt
Reading sequences...
7 sequences were obtained
Input parameters:
NT substitution matrix  NUC-45
AA substitution matrix  BLOSUM60.txt
Gap open cost           -10
Gap extension cost      -3
Gap frame cost          -20
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.

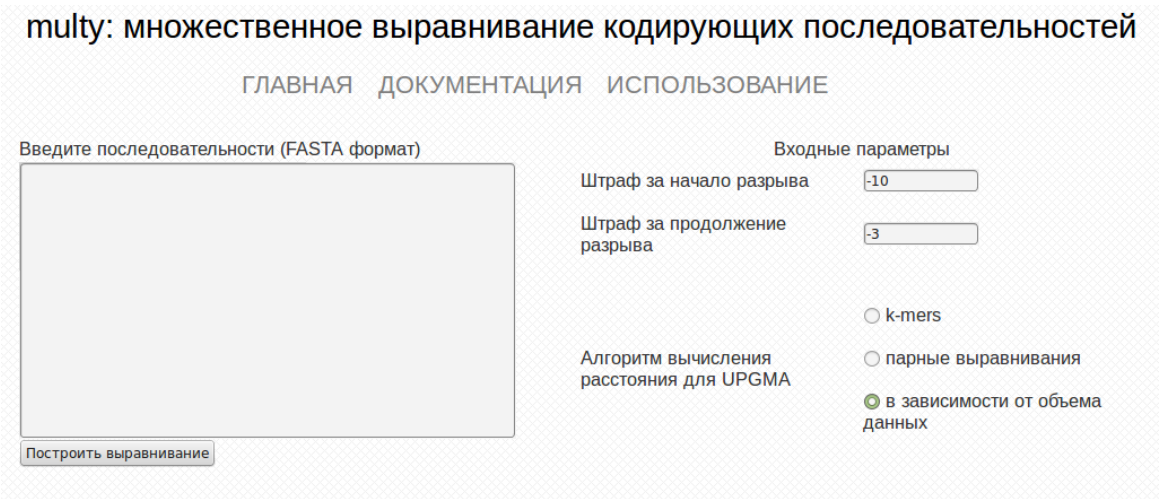


Рисунок 17 – Графический интерфейс для построения выравнивания

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

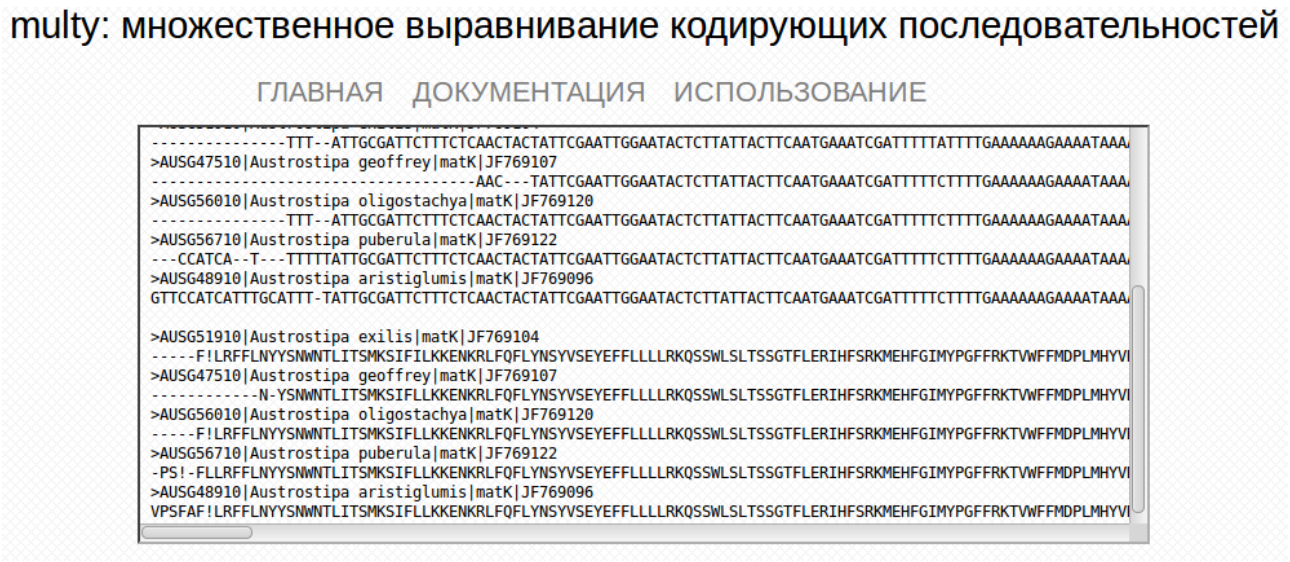


Рисунок 18 – Форма с результатами выравнивания

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 программы с импортированными методами и структурами построения выравниваний, необходимо указать компилятору используемую библиотеку. Ниже представлен пример компиляция новой программы *new_program.c++* с библиотекой *libmulty.a*:

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

4 Оценка производительности

