Санкт-Петербургский государственный университет информационных технологий,
механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Капун Евгений Дмитриевич

# Разработка метода сравнения нуклеотидных последовательностей путём разбиения на фрагменты

Научный руководитель: доктор технических наук, профессор А. А. Шалыто.

Санкт-Петербург
2010

# Содержание

# Введение

*Геном* — совокупномть информации, пердаваемой живыми существами по наследству. *Геномика* — наука, изучающая геном. В её задачи входят, среди прочего, секвенирование геномов и определение механизма связи между генами и признаками.

*Дезоксирибонуклеиновая кислота* (*ДНК*) и *рибонуклеиновая кислота* (*РНК*) — молекулы, хранящие и переносящие генетическую информацию. Они представляют собой цепочки нуклеотидов, закрученные в спирали. На их основе синтезируются белки согласно *генетическому коду*.

*Мутация* — самопроизвольное или спровоцированное изменение генома, передающееся по наследству. Мутации могут быть точечными, а могут затрагивать большие участки ДНК. Считается, что мутации являются необходимым фактором эволюции и образования новых видов.

В настоящее время существует несколько способов электронного представления генетических данных. Среди них наиболее популярные — форматы *FASTA* и *FASTQ*. Оба эти формата текстовые и кодируют каждый нуклеотид одной буквой.

Существующие алгоритмы позволяют выравнивать биологические последовательности. Это позволяет оценить их степень сходства, но без учёта не точечных мутаций.

В настоящей работе предложен алгоритм, позволяющий оценить сходство двух биологических последовательностей и учитывающий как точечные мутации, так и мутации, затрагивающие более длинные участки.

# Глава 1.

# Обзор предметной области

## 1.1. Геномика

### 1.1.1. Ген и геном

*Ген* — структурная единица информации, передаваемой от живого существа к потомкам при размножении. Вся информация о строении отдельных клеток и всего организма, структуре всех белков, а также о процессах, происходящих в организме, закодирована в генах.

*Геном* — совокупность генов некоторой особи. Развитие и функционирование всех известных живых существ определяется их геномом. Кроме того, геном определяет то, каким может получиться потомство того или иного существа.

Различия между разными видами, как и между разными особями одного вида, возникают в результате того, что у них различные гены. Также гены отвечают за передачу потомкам признаков родителей.

На молекулярном уровне гены представляют собой участки молекул ДНК или РНК.

Большая часть живых существ хранят свои гены в нескольких длинных молекулах ДНК, находящихся в клеточном ядре. При *экспрессии* генов соответствующий участок ДНК *транскрибируется*, в результате чего создаётся молекула *транспортной РНК* (*mPHK*), которая в дальнейшем может используется для синтеза молекулы белка.

Многоклеточные организмы хранят полную копию своего генома в каждой клетке. Кроме того, существа, размножающиеся половым путём, обычно состоят из диплоидных клеток. Это значит, что каждая клетка хранит два экземпляра генома — по одному от каждого родителя. При размножении эти экземпляры комбинируются, для того чтобы получить экземпляр для передачи потомку.

Кроме ядра, некоторые органеллы (структуры, постоянно существующие внутри клетки), такие как митохондрии и хлоропласты, содержат собственные молекулы РНК, которые передаются при делении этих органелл. Эти РНК не всегда считаются частью генома.

Большая часть генома эукариотов (живых существ, клетки которых содержат ядро) состоит из некодирующих последовательностей — последовательностей, которые не хранят информацию о строении белков и РНК.

### 1.1.2. Геномика

*Геномика* — раздел генетики, изучающий геномы в целом. В задачи геномики входит:

- *Секвенирование* ДНК — определение последовательности нуклеотидов, из которых она состоит.
- Определение структуры информации в геноме.
- Определение связи между генами и признаками, которые они кодируют.
- Определение степени генетического сходства между различными видами и особями.
- Определение расстояния между генами — вероятности того, что некоторые гены окажутся на разных хромосомах в результате кроссинговера.

## 1.2. Структура молекул ДНК и РНК

### 1.2.1. Строение ДНК и РНК

*ДНК* и *РНК* — вещества, хранящие генетическую информацию в клетке. Молекула каждой из этих кислот представляет собой двойную спираль, каждая цепь которой состоит из нуклеотидов [1] [2]. Нуклеотиды, находящиеся друг напротив друга, соединены водородными связями, которые удерживают цепи вместе. Каждый нуклеотид содержит в себе азотистое основание, которое может быть одного из четырёх видов, в остальном все нуклеотиды внутри молекул одного вещества одинаковые.

Молекула ДНК может содержать следующие виды азотистых оснований: *аденин*, *гуанин*, *тимин* и *цитозин*. Молекулы РНК могут содержать *аденин*, *гуанин*, *урацил* и *цитозин*.

Пары нуклеотидов, лежащие друг напротив друга, соединяются согласно *принципу комплементарности*: аденин может соединяться только с тимином и урацилом, а гуанин — с цитозином. Таким образом, последовательность нуклеотидов одной цепи позволяет однозначно определить последовательность нуклеотидов во второй. Последовательности нуклеотидов, соответствующие двум цепям одной молекулы ДНК или РНК, называются *комплементарными*.

Каждая из цепей в молекулах ДНК и РНК имеет выделенное направление, причём это направление противоположно для двух цепей в одной молекуле. Поэтому, кроме замены одних нуклеотидов на другие, комплементарные цепи отличаются ещё и тем, что порядок нуклеотидов в них противоположен друг другу.

### 1.2.2. Репликация ДНК

Комплементарность позволяет клеткам *реплицировать* ДНК. Репликация — процесс создания копии молекулы. Для его осуществления специальные ферменты разделяют цепи ДНК и синтезируют начальные участки новых цепей [3] [2]. Затем

фермент *ДНК полимераза* движется вдоль одной из цепей и достраивает новую, комплементарную к ней цепь. Поскольку цепи направлены в разные стороны, а синтез может происходить лишь в одном направлении, вторая цепь достраивается до двойной спирали небольшими участками. Эта цепь называется отстающей, а первая цепь — лидирующей.

