ISSN 2663-1830

Қазақ технология және бизнес университеті
Казахский университет технологии и бизнеса

№ 4 (2019)

ҚазТБУ Хабаршысы

Вестник КазУТБ

Vestnik KazUTB

Нур - Султан - 2019

***Собственник**:*
АО «Казахский университет технологии и бизнеса»

Регистрация:

Министерство информации и коммуникаций Республики Казахстан,
Комитет Информации № 14139 – Ж «07» 02. 2014 г.
Выходит 4 раза в год

ISSN: 26631830

Адрес редакции: 010000, г. Нур - Султан, Есильский район,
ул. Кайыма Мухамедханова, 37 «А»
каб. 602, тел.: +7 - 7172 - 279230 (134)
*e-mail:*journal.vestnik.kazutb@mail.ru

ҚАЗАҚ ТЕХНОЛОГИЯ ЖӘНЕ БИЗНЕС УНИВЕРСИТЕТІ

КАЗАХСКИЙ УНИВЕРСИТЕТ ТЕХНОЛОГИИ И БИЗНЕСА

ҚазТБУ ХАБАРШЫСЫ
ВЕСТНИК КазУТБ
VESTNIK KazUTB

НУР - СУЛТАН - 2019

УДК 51-76: 004.7

B.R. Zholmagambetova[3], Sh.A. Jomartova[1,2], A.T. Mazakova[2], B.S. Amirkhanov[2], T.S. Shormanov[2]
([1]RSE Institute of Information and Computational Technologies MES RK CS, Almaty, Kazakhstan, [2]Al-Farabi Kazakh National University, Almaty, Kazakhstan, [3]L.N. Gumilyov Eurasian National University, Nur-Sultan, Kazakhstan, bakhytgulz@mail.ru)

DEVELOPMENT OF THE POTENTIAL COMPLEXITY OF THE BIOINFORMATICS ALGORITHM AT BIOPYTHON

Abstract. A review of bioinformatics algorithms on BioPython. The main tasks of bioinformatics and algorithms, with examples in Python, are considered: an algorithm for processing DNA sequences, searching for patterns, a multiple sequence alignment algorithm, phylogenetic analysis algorithms, Count de Bruyne. The main processing of biological sequences is shown as the implementation of processes associated with gene expression, including transcription, translation and identification of open reading frames. And also considered the ability to calculate the frequency of various characters in the sequences. The potential complexity of some algorithms is determined, which shows the importance of these algorithms and the potential problems. Hidden Markov models, graphs, and biological networks are considered.

Key words: BioPython, bioinformatics, algorithms, DNA, RNA.

Б.Р. Жолмагамбетова[3], Ш.А. Джомартова[1,2], А.Т. Мазакова[2], Б.С. Амирханов[2], Т.С. Шорманов[2]
([1]Институт информационных и вычислительных технологий КН МОН РК, Алматы, Казахстан, [2]Казахский национальный университет имени аль-Фараби, Алматы, Казахстан, [3]Евразийский национальный университет имени Л.Н. Гумилева, Нур-Султан, Казахстан, bakhytgulz@mail.ru)

РАЗРАБОТКА ПОТЕНЦИАЛЬНОЙ СЛОЖНОСТИ АЛГОРИТМА БИОИНФОРМАТИКИ НА BIOPYTHON

Аннотация. Проведен обзор по алгоритмам биоинформатики на BioPython. Рассмотрены основные задачи биоинформатики и алгоритмы, с примерами на Python: алгоритм по обработке последовательностей ДНК, поиск паттернов, алгоритм множественного выравнивания последовательностей, алгоритмы филогенетического анализа, граф де Брёйна. Показана основная обработка биологических последовательностей в качестве реализации процессов, связанных с экспрессией генов, включая транскрипцию, трансляцию и идентификацию открытых рамок считывания. А также рассмотрена возможность вычислять частоту различных

символов в последовательностях. Определена потенциальная сложность некоторых алгоритмов, что показывает важность этих алгоритмов и потенциал проблем. Рассмотрены скрытые марковские модели, графы и биологические сети.

Ключевые слова: Bio Python, биоинформатика, алгоритмы, ДНК, РНК.