Производительность разработанного алгоритма против существующей программы парного выравнивания MACSE оценивалась на группах синтетических тестов — наборы созданных случайных последовательностей нуклеотидов заданной длины. Тестирование происходило на машине с процессором Intel Xeon E5645 под управлением ОС Linux Debian 7 (wheezy). Результаты тестирования представлены в таблице 3, гистограмме (рисунок 19) и на точечном графике (рисунок 20).

При тестировании были рассмотрены различные сборки разработанной программы: был использован компилятор intel icpc с различными флагами оптимизации и компилятор gcc с флагом -O3. Из графиков и таблицы хорошо видно серьезное замедление времени выполнения на больших тестах для обоих алгоритмов. Сложность полученного решения отличается от MACSE только на константу, тем не менее, был получен существенный прирост производительности.

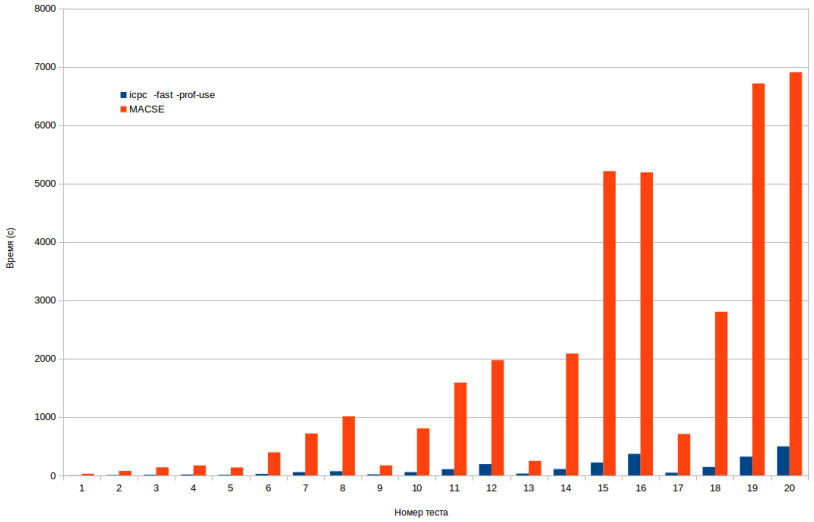


Рисунок 19 – Время выполнения тестов разработанного алгоритма и MACSE

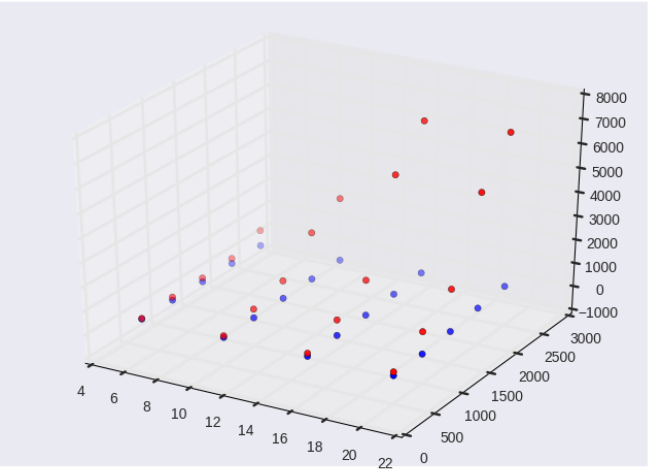


Рисунок 20 – График роста времени выполнения при увеличении объема входных данных