### 1.2.3. Транскрипция ДНК

*Транксрипция* — более простой процесс, позволяющий получить молекулу РНК, кодирующую информацию с одного из участков ДНК. Для этого необходимый участок ДНК временно разделяется на цепи. Затем одна из цепей используется как шаблон для новой цепи РНК. Из-за принципа комплементарности, однако, последовательность нуклеотидов в РНК оказывается идентичной последовательности нуклеотидов в *другой* цепи ДНК, за исключением того, что тимин заменён на урацил.

## 1.3. Генетический код

*Генетический код* — способ кодирования структуры белков при помощи последовательности нуклеотидов.

Молекула белка представляет собой цепочку из аминокислот. Хотя общее число различных аминокислот достаточно велико, лишь небольшое число аминокислот, так называемые *стандартные аминокислоты*, могут становиться частью молекул белка. Всего известно 20 стандартных аминокислот.

Поскольку молекулы ДНК и РНК могут содержать лишь четыре вида нуклеотидов, каждая аминокислота кодируется последовательностью из трёх нуклеотидов, называемой *триплетом*. Всего возможны $4^3 = 64$ различных триплета.

Поскольку число возможных триплетов больше, чем число стандартных аминокислот, некоторые аминокислоты кодируются несколькими возможными триплетами (до шести триплетов для одной аминокислоты). Это придаёт коду некоторую степень помехоустойчивости — некоторые изменения последовательности нуклеотидов не приводят к изменению последовательности аминокислот, которую она кодирует.

## 1.4. Передача генома по наследству

Одноклеточные организмы, такие как бактерии, размножаются делением. При этом одна особь превращается в две, и геном каждой из них повторяет родительский.

Некоторые многоклеточные организмы размножаются бесполым путём. При этом особь-родитель производит множество особей-потомков, и опять же каждая из них получает копию родительского генома.

Однако, наиболее сложные формы жизни размножаются половым путём. При этом у каждой особи имеется двое родителей (хотя при *партеногенезе* оба родителя — одна и та же особь). Кроме того, каждая особь содержит не один, а некоторое чётное число (чаще всего два) экземпляров генома, из которых половина досталась

ей от одного из родителей, а половина — от другого. При размножении организм таких существ производит *гаметы* — половые клетки.

Каждая такая клетка содержит половину обычного набора хромосом. Гаметы образуются в результате процесса, называемого *мейозом*. При мейозе происходит *кроссинговер* — обмен участками между гомологичными хромосомами (хромосомами, выполняющими одни и те же функции в различных экземплярах генома). В результате геном половых клеток представляет собой результат смешения геномов, доставшихся особи от родителей. Кроссинговер увеличивает генеточеское разнообразие потомства, что в итоге даёт преемущество при естественном отборе.

Хотя обычно кроссинговер не вызывает каких-либо проблем, нарушения возможны при сочетании кроссинговера с *мутациями*, приводящими к переносу больших участков генома. Если порядок генов в гомологичных хромосомах не совпадает, то после кроссинговера может оказаться так, что в получившихся хромосомах некоторые гены встречаются по два раза, а некоторые — вообще отсутствуют. И то, и другое может привести к намного более серьёзным нарушениям, чем сама мутация, повлекшая перестановку.

## 1.5. Мутации

*Мутация* — изменение генома, которое может быть передано потомству. Мутации могут происходить самопроизвольно, а могут провоцироваться некоторыми факторами — *мутагенами*.

Мутация может не иметь никаких последствий для потомства, изменить его признаки или сделать его нежизнеспособным. Однако, некоторые мутации могут, наоборот, увеличить приспособленность особей, что даст им преимущество перед остальными. Считается, что мутации являются необходимым фактором эволюции. Мутации сами по себе случайны, однако благодаря естественному отбору особи с вредными мутациями преждевременно гибнут или не оставляют потомства, поэтому эти мутации не передаются следующим поколениям. Если же мутация оказалась полезной, то особь может не только выжить, но и иметь большие шансы на выжывание, чем особи, которые не мутировали. В итоге менее приспособленные особи оказываются вытесненными, и выживают только особи, оказавшиеся наиболее приспособленными к условиям окружающей среды.

Поскольку большинство мутаций вредны, клетки вырабатывают механизмы защиты от мутаций. В случае некоторых бактерий, эти механизмы позволяют им выживать, несмотря на сильное воздействие мутагенов (например, выдерживать большие дозы радиации).

Существуют несколько видов мутаций.

### 1.5.1. Точечные мутации

*Точечные мутации* — мутации, затрагивающие небольшой участок ДНК, обычно один нуклеотид. Такие мутации могут быть вызваны ошибками при репликации ДНК, а также воздействием мутагенов.

Сама мутация обычно заключается в замене одного нуклеотида на другой, хотя возможны и другие варианты — вставка нового нуклеотида или удаление существующего. Замены делятся на:

- *транзиции*, при которых пурин заменяется на пурин, а пиримидин — на пиримидин;
- *трансверсии*, при которых пурин заменяется на пиримидин и наоборот.

Не все возможные замены равновероятны: хотя возможных транзиций в два раза меньше, чем трансверсий, они случаются примерно в два раза чаще. Это связано с тем, что при транзиции нуклеотид заменяется на более схожий по химическим свойствам. Транзиции не вносят столь заметных изменений в структуру ДНК более высокого порядка, поэтому имеют меньший шанс сделать какую-либо критически важную часть ДНК неработоспособной.

Мутации, при которых происходит вставка или удаление нуклеотида, также называют *мутациями сдвига*. Дело в том, что нуклеотиды читаются триплетами, а из-за вставки или удаления все последующие триплеты считываются неправильно.

### 1.5.2. Дупликации и делеции

*Дупликация* — удвоение участка хромосомы.

Дупликации обычно происходят при кроссинговере, когда хромосомы обмениваются участками разной длины. В этом случае одна из хромосом получает участок, аналогичный одному из своих, но из другой хромосомы. В другой хромосоме, соответственно, происходит делеция.