Введение. Научное сообщество использует несколько языков программирования для решения задач биоинформатики. Основным для исследователей является язык программирования Python. Python используется многими компаниями, от небольших и неизвестных магазинов до крупных игроков в своих областях, таких как Google, National Geographic, Disney, NASA, NYSE и многие другие. Это один из четырех “официальных языков” Google среди Java, C++ и Go. У них есть веб-сайты, сделанные на Python, автономные программы и даже хостинг-решения. В качестве подтверждения того, что Google серьезно относится к Python, в декабре 2005 года они наняли Гвидо ван Россума, создателя Python. Возможно, это не основной язык Google, но это показывает, что Google является его решительным сторонником. Даже Microsoft разработала версию Python для своей платформы «.Net» (Iron Python), а также разработала Python Tools for Visual Studio 21, бесплатный плагин с открытым исходным кодом, который превращает Visual Studio в среду разработки Python. Многие известные дистрибутивы Linux уже используют Python в своих ключевых инструментах. Ubuntu Linux «предпочитает, чтобы сообщество вносило вклад в работу с Python». Python настолько тесно интегрирован в Linux, что некоторые дистрибутивы не будут работать без рабочей копии Python [1].

Biopython–это очень полезный пакет модулей для разработки приложений биоинформатики. Хотя каждая задача биоинформатики уникальна, все же есть некоторые задачи, которые повторяются, у которых возникает необходимость обмена между раздельными программами и стандартными форматами файлов. Такая ситуация говорит о необходимости разработки пакета мер по решению таких задач, основанных на биологических проблемах. Для решения таких задач (и не только) используется Biopython [2-6].

Biopython в качестве идеи появился в августе 1999 года – это была инициатива Джеффа Чанга и Эндрю Далка. Позже к авторам присоединилось сообщество, которое начало поддерживать и развивать проект. Среди наиболее активных разработчиков можно выделить Брэда Чепмена, Питера Кока, Мичил де Хун и Иддо Фридберг. Проект начал принимать кодовую форму в феврале 2000 года, а в июле того же года был выпущен первый релиз. Первоначальная идея заключалась в создании пакета, эквивалентного Bio Perl, который в то время был основным пакетом биоинформатики. Хотя Bio Perl, возможно, был вдохновением Biopython, концептуальные различия между языками Perl и Python дали Biopython особый путь для развития. Biopython является частью семейства open-bio проектов (также известных как Bio*), которые институционально являются членами Фонда Open Bio informatics Foundation.

Установка Biopython подробно расписана по этой ссылке http://biopython.org/wiki/Download и мы не ставим целью расписывать в рамках этой статьи вопросы установки. Просто

хотелось отметить, что пакет полностью кроссплатформенный и работает на многих платформах (Windows, Mac и различные версии Linux и Unix). Для Windows представлены предварительно скомпилированные установщики типа «нажми и работай», а для Linux можно использовать либо готовые бинарные пакеты, либо что правильно, собрать пакет биопайт из исходных кодов.

Давайте посмотрим на некоторые особенности Python, на которые стоит обратить внимание. Основные особенности Python:

– раткий и понятный синтаксис. Синтаксис не только улучшает читаемость кода, но также позволяет легко писать код, который повышает производительность программирования;

– набор высокоуровневых и мощных типов данных. Встроенные типы данных включают в себя примитивные типы, которые хранят атомарные элементы данных или типы контейнеров, которые содержат коллекции элементов (сохраняя или не упорядочивая порядок их элементов);

– наличие сторонних модулей для широкого спектра деятельности. Визуализация и отображение данных, генерация PDF, анализ биоинформатики, обработка изображений, машинное обучение, разработка игр, интерфейс с популярными базами данных и прикладное программное обеспечение, это лишь несколько примеров модулей, которые можно установить для расширения функциональности Python;

– встроенные высокоуровневые структуры данных: словари, наборы, списки, кортежи и другие. Они очень полезны для моделирования реальных данных. Сторонние модули, такие как NumPy и SciPy, также могут расширять структуры до kd-деревьев, n-мерных массивов, матричных операций, временных рядов, объектов изображений и многого другого;

– мультипарадигма: Python может использоваться как «классический» процедурный язык или как «современный» язык объектно-ориентированного программирования (ООП). Большинство программистов начинают писать код процедурным способом, а когда им нужно, они переходят на ООП. Python не заставляет программистов писать ООП-код, когда они просто хотят написать простой скрипт;

– кроссплатформенность: программу, созданную на Python, можно запускать на любом компьютере с интерпретатором Python. Таким образом, программа, созданная в Windows 10, может работать без изменений в Linux или OSX. Интерпретаторы Python доступны для большинства компьютеров и операционных систем, и даже для некоторых устройств совстроенными компьютерами, таких как Raspberry Pi;

– процветающее сообщество: Python в настоящее время является языком программирования для ученых и исследователей. Это приводит к большему количеству библиотек для ваших проектов и людей, к которым вы можете обратиться за поддержкой.