Таблица 3: Результаты тестирования

Длина поледовательностей	Количество последовательностей	Время выполнения (с)						
		Опции компиляции						MACSE
		icpc -O0	icpc -fast	icpc -fast -prof-use	icpc -fast -parallel	icpc -fast -prof-use -parallel	gcc -O3	
500	5	11.857	2.364	2.256	2.344	2.18	2.832	27.294
500	10	40.171	6.724	6.096	7.196	6.06	7.748	76.309
500	15	75.969	12.493	11.409	11.997	11.097	14.017	137.789
500	20	114.995	18.697	14.969	16.029	15.713	18.473	169.339
1000	5	48.759	9.093	9.625	8.849	8.101	10.837	136.294
1000	10	165.338	27.446	25.194	28.518	25.31	34.662	394.305
1000	15	396.281	62.408	55.759	62.884	60.908	69.872	716.885
1000	20	556.679	75.113	70.672	86.685	74.793	87.449	1012.063
1500	5	100.462	18.057	17.009	18.117	16.553	22.037	171.643
1500	10	384.48	60.36	55.371	65.152	57.016	78.105	805.822
1500	15	829.996	117.475	106.651	115.611	105.979	135.952	1589.375
1500	20	1551.437	205.153	192.72	203.841	196.264	252.428	1974.075
2000	5	184.928	33.158	31.106	40.807	29.294	39.91	248.244
2000	10	669.37	106.687	107.559	103.818	101.83	124.38	2087.51
2000	15	1540.484	219.714	219.914	273.977	203.241	257.508	5212.532
2000	20	2909.802	397.469	368.763	401.813	375.587	529.841	5191.924
2500	5	290.254	53.391	46.495	50.919	46.239	62.136	708.756
2500	10	1006.799	156.85	144.593	166.842	143.621	187.908	2803.231
2500	15	2417.331	344.09	321.168	352.034	334.809	408.606	6714.268
2500	20	3944.915	596.849	496.883	571.988	511.28	665.694	6907.332

ВЫВОДЫ

Целью настоящей работы являлось создание приложения для построения множественных выравниваний кодирующих последовательностей ДНК с учетом сдвигов рамки считывания. Поставленная задача была успешно выполнена, кроме этого, разработанная программа в результате тестирования показала хороший прирост скорости по сравнению с существующим аналогом. Также были созданы статические библиотеки парного и множественного выравниваний для использования в сторонних проектах. Приложение написано на языке программирования C++ и является кроссплатформенным. Для упрощения работы дополнительно к программе был создан веб-интерфейс.

Дальнейшее развитие проекта предполагает поиск новых подходов для улучшения качества получаемого ответа. При объединении профилей разработанный алгоритм не пересматривает предыдущие решения. Любой поставленный разрыв будет присутствовать в итоговом выравнивании. Возможно, дополнительный набор проверок и перевыравниваний сделает ответ более целым и, соответственно, более качественным.

Список литературы

[1] Миронов Андрей Александрович. Лекция "Введение в биоинформатику". Режим доступа — URL: http://mipt.ru/dbmp/student/files/bioinformatics/public_lection/.

[2] Mount DM. Bioinformatics: Sequence and Genome Analysis. — 2nd. — Cold Spring Harbor, NY., 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.

[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.

[9] Параллельный алгоритм глобального выравнивания с оптимальным использованием памяти [электронная публикация]. — URL: http://www.science-education.ru/107-8139.

[10] Учебное пособие: Биология. Ярыгин книга 1. Под редакцией академика РАМН профессора В.Н. Ярыгина.

[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 [электронная публикация]. — URL: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0022594.

[18] Legendre Pierre, Legendre Louis. Numerical ecology: second English edition // Developments in environmental modelling. 1998. T. 20.

[19] What is FASTA format? [электронная публикация]. — URL: http://zhanglab.ccmb.med.umich.edu/FASTA/.

[20] The GNU Compiler Collection [электронная публикация]. — URL: http://gcc.gnu.org/.

[21] GNU Make [электронная публикация]. — URL: http://www.gnu.org/software/make/.

[22] BLOSUM Clustered Scoring Matrix [электронная публикация]. — URL: http://www.ncbi.nlm.nih.gov/Class/FieldGuide/BLOSUM62.