Считается, что дупликации играют важную роль в эволюции. Копия, получившаяся при дупликации, может в дальнейшем мутировать независимо от оригинала. Это иногда приводит к появлению новых признаков.

Кроме того, дупликации приводят к усилению функции удвоенных генов. Как и большинство мутаций, это часто приводит к негативным последствиям для организма.

Кроме того, возможно удвоение хромосомы целиком. Иногда удваиваются все хромосомы, после этого клетка несёт сразу несколько копий всего генома. Это называется полиплоидизацией.

*Делеция* — удаление участка хромосомы, мутация, обратная дупликации.

Как и дупликации, они возникают в результате кроссинговера, хотя и не обязательно. Делеции делятся на *интерстициальные*, при которых удаляется внутренний участок хромосомы, и *терминальные*, при которых удаляемый участок лежит с конца хромосомы.

### 1.5.3. Транслокации

*Транслокация* — обмен участков двух хромосом. Обычно транслокации не приводят к нарушениям в работе организма, хотя в редких случаях они вызывают рак и

другие заболевания. Однако, транслокации зачастую приводят к образованию несбалансированных гамет, что может привести к появлению неполноценного потомства.

Хотя транслокация может затрагивать большой участок хромосомы, пострадать могут только те гены, которые оказались разорваны в результате транслокации. Кроме того, изменение порядка генов часто приводит к образованию клеток с несбалансированным геномом при *мейозе*, что может привести к образованию нежизнеспособных гамет или нарушениям в потомстве.

### 1.5.4. Инверсии

*Инверсия* — мутация, при которой участок хромосомы переворачивается. Поскольку цепи ДНК направлены, это также приводит к тому, что нуклеотиды заменяются на комплементарные.

Как и транслокации, инверсии редко сказываются на организме, в котором произошли. Намного чаще они приводят к отсутствию потомства или неполноценному потомству.

## 1.6. Представление генетической информации в электронном виде

Задача электронного представления генетической информации встала перед исследователями, когда появились устройства, способные считывать эту информацию. Поскольку различных нуклеотидов и стандартных аминокислот немного, логично, что их стали кодировать одним символом. Обычно для кодирования нуклеотида используется первая буква в его названии. В то же время, названия многих аминокислот начинаются с одинаковых букв, поэтому коды многих аминокислот не совпадают с из первыми буквами — используются те буквы, которые не заняты.

### 1.6.1. Формат FASTA

*FASTA* — один из наиболее популярных форматов для представления последовательностей нуклеотидов или аминокислот.

Файл в формате FASTA — простой текстовый файл. Первая строка должна начинаться с символа `>` или `;`. Она содержит имя последовательности и некоторую дополнительную информацию, предназначенную для идентификации. Другие строки, начинающиеся с `;`, являются комментариями и игнорируются.

После первой строки начинается, собственно, описание последовательности. При кодировании последовательности нуклеотидов, буквы `A`, `C`, `G`, `T` и `U` кодируют, соответственно, аденин, цитозин, гуанин, тимин и урацил. Также некоторые буквы кодируют позиции, в которых находится один нуклеотид из некоторого множества (это используется, если неизвестно, какой именно нуклеотид там находится). Символ `-` (дефис) кодирует неизвестную последовательность произвольной длины.

### 1.6.2. Формат FASTQ

*FASTQ* — формат представления биологической последовательности совместно с данными о качестве.

Этот формат используется для представления данных секвенирования, так как позволяет представить как саму последовательность, так и вероятность, что каждый из элементов последовательности указан правильно. Для этого, кроме символов, кодирующих элементы последовательности, используется символы, кодирующие уровень качества.

Уровень качества — целое число в некотором диапозоне. Известно два различных способа выражать уровень качества через вероятность ошибки. Чаще всего используется следующая формула:

$$Q = -10 \log_{10} p.$$

Здесь $Q$ — уровень качества, а $p$ — вероятность, что этот элемент последовательности — ошибочный. Реже применяется второй способ:

$$Q = -10 \log_{10} \frac{p}{1-p}.$$

При больших значениях $Q$ (и, соответственно, при малых $p$) эти способы дают практически идентичные результаты, но при малых значениях $Q$ и больших значениях $p$ их значения заметно различаются.

Сам же файл содержит четыре строки для каждой последовательности. Первая строка начинается с символа `@`, после которого идёт описание последовательности, как и в формате FASTA. Следующая строка содержит последовательность символов, кодирующих саму последовательность, аналогично формату FASTA. За ней идёт строка, начинающаяся с символа `+`, после которого может идти описание последовательности (третья строка будет отличаться от первой только тем, что первый символ заменён на `+`), а может ничего не идти.

Последняя строка содержит уровни качества. Её длина равна длине второй строки, а каждый символ кодирует информацию о качестве элемента последовательности, закодированного соответствующим символом второй строки. Сами же уровни кодируются таким образом: ASCII-код символа равен уровню качества плюс некоторая константа. Константа обычно имеет значение 33 или 64. В любом случае, код символа не должен превышать 127.

### 1.6.3. Формат GenBank

Формат GenBank позволяет представить больше дополнительной информации о последовательности. Файл в формате GenBank состоит из нескольких записей, каждая из которых может занимать несколько строк. Все строки в записи, кроме первой, начинаются с пробела, это позволяет легко находить границы записей.

Каждая запись начинается с имени, за которым идёт значение (через один или несколько пробелов. Некоторые записи могут содержать подзаписи. Они форматируются аналогично записям. После этого в начало каждой строки записывается на-

сколько пробелов. Таким образом, значения подзаписей выравнены правее значений записей, а те выравнены правее имён записей.

Имена записей имеют предопределённое значение. Например, запись `DESCRIPTION` хранит описание последовательности, запись `SOURCE` идентифичирует особь, с которой считана последоваиельность, а запись `REFERENCE` (их может быть несколько) используется для ссылок на публикации.