Что можно найти в пакете Biopython? Основные выпуски Biopython имеют множество функций, в том числе:

– возможность разбирать файлы биоинформатики в используемые Python структуры данных, включая следующие форматы: Blastoutput, Clustalw, FASTA, GenBank, PubMed и Medline, ExPASy файлы (Enzyme и Prosite), SCOP включая ‘dom’ и ‘lin’ файлы, UniGene, SwissProt;

– файлы в поддерживаемых форматов могут быть выбраны по записи или проиндексированы, а также доступны через интерфейс словаря;

– код для работы с популярными онлайн пораталами биоинформатики, такими как: NCBI-Blast, Entrezand Pub Medservices, ExPASy - Swiss-Prot и Prosite записи, а также Prosite поиск;

– интерфейсы к общим программам биоинформатики, таким как: автономный Blast от NCBI, программа выравнивания Clustalw, инструменты командной строки EMBOSS;

– стандартный класс, который работает с последовательностями, идентификаторами последовательностей и особенностями последовательностей;

– инструменты для выполнения общих операций над последовательностями, таких как перевод, транскрипция и вычисления веса;

– код для выполнения классификации данных с использованием k ближайших соседей, наивных байесовских методов или метод опорных векторов;

– код для работы с выравниваниями, включая стандартный способ создания и обработки матриц подстановки;

– код, упрощающий разделение распараллеливаемых задач на отдельные процессы;

– программы на основе графического интерфейса для выполнения основных операций с последовательностями, переводами, BLASTing и т. д.;

– обширная документация и помощь по использованию модулей, вики-документация, веб-сайт и списки рассылки;

– интеграция с BioSQL, СУБД для последовательностей.

Методы. Разработана методика потенциальной сложности алгоритмов, скрытых марковских моделей, графов и биологических сетей.

Результаты. *Основная обработка биологических последовательностей.* Здесь мы рассмотрим реализацию процессов, связанных с экспрессией генов, включая транскрипцию, трансляцию и идентификацию открытых рамок считывания. Так как последовательность ДНК является строкой начнем с функции, в которой подсчитываются действительные символы, а их сумма сравнивается с длиной строки, чтобы проверить правильность всех символов. Подобные функции могут быть легко построены для последовательностей РНК или белка.

```
def validate_dna(dna_seq ):
"Проверка на правильность последовательности ДНК. Возвращает True, если последовательность действиельна иначе False. "
seqm = dna_seq .upper ()
valid = seqm.count("A") + seqm.count("C") + seqm. count("G") +
seqm. count("T")
if valid == len (seqm): return True
else :return False
>>>validate_dna (" atagagagatctcg")
True
>>>validate_dna (" ATAGAXTAGAT ")
False
```

Другим полезным примером является возможность вычислять частоту различных символов в последовательностях. Результатом функции будет словарь, где ключи – это, символы, а соответствующие значения – частоты. Обратите внимание, что это может быть применено к любому из типов последовательностей, определенных

выше (ДНК, РНК и белки):

```
def frequency (seq):
"""Вычисляет частоту каждого символа в последовательности.
Возвращает словарь."""
dic = {}
for s in seq.upper ():
if s in dic: dic[s] += 1
else :dic[s] = 1
return dic
>>>frequency ("atagataactcgcatag ")82 Chapter 4
{'A': 7, 'C': 3, 'G': 3, 'T': 4}
>>>frequency (" MVVMKKSHHVLHSQSLIK ")
{'H': 3, 'I': 1, 'K': 3, 'L': 2, 'M': 2, 'Q': 1, 'S': 3, 'V': 3}
```

Также вы можете запрограммировать функции, которые

• Возвращает частоту аминокислот от самых частых до наименее частых;

• Возвращает процентное содержание G и C нуклеотидов в последовательности ДНК;

• Возвращает содержимое GC непересекающихся подпоследовательностей размера k;

• Функция, которая вычисляет РНК, соответствующую транскрипции предоставленной последовательности ДНК;

• Вычисляет обратный комплемент последовательности ДНК;

• Переводит кодон в аминокислоту, используя внутренний словарь со стандартным генетическим кодом;

Переводит последовательность ДНК в аминокислотную последовательность.