Запись `ORIGIN` содержит саму последовательность. Каждая строка, кроме последней, содержит 60 элементов, разбитых пробелами на группы по 10. Для представления элементов используются строчные буквы. В начало каждой строки добавляется текужая позиция, начиная с единицы, и пробел.

## 1.7. Существующие методы поиска гомологий в биологических последовательностях

*Гомология* — структурное сходство. В генетике под гомологиями понимаются участки белков или ДНК, имеющую 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. 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.

Поэтому существующие методы поиска гомологий умеют находить участки (подстроки) двух последовательностей, которые отличаются не очень сильно.

### 1.7.1. Алгоритм Нидлмана-Вунша

*Алгоритм Нидлмана-Вунша* [4] был опубликован в 1970 году и позволяет определять степень сходства последовательностей, а также находить глобальное выравнивание, — находить, какой именно символ из одной последовательности соответствует некоторому символу из другой последовательности.

Для своей работы алгоритм использует матрицу сходства, которая указывает, насколько схожими считать разные нуклеотиды. Для различных нуклеотидов используются отрицательные элементы. Поэтому последовательности должны содержать некоторую долю совпадающих нуклеотидов, для того чтобы быть признаными гомологичными.

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 пиримиди-

дина на пиримидин, более схожими, чем те, которые отличаются заменой пурина на пиримидин или наоборот.

Вообще, матрица позволяет приписать любой вес любым заменам. Обычно используется симметричная матрица, однако применение несимметричной матрицы позволяет различать замены в одну и в другую сторону.

Вот пример матрицы сходства:

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

Здесь **А**, **Г**, **Т** и **Ц** обозначают, соответственно, аденин, гуанин, тимин и цитозин, а числа в матрице указывают степень сходства между двумя нуклеотидами.

Алгоритм Нидлмана-Вунша способ сопоставить символы двух последовательностей так, что сумма значений сходства для соответствующих символов максимально. Кроме того, алгоритм может учитывать вставки и удаления. При этом считается, что символ в одной строке, которому не соответствует никакой символ из другой строки, имеет некоторый уровень сходства, который является параметром алгоритма (например, $-5$).

Для работы алгоритм использует матрицу $F_{i,j}$, 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, $di$ и $dj$ для всех $i$ и $j$. Затем он вычисляет значения в матрице по следующей формуле:

$$F_{i,j} = \max \begin{cases} F_{i-1,j-1} + S_{A_{i-1},B_{j-1}} \\ F_{i-1,j} + d \\ F_{i,j-1} + d \end{cases}.$$

Здесь $A_i$ и $B_j$ — соответственно $i$-й и $j$-й символы в первой и второй строках, $S$ — матрица сходства, а $d$ — степень сходства символа, который ничему не соответствует. Другими словами, это уровень сходства, добавляемый при вставке или удалении.

После того, как матрица посчитана, значение правого нижнего её элемента и будет степенью схожести двух последовательностей. Для того чтобы произвести выравнивание, необходимо отследить, каким путём появилось это значение, и перейти к предыдущему. 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.

Таким образом, если переходить от элемента в правом нижнем углу к предыдущему, от него к предудущему, и т. д., то в итоге путь приведёт к левому верхнему элементу, у которого нет предыдущего. Теперь, переход вида $(i, j) \to (i-1, j-1)$ означает, что $(i-1)$-му символу в первой строке соответствует $(j-1)$-й символ во второй строке. Переход вида $(i, j) \to (i-1, j)$ означает, что $(i-1)$-му символу первой строки ничего не соответствует, а переход $(i, j) \to (i, j-1)$ — что $(j-1)$-му символу второй строки ничего не соответствует.

Хотя этот алгоритм всегда находит оптимальное выравнивание, его существенным недостатком является большое время работы и большое потребление памяти (и то и другое — порядка произведения длин строк).

### 1.7.2. Алгоритм Смита-Вотермана

*Алгоритм Смита-Вотермана* [5] аналогичен алгоритму Нидлмана-Вунша, но при этом решает задачу локального выравнивания: находит подстроки первой и второй строк, обладающие максимальным сходством, а также выравнивает их. Алгоритм был опубликован в 1981 году.

Как и алгоритм Нидлмана-Вунша, алгоритм Смита-Вотермана использует матрицу сходства. Это позволяет учитывать различные замены с различным весом. Также он позволяет учитывать разные добавления и удаления по-разному в зависимости от того, какой именно нуклеотид был добавлен или удалён.

Реализация алгоритма Смита-Вотермана похожа на реализацию алгоритма Нидлмана-Вунша: берётся матрица $F_{i,j}$ такого же размера, однако, элементы $F_{i,0}$ и $F_{0,j}$ инициализируются нулями. Пересчёт происходит по следующей формуле:

$$F_{i,j} = \max \begin{cases} F_{i-1,j-1} + S_{A_{i-1},B_{j-1}} \\ F_{i-1,j} + D_{A_{i-1}} \\ F_{i,j-1} + I_{B_{j-1}} \\ 0 \end{cases} .$$

Здесь $A_i$ и $B_j$ — символы из первой и второй строк, $S$ — матрица сходства для пар символов, $I$ — вектор стоимостей добавления (взятых с противоположным знаком), а $D$ — вектор стоимостей удаления.

Теперь максимальная степень сходства будет максимальным элементом матрицы. Если переходить от этого элемента по цепочке предыдущих, то путь закончится в каком-то нулевом элементе — не обязательно левом верхнем. 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.

Как и алгоритм Нидлмана-Вунша, алгоритм Смита-Вотермана всегда находит оптимальное решение, да и время работы и занимаемая память у него такие же. Это делает эти алгоритмы неприемлемыми для работы с большим количеством генетического материала.

# Глава 2.

# Описание используемого подхода

## 2.1. Критерий сходства биологических последовательностей

Чтобы оценивать сходство двух последовательностей, необходимо придумать некоторую меру сходства. Предложенная в данной работе мера сходства — число от нуля до единицы, где единица соответствует максимальному сходству, а ноль — минимальному. Кроме того, мера удовлетворяет следующим свойствам:

- Сходство последовательности с самой собой равно единице.
- Сходство несхожих последовательностей обычно невелико.
- Сходство не изменяется значительно, если одну из последовательностей подвергнуть мутации. Чем менее вероятна мутация, тем сильнее изменяется мера сходства.
- Существует способ эффективно вычислять меру сходства для практически встречающихся последовательностей.

Сложность разработки подобной меры заключается в том, что нелокальные мутации могут приводить к значительным перестановкам частей последовательности. Способ, предложенный в данной работе, решает эту проблему следующим образом: назовём одну из последовательностей *исходной*, а другую — *целевой*. Будем рассматривать все возможные разбиения целевой последовательности на фрагменты (подстроки) и для каждого из них подсчитаем *степень различия* — число, тем большее, чем хуже, с точки зрения данного разбиения, целевая последовательность аппроксимирует исходную. Затем найдём минимум степени различия по всем разбиениям.

В каждом разбиении каждый фрагмент можно считать *свободным* или *связанным*. От этого зависит вклад этого фрагмента в степень различия. Вклад свободного фрагмента в степень различия пропорционален его длине с константой, являющейся параметром метода, и не зависит от его содержимого. Вклад же связанного фрагмента зависит от его содержимого: каждому связанному фрагменту сопоставляется подстрока исходной последовательности или комплементарной к ней, и вклад фрагмента в степень различия равен редакционному расстоянию между содержимым фрагмента и этой подстрокой. Здесь редакционное расстояние рассматривается в

обобщённом смысле: можно приписывать различные веса разным заменам, вставкам и удалениям.

Как и разбиение на фрагменты, назначение типов фрагментов и соответствующих подстрок происходит таким образом, чтобы минимизировать суммарную степень различия. Кроме того, чтобы сделать большое число коротких фрагментов менее оптимальным, к степени различие добавляется ещё одно слагаемое, пропорциональное общему числу фрагментов в разбиении.

Мера сходства вычисляется по степени различия путём применения линейного преобразования, переводящего нулевую степень различия в единицу, а максимально возможную степень различия в ноль. Поскольку одним из разбиений является разбиение на один свободный фрагмент, степень различия не превосходит такую, которая соответствует этому разбиению, поэтому удобно принимать это значение соответствующим нулевой степени сходства.

### 2.1.1. Устойчивость относительно мутаций и других изменений

Геном может изменяться в результате мутаций, а также, в случае полового размножения, за счёт кроссинговера, при котором может происходить обмен генами между разными геномами. Мутации, в свою очередь, делятся на *локальные*, которые вызывают замену небольшой подстроки геномной последовательности на другую небольшую подстроку, и *нелокальные*, которые сводятся к разрезанию последовательности на части, удвоению и удалению некоторых из них, замену некоторых на комплементарные и соединению того, что получилось, в некотором порядке. Кроссинговер тоже подходит под этот шаблон, но с тем отличием, что может происходить обмен частями между последовательностями.

Локальные мутации можно разделить на три вида:

1. Замена — наиболее частая мутация. При такой мутации один нуклеотид заменяется на другой.

2. Вставка — появление в последовательности нового нуклеотида.

3. Удаление — исчезновение из последовательности одного нуклеотида.

Степень различия последовательностей по отношению к таким изменением называется редакционным расстоянием. Хотя поиск редакционного расстояния в общем случае — непростая задача для длинных строк, в данном методе не требуется поиск редакционного расстояния в общем случае. Дело в том, что если редакционное расстояние между связанным фрагментом и соответствующей подстрокой достаточно велико, то такое разбиение заведомо неоптимально: можно сделать этот фрагмент свободным и таким образом уменьшить суммарную степень различия.

Более того, даже если фрагмент содержит немного изменений относительно его длины, но все эти изменения расположены недалеко друг от друга, можно разбить этот фрагмент на три фрагмента, вырезав из него кусок, содержащий большую часть изменений, и заменив его свободным фрагментом. В результате, во многих случаях можно не считать (или не продолжать считать) редакционное расстояние

между фрагментами, поскольку и так понятно, что оптимальное разбиение не содержит такой фрагмент.

Кроме того, использование редакционного расстояния не даёт локальным мутациям существенно изменять степень различия: если мутация попала в свободный фрагмент, то степень различия может лишь слегка измениться за счёт изменения длины фрагмента, а если мутация попала в связанный фрагмент, то, опять же, соответствующее изменение редакционного расстояния не будет большим.

В отличии от локальных мутаций, которые сказываются на редакционном расстоянии, нелокальные мутации непосредственно сказываются на самом разбиении. Например, если подвергнуть последовательность дупликации (вставить в неё копию её подстроки), то в соответствующее разбиение можно тоже вставить копию его подстроки (при этом крайние фрагменты могут оказаться обрезанными). При этом вклад в степень различия от добавленных фрагментов не превосходит вклад от исходных фрагментов. Кроме того, некоторый вклад в степень различия происходит из-за того, что тот фрагмент, в котором находится место вставки, оказывается разделён на два.

Другие нелокальные мутации учитываются аналогично. При учёте инверсий используется то, что подстроки, соответствующие связанным фрагментам, можно брать как из исходной последовательности, так и из комплементарной к ней.

### 2.1.2. Вычисление степени сходства

Пусть задано некоторое разбиение целевой последовательности на фрагменты. Введём следующие обозначения:

- $n_f$ — число свободных фрагментов.
- $n_b$ — число связанных фрагментов.
- $l_i$ — длина $i$-го свободного фрагмента.
- $d_i$ — редакционное расстояние между $i$-м связанным фрагментом и соответствующей ему подстрокой.
- $D_f$ — коэффициент при слагаемом, пропорциональном числу фрагментов.
- $D_l$ — коэффициент при суммарной длине свободных фрагментов.