Поиск конкретных паттернов в биологических последовательностях является одной из наиболее распространенных задач, с которыми биоинформатики сталкиваются ежедневно. Поскольку соответствующая биологическая информация хранится в последовательностях, анализ локальных закономерностей в этих последовательностях является весьма актуальным. Алгоритмы данного направления – это, алгоритмы относящиеся к поиску подстроки в строке. Наиболее распространенные: это примитивный поиск (правда его не используют), алгоритм Бойера-Мура, использование детерминированного конечного автомата, алгоритм Кнута-Морриса-Пратта и многие другие. Очень часто для поиска паттернов используют регулярные выражения. Регулярные выражения – это, концепция программирования, существующая во всех современных языках программирования, которая позволяет гибко определять шаблоны для поиска в строках, что позволяет находить паттерны используя гибкие шаблоны. Пример использования регулярных выражений для проверки корректности входящей последовательности представлен ниже:

```
def validate_dna_re (seq):
from re import search
if search("[^ ACTGactg]", seq) != None:
return False
else :
return True
>>>validate_dna_re("ATAGAGACTATCCGCTAGCT")
True
>>>validate_dna_re("ATAGAGACTAXTCCGCTAGCT")
False
```

Сравнение последовательностей и выравнивание последовательностей. Одним из основных вкладов, которые биоинформатика вносит в биологические исследования, является помощь в выяснении функции данных

генов или белков, которые они кодируют. Общий подход к решению этих задач состоит в том, чтобы генерировать гипотезы относительно их биологической функции на основе сходства их последовательностей с другими, которые имеют определенную функцию, предпочтительно с экспериментальной проверкой. На самом деле мы пытаемся достичь гомологии, то есть существования общей родословной, основанной на сходстве последовательностей. Важно подчеркнуть, что эти понятия не эквивалентны. Однако на практике последовательности, которые демонстрируют высокую степень сходства, имеют высокую вероятность того, что они будут гомологичны и имеют схожие функции. Эта вероятность возрастает с увеличением степени подобия. Когда сходство достигает высоких уровней (определение правильного порога, к сожалению, не очевидно), мы можем вывести функцию с некоторой достоверностью, хотя идеалом всегда является экспериментальное подтверждение. Целевую функцию для задачи оптимизации (проблемы парного выравнивания последовательности) можно определить следующим образом:

Вход: две последовательности с четко определенным алфавитом; целевая функция для оценки каждого решения выравнивания;

Вывод: оптимальное сочетание символов в каждой последовательности, расставляя пробелы в соответствующих позициях, чтобы максимизировать целевую функцию.

Прежде чем смотреть более подробно на особенности целевой функции, давайте попробуем понять потенциальную сложность этой проблемы, оценив число возможных решений. Для этой цели мы рассмотрим упрощение, согласно которому обе последовательности имеют одинаковый размер n. Это ограничение будет служить только для упрощения наших выражений и будет достаточным, чтобы дать представление о сложности. Поскольку для нашей задачи решения различаются по способу размещения пробелов в последовательностях, мы можем легко выяснить, что максимальный размер всего выравнивания будет равен 2n, для случая, когда мы помещаем n пробелов, поскольку это не имеет смысла иметь разрыв в обеих последовательностях. Таким образом, общее количество решений будет обеспечено возможными комбинациями размера n из 2n элементов, определяемыми как:

$$\binom{2n}{n} = \frac{(2n)!}{n!^2} \qquad (1)$$

Чтобы иметь представление о представленном значении, для n = 20 (очень маленькое значение для реальных биологических последовательностей) это составляет около 130 миллиардов решений. Это дает представление о том, что проблема довольно сложна и что ее невозможно решить с помощью метода грубой силы, в котором перебираются все возможные варианты [7].

В первые годы биоинформатики различные исследователи предложили использовать алгоритмы динамического программирования (DP) для решения задач выравнивания биологической последовательности. DP – это универсальный класс алгоритмов оптимизации, основанный на принципе «разделяй и властвуй», где оптимальные решения для подзадач и их оценки используются повторно (а не