Тогда общая степень различия, соответствующая этому разбиению, определяется следующей формулой:

$$D = D_f(n_b + n_f - 1) + \sum_{i=1}^{n_b} d_i + D_l \sum_{i=1}^{n_f} l_i.$$

В первом слагаемом $-1$ требуется, для того чтобы степень различия между одинаковыми последовательностями была равна нулю. Поскольку общее число фрагментов положительно, первое слагаемое не станет отрицательным.

Для вычисления $d_i$ используется матрица, аналогичная матрице сходства в алгоритме Смита-Вотермана. Однако, вместо сходства элементы матрицы соответствуют степени различия между заданными нуклеотидами. Кроме того, для добавлений и удалений вводятся векторы стоимостей, как в алгоритме Нидлмана-Вунша. Для вычисления $d_i$ используется следующая формула:

$$d_i = \sum_{j,k} S_{A_j,B_k} + \sum_j D_{A_j} + \sum_j I_{B_j}.$$

Здесь первая сумма берётся по парам индексов, соответствующих заменяемым символам, вторая сумма — по индексам удаляемых, а третья — по индексам добавляемых символов.

Поскольку любую последовательность можно покрыть одним свободным фрагментом, $D$ не превосходит $D_l l_t$, где $l_t$ — длина целевой последовательности. Это значит, что $\frac{D}{D_l l_t}$ — число от нуля до единицы, являющееся нормализованной степенью различия, а $1 - \frac{D}{D_l l_t}$ — число от нуля до единицы, являющееся искомой мерой сходства двух последовательностей.

## 2.2. Представление последовательности нуклеотидов в памяти компьютера

Существует несколько форматов файлов, предназначенных для хранения нуклеотидных последовательностей. Однако, эти форматы создавались для удобного восприятия человеком, а также для совместимости с программами и протоколами, рассчитанными на работу с текстовыми данными.

В то же время, для работы с нуклеотидной последовательностью желательно такое представление, с которым можно быстро выполнять следующие операции:

- Индексация — определение нуклеотида по порядковому номеру.
- Взятие подстроки — получение получение последовательности, содержащей те из нуклеотидов исходной последовательности, порядковые номера которых лежат в заданном диапозоне.
- Сравнение — определение, совпадают ли две заданные последовательности.
- Хеширование — операция, получающая по заданной последовательности число таким образом, чтобы одинаковым последовательностям соответствовали одинаковые числа, а разным, по возможности, разные.

Форматы вроде FASTA допускают наличие в файле произвольного числа символов перевода строки между символами, кодирующими нуклеотиды, что препятствует эффективной индексации и сравнению, поэтому этот формат не подходит для выполнения операций, при которых необходима возможность произвольного доступа к последовательности.

Вместо этого, в данной работе используется двоичное представление последовательности. Поскольку в одной последовательности возможны четыре разновидности нуклеотидов, каждый нуклеотид кодируется двумя битами. Таким образом, каждый байт кодирует сразу четыре нуклеотида.

Поскольку современные процессоры обрабатывают информацию блоками сразу по четыре или даже по восемь байт, подобное кодирование позволяет в несколько раз быстрее выполнять операции, которые не требуют обращения к индивидуальным нуклеотидам, например, сравнение, которое можно делать поблочно.

Кроме того, для представления подстрок используется механизм ссылок: вместо копирования данных при взятии подстроки создаётся объект, ссылающийся на исходные данные, с указанием смещения и длины подстроки.

## 2.3. Алгоритм сравнения последовательностей

Сравнение нуклеотидных последовательностей происходит в два этапа. На первом этапе создаётся словарь, позволяющий быстро находить участки исходной последовательности, совпадающие с заданными короткими последовательностями (в качестве которых используются подстроки целевой последовательности).

На втором этапе ищется оптимальное разбиение целевой последовательности на фрагменты. При этом словарь используется, для того чтобы быстро находить подстроки исходной последовательности, соответствующие связанным фрагментам. Поскольку в оптимальном покрытии редакционное расстояние между связанным фрагментом и соответствующей ему подстрокой невелико, в них найдётся подстрока, совпадающая полностью. Каждое такое совпадение расширяется до тех пор, пока ошибок не станет настолько много, что дальнейшее расширение не имеет смысла.

Для подсчёта степени различия используется динамический алгоритм: подсчитывается степень различия для каждого префикса целевой последовательности, а фрагменты генерируются от начала целевой последовательности к её концу, что позволяет эффективно пересчитывать степени различия. Результатом является степень различия для всей целевой последовательности.

### 2.3.1. Создание словаря

Для быстрого поиска схожих участков в данной работе используется *словарь* — структура, позволяющая по нуклеотидной последовательности определённой длины (ключу) быстро найти все её вхождения в исходной подпоследовательности, а также в последовательности, комплементарной к ней.

В качестве словаря используется хеш-таблица, в которой ключами служат последовательности, а значениями — списки вхождений. Для каждого вхождения указывается, в какой из строк находится это вхождение (исходной или комплементарной), а также смещение этого вхождения относительно начала последовательности.

Для создания словаря перебираются все подстроки исходной последовательности некоторой длины, а также подстроки последовательности, комплементарной к ней. Длина ключа является параметром алгоритма и подбирается так, чтобы обеспечить

оптимальное потребление ресурсов: если ключи будут слишком короткими, то каждой подстроке будет соответствовать слишком длинный список позиций, поэтому увеличится время работы.

Если же ключи будут слишком длинными, то алгоритм будет потреблять слишком много памяти. Кроме того, из-за слишком длинных ключей алгоритм может начать пропускать фрагменты, в которых нет полностью совпадающих участков достаточной длины. На практике оптимальная длина ключа получается около $\log_4 l_s - 1$, где $l_s$ — длина исходной последовательности.

Для каждой подстроки соответствующей длины проверяется, есть ли она в словаре. Если есть, то в соответствующий список добавляется новое вхождение. Если нет, то в словарь добавляется новая запись, в котором ключом служит подстрока, а значением — новый список, содержащий единственное вхождение.