пересчитываются) при решении более крупных задач. Эта идея была применена к выравниванию последовательностей, учитывая, что выравнивание двух последовательностей может быть составлено из выравниваний подпоследовательностей этих последовательностей. Примерами использования динамического программирования являются алгоритмы: алгоритм Нидлмана-Вунша и алгоритм Смита-Уотермана. Описание построения этих алгоритмов вы можете найти в [4]. В пакете BioPython реализованы алгоритмы динамического программирования, например в следующем примере показано, как выполнить выравнивание двух последовательностей ДНК, используя оценку соответствия 1, в то время как оценка несоответствия и штрафы за пропуски равны 0:

```
from Bio import pairwise2
from Bio. pairwise2 importformat_ alignment
alignments = pairwise2 .align. globalxx(" ATAGAGAATAG ", " ATGGCAGATAGA ")
print( l e n ( alignments))160 Chapter 6
forainalignments:
print(format_alignment(*a))
```

Особо стоит отметить отдельную проблему: поиск похожих последовательностей в базах данных. За последние годы сообществом биоинформатики было предложено несколько эвристических алгоритмов для решения этой задачи. Наиболее популярными из них являются FASTA, появившаяся в первые годы биоинформатики, и BLAST (Basic Local Search Alignment Tool), который в настоящее время является широко используемой программой, доступной для запуска сообществом на большом количестве серверов, включая основные исследовательские институты как NCBI или EBI.

Выравнивание нескольких последовательностей. Отдельной задачей стоит выравнивание нескольких последовательной сразу. Если выше мы рассматривали попарное выравнивание, то сейчас задача ставится для n последовательностей сразу. Это имеет особое значение, например, при определении функции белков, где комбинация различных последовательностей с потенциальной гомологией нашей цели из разных организмов может придать дополнительную уверенность в создании гипотез для функционального описания и обеспечивает более надежный способ идентифицировать консервативные области тех белков, которые могут играть соответствующую функциональную роль. Белковые выравнивания могут также очень помочь в определении их вторичных или третичных структур.

Проблема множественного выравнивания последовательностей (MSA) обобщает попарное выравнивание, рассматривая N (N> 2) последовательностей для выравнивания. Хотя определение проблемы очень похоже, сложность возрастает с ростом числа последовательностей. Действительно, проблема считается NP-трудной с точки зрения сложности. Таким образом, нет эффективных алгоритмов для решения проблемы, когда N становится большим. Для решения задач множественного выравнивания используют методы динамического программирования с построением гиперкуба и методы эвристических алгоритмов. В целом,

эвристические алгоритмы для MSA могут быть классифицированы как:

– прогрессивный – начинается с выравнивания двух последовательностей и затем итеративно добавляются остальные последовательности к выравниванию;

– итеративный – рассмотрите начальное выравнивание и затем попытайтесь улучшить это выравнивание, перемещая, добавляя или удаляя пробелы;

– гибридный – может сочетать разные стратегии и использовать дополнительную информацию (например, информацию о структуре белка, библиотеки с хорошим локальным выравниванием).

Филогенетический анализ. Филогенетика – это раздел биологии, который изучает эволюционную историю и отношения между людьми или видами. Вывод филогений, представленных в виде филогенетических деревьев, которые объясняют эволюционные отношения между этими индивидуумами или видами, традиционно достигается на основе наследственных признаков, в основном морфологических. Наличие последовательностей ДНК позволило изменить эту область, сделав ее гораздо более строгой, поскольку она основана на прямом продукте эволюции, учитывая, что мутации происходят непосредственно над ДНК.

Алгоритмы филогенетического анализа можно в целом разделить на три основных класса, которые в основном различаются по стратегии, используемой для вычисления целевой функции, и в то же время предлагают альтернативные подходы для поиска в огромном пространстве решения этой проблемы. Основными классами являются следующие:

– алгоритмы на основе расстояний: включают в себя методы, основанные на расчете парных расстояний между последовательностями (на основе выравнивания последовательностей), с целью поиска деревьев, в которых расстояния соответствуют значениям во входной матрице;

– максимальная экономия: включает методы, которые ищут деревья, где количество необходимых мутаций (во внутренних узлах дерева) для объяснения изменчивости последовательностей минимизировано;

– статистические/байесовы методы: определяет вероятностные модели для возникновения различных типов мутаций и использует эти модели для оценки деревьев на основе их вероятности (или вероятности), поиска наиболее вероятных деревьев, которые объясняют последовательности в соответствии с предполагаемой моделью.