### 2.3.2. Поиск оптимального покрытия

Поиск оптимального покрытия производится методом динамического программирования: для каждого префикса в целевой строке хранится минимальная степень различия и пересчитывается на основе информации о возможных фрагментах. Кроме минимальной степени различия хранится минимальная степень различия для случаев, когда в конце префикса находится свободный фрагмент. Это включает случаи, когда этот фрагмент имеет нулевую длину. Это требуется для того, чтобы легче было учитывать свободные фрагменты.

Пусть $g_i$ — минимальная степень различия для префикса длины $i$, а $h_i$ — минимальная степень различия для префикса длины $i$ при условии, что последний фрагмент — свободный. Тогда выполняется неравенство:

$$g_i \leq h_i \leq g_i + D_f.$$

Здесь $D_f$ — коэффициент к добавке к степени различия, пропорциональной числу фрагментов.

Во всех точках, кроме нуля, эти значения инициализируются бесконечностями. В нуле используются значения $g_0 = -D_f$ (ноль фрагментов, поэтому отрицательное число) и $h_0 = 0$ (здесь один пустой свободный фрагмент). Заметим, что $g_0$ — единственный из всех $g_i$ и $h_i$ может быть отрицательным, поскольку остальные соответствуют покрытиям, содержащим хотя бы один фрагмент.

Первое неравенство выполняется, потому что $g_i$ — минимум на множестве покрытий префикса длины $i$, а $h_i$ — минимум на подмножестве этого множества. Второе неравенство выполнено, так как к любому покрытию можно добавить свободный фрагмент длины ноль, что увеличит степень различия на $D_f$.

В процессе работы алгоритма поддерживается инвариант, что для всех префиксов длины меньше некоторого $i$ числа $g_i$ и $h_i$ уже вычислены и в дальнейшем не будут изменяться, а для остальных префиксов значения ещё не вычислены, поэтому не будут использоваться. В процессе работы алгоритма $i$ может только увеличиваться. Для того чтобы выполнить пересчёт, соответствующий фрагменту, необходимо

выполнить операцию

$$g_e \leftarrow \min(g_e, g_b + d + D_f),$$

где $b$ — индекс начала фрагмента, $e$ — индекс его конца, а $d$ — его вклад в степень различия. Существенно, что $g_b$ уже вычислено, а $g_e$ ещё нет.

При таком методе требуется выполнять пересчёт для каждого фрагмента в тот момент, когда позиция с индексом $i$ лежит внутри него. Это условие выполняется автоматически, если начинать поиск фрагмента с ключа, начинающегося с позиции $i$. После того, как массивы $g$ и $h$ обновлены с учётом найденных связанных фрагментов, необходимо увеличить $i$ на один. Перед этим следует пересчитать $g_{i+1}$ и $h_{i+1}$ для учёта свободных фрагментов. При этом $g_{i+1}$ вычисляется по формуле

$$g_{i+1} \leftarrow \min(g_{i+1}, h_i + D_l)$$

(можно взять значение, полученное при пересчёте или продолжить свободную последовательность). Здесь $D_l$ — вклад единицы длины свободного фрагмента в степень различия. Число $h_{i+1}$ вычисляется по формуле

$$h_{i+1} \leftarrow \min(h_i + D_l, g_{i+1} + D_f)$$

(можно продолжить свободную последовательность или начать новую). Заметим, что $g_{i+1}$ может присваиваться до этого пересчёта, поэтому использование этого значения до присваивания осмысленно.

Алгоритм заканчивает работу, когда $i = l_t + 1$ — когда пройдена вся целевая строка. Тогда $D = g_{l_t+1}$ по индукции.

### 2.3.3. Сканирование целевой строки

Чтобы быстро учесть все связанные фрагменты, получающиеся из вхождения ключа, используется такой метод: пусть ключ входит, начиная с позиции $i$ целевой строки и $j$ исходной. Поскольку сам ключ совпадает полностью, редакционное расстояние между фрагментом и соответствующей подстрокой складывается из двух слагаемых: редакционное расстояние между участками до ключа и после ключа.

Соответственно, если $s$ и $e$ — индексы начала и конца фрагмента в целевой строке, то выполняется неравенство:

$$g_e \leq g_s + d_{s,i} + d_{i,e}.$$

Здесь $d_{s,i}$ и $d_{i,e}$ — вклад в редакционное расстояние участков от $s$ до $i$ и от $i$ до $e$. Чтобы обеспечить это неравенство, можно было бы перебирать все возможные значения $s$ и $e$ в некотором диапозоне, но можно сделать это намного быстрее: вначале найти $\min_s(g_s + d_{s,i})$ перебором $s$, а затем перебирать только значения $e$.

Для определения границ перебора $s$ и $e$ используется изложенный выше критерий оптимальности: если для какого-то $s$ выгодно заменить подстроку от $s$ до $i$ свободным фрагментом, то дальнейшее уменьшение $s$ не имеет смысла. Аналогично для $e$.

## 2.4. Оценка сложности алгоритма

Поскольку представленный в данной работе алгоритм использует эвристики, его сложность существенно зависит от входных данных. В частности, чем более схожи исходная и целевая последовательности, тем медленнее будет работать алгоритм.

Пусть $l_k$ — длина ключа в словаре. Тогда к словарю будет в общей сложности $l_t - l_k + 1$ запросов, что примерно равно $l_t$. После каждого запроса, каждое вхождение в словарь обрабатывается отдельно. Если бы последовательности были случайными, то каждый запрос выдавал бы в среднем около $\frac{l_s}{4^{l_k}}$ вхождений. На самом деле вхождений обычно заметно больше.

Для каждого вхождения, происходит посимвольное сравнение строк, которое обрывается, если строки становятся слишком непохожими. Для случайных строк это будет происходить достаточно быстро, но в реальных нуклеотидных последовательностях похожие участки встречаются достаточно часто и могут быть весьма длинными, поэтому большую часть времени работы программы занимает именно это сравнение.

В лучшем случае ни одна из подстрок длины $l_k$ из целевой последовательности не содержится в исходной. При этом алгоритм будет работать за время $O(l_k(l_s + l_t))$, где первое слагаемое — время составления словаря, а второе — время на запросы к нему.

В худшем случае все подстроки длины $l_k$ в исходной и целевой последовательностях совпадают. Такое возможно, если обе последовательности состоят из повторённого много раз одного и того же нуклеотида. В этом случае каждое сравнение будет затрагивать в среднем $O(\min(l_s, l_t))$ символов, и всего будет примерно $l_s l_t$ сравнений. Поэтому сложность будет равна $O(l_s l_t \min(l_s, l_t))$ (предполагается, что $l_k$ мало).

В лучшем случае совпадений вообще не будет, и время работы алгоритма будет равно $O(l_s + l_t)$. Такое время почти достигается для очень непохожих последовательностей.

На практике алгоритм работает за преемлемое время, если только не пытаться сравнивать последовательность саму с собой, что и так не имеет большого смысла: и так известно, что степень сходства последовательности с собой равна единице.

# Глава 3.

# Описание эксперимента и результаты

## 3.1. Описание эксперимента

Для эксперимента были выбраны пять хромосомных последовательностей бактерий: три вида *Escherichia coli* (NC_012759, NC_012947 и NC_012967), одну вида *Salmonella enterica* (NC_010067) и одну вида *Staphylococcus aureus* (NC_002952). Каждая из последовательностей сравнивалась с каждой двумя способами (первая в качестве исходной, а вторая — в качестве целевой и наоборот).

При сравнении использовались следующие параметры:

- $l_k = 10$.
- $D_f = 16$.
- $D_l = 1$.
- $S_{x,y} = \begin{cases} 0, & \text{если } x = y, \\ 8, & \text{если } x \neq y. \end{cases}$.

Параметры подбирались, исходя из соображений производительности и потребления памяти $l_k$, из соображений того, чтобы связанный фрагмент не смог оказаться оптимальным в результате случайного совпадения (поскольку длины последовательностей около $4^{10}$, отношение $\frac{D_f}{D_l}$ должно быть несколько больше десяти), а также для того чтобы не тратить время на слишком непохожие последовательности (матрица $S_{x,y}$ совместно с $D_l$ допускают не более $\frac{1}{8}$ несовпадений).

## 3.2. Результаты

В таблице приведены результаты эксперимента. Строки соответствуют исходным последовательностям, а столбцы — целевым. В таблице указана степень сходства, округлённая до целого числа процентов, за исключением случаев, в которых степень сходства меньше одного процента.

| | NC_012759 | NC_012947 | NC_012967 | NC_010067 | NC_002952 |
|---|---|---|---|---|---|
| NC_012759 | 100% | 90% | 88% | 11% | 0,1% |
| NC_012947 | 90% | 100% | 96% | 11% | 0,1% |
| NC_012967 | 89% | 97% | 100% | 11% | 0,1% |
| NC_010067 | 11% | 11% | 11% | 100% | 0,1% |
| NC_002952 | 0,1% | 0,1% | 0,1% | 0,1% | 100% |

## 3.3. Выводы

Результаты эксперимента показывают, что нуклеотидные последовательности бактерий одного вида существенно более схожи, чем последовательности бактерий различных видов. Также видно, что виды *Escherichia coli* и *Salmonella enterica* более схожи между собой, чем с видом *Staphylococcus aureus*.

Небольшие значения степени сходства объясняются тем, что алгоритм не способен определять степень различия между фрагментами, если она превосходит некоторый порог (в данном эксперименте $\frac{1}{8}$ замен). Это значит, что любые фрагменты, в которых плотность замен выше, будут считаться абсолютно различными, а для остальных фрагментов степень сходства будет нормирована, исходя из этой границы.

В то же время, это ограничение позволяет алгоритму быстрее работать.

# Заключение

В данной работе предложен новый способ сравнения нуклеотидных последовательностей. Этот способ позволяет учитывать нелокальные мутации, что отличает его от известных способов.

Также предложен алгоритм, реализующий этот способ. Было проверено, что алгоритм позволяет сравнивать реальные последовательности за преемлемое время.

Кроме того, алгоритм был проверен на существующих последовательностях и показал корреляцию между степенью сходства нуклеотидных последовательностей и тем, принадлежат ли соответствующие организмы одному и тому же виду или нет.

Возможности для дальнейшего исследования в данном направлении:

- При создании алгоритма было найдено несколько возможностей для оптимизации. Их использование потенциально позволит ускорить программу, а также сделать так, чтобы она использовала меньше памяти.
- Текущая мера различия может неправильно учитывать некоторые комбинации мутаций: например, точечная мутация, размноженная дупликацией, вносит такой же вклад в степень различия, как и две отдельные точечные мутации. Хотя пути решения этой проблемы неясны, над этим можно поработать.
- Предложенный алгоритм не позволяет распараллеливать вычисления. Можно попытаться разработать алгоритм, лишённый этого недостатка.

# Источники


[1] *Berg J. M., Tymoczko J. L., Stryer L.* DNA, RNA, and the Flow of Genetic Information // Biochemistry. — W. H. Freeman and Company, 2002.

[2] Молекулярная биология. Структура и биосинтез нуклеиновых кислот. — М.: Высшая школа, 1990.

[3] *Berg J. M., Tymoczko J. L., Stryer L.* DNA Replication, Recombination, and Repair // Biochemistry. — W. H. Freeman and Company, 2002.

[4] *Needleman S. B., Wunsch C. D.* A general method applicable to the search for similarities in the amino acid sequence of two proteins // *Journal of Molecular Biology.* — 1970. — Vol. 48, no. 3. — Pp. 443 – 453. `http://www.sciencedirect.com/science/article/B6WK7-4DN8W3K-7X/2/0d99b8007b44cca2d08a031a445276e1`.

[5] *Smith T. F., Waterman M. S.* Identification of common molecular subsequences // *Journal of Molecular Biology.* — 1981. — Vol. 147, no. 1. — Pp. 195 – 197. `http://www.sciencedirect.com/science/article/B6WK7-4DN3Y5S-24/2/b00036bf942b543981e4b5b7943b3f9a`.