Алгоритмы обнаружения мотива. В контексте анализа биологической последовательности термин «мотив» относится к нетривиальному последовательному шаблону, который характерен для нескольких последовательностей. Под нетривиальным мы подразумеваем, что мотив имеет минимальную соответствующую длину и захватывает комбинацию символов, которая отличается от базового распределения символов. Однако его главной особенностью является повторение, то есть тот факт, что это происходит в нескольких из проанализированных последовательностей. Проблема обнаружения детерминированных мотивов представляет собой

интересную вычислительную задачу, которая привлекала внимание многих биоинформатиков и ученых на протяжении многих лет. Алгоритмы для решения этой задачи:

– алгоритмы грубой силы: исчерпывающий поиск;

– алгоритм ветвления и границ. Алгоритм принадлежит группе комбинаторных алгоритмов оптимизации. Он перечисляет кандидатов, но использует интеллектуальный механизм прогнозирования, чтобы избежать явного перечисления некоторых решений. Поэтому он хорошо подходит для этой задачи;

– эвристические алгоритмы. Эвристические алгоритмы были введены в различных методах, с компромиссом между оптимальностью решения и вычислительной эффективностью основанные на предыдущих алгоритмах.

Вероятностные мотивы и стохастические алгоритмы. Вероятностные мотивы, как правило, выражаются через позиционную весовую матрицу (PWM), также называемые шаблонами или профилями. PWM – это матрица взвешенных совпадений каждого из биологических символов (нуклеотидов или аминокислот) в виде строк и каждой позиции мотива в виде столбцов. PWM обеспечивает вероятностное представление мотива путем захвата частоты каждого символа вдоль его положений последовательности. PWM может затем использоваться для поиска новых совпадений мотивов во входных последовательностях и может быть уточнено путем включения других совпадений с высокими показателями. Стохастические алгоритмы, основанные на максимизации ожиданий (EM), могут решить эту проблему эффективного поиска мотива, предоставляя способ обхода пространства поиска и оптимизации мотивов [8]. Эта стратегия опирается на итеративный подход, который может одновременно определять наиболее представительные подпоследовательности, т.е. лучшие совпадения мотива по входным последовательностям при обновлении и уточнении модели мотива. Предыдущий алгоритм можно улучшить, введя стратегию под названием «выборка Гиббса». Эта идея, предложенная Lawrence [11], используется несколькими алгоритмами поиска мотивов и в качестве эвристического метода не гарантирует наилучшего решения, но на практике при многократном запуске он работает хорошо.

Скрытые марковские модели (HMM). В анализе биологической последовательности события, связанные с последовательностью, часто являются вероятностными. Примеры этого включают, например, классификацию субклеточной локализации (например, цитозоль, ядро или мембрана) белка по его последовательности; наличие белковых доменов в последовательности или вероятность связывания транскрипционного фактора с последовательностью области промотора гена. Скрытые марковские модели – стохастические модели, которые фиксируют статистические закономерности из входных последовательностей и позволяют разрабатывать алгоритмы для поиска мотивов и поиска в базе данных [9]. HMM – вероятностные модели, которые фиксируют статистические закономерности в линейной последовательности. Действительно,

HMM широко использовались во многих различных задачах молекулярной биологии, включая поиск генов, поиск профилей, множественное выравнивание последовательностей, идентификацию регуляторных сайтов или предсказание вторичной структуры в белках.

Общее введение в скрытые марковские модели можно найти в [12]. HMM очень хорошо подходят для многих задач при анализе биологических последовательностей. Это включает в себя задачи поиска генов, поиска профиля, выравнивания нескольких последовательностей, идентификации сайта регулятора и прогнозирования вторичной структуры или определения числа копий. В поиске генов стоит выделить методы HMMgene и GENSCAN.

Графы и биологические сети. Граф может быть определен в математике как набор объектов, с которыми связаны некоторые пары объектов в этом наборе. Хотя они могут быть легко определены и имеют простую структуру, они представляют собой мощные и гибкие структуры данных с огромным набором приложений в области компьютерных наук и многих областях науки и техники.

В последние годы различные типы представлений, основанные на концепции графов использовались для характеристики и моделирования различных типов биологических систем, в основном внутри клеток. Эти графы, глобально обозначенные как биологические сети, обычно представляют в своих узлах биологические объекты, такие как гены, белки или химические соединения, в то время как ребра представляют различные типы взаимодействий или отношений между этими объектами с четко определенным биологическим значением (например гены – кодирующие белки, метаболиты – участвующие в реакциях, белки – которые регулируют экспрессию генов или связываются с соединениями). Эти сети предоставляют мощные инструменты в области системной биологии для поддержки глобального анализа клеточной организации и фенотипического поведения, а также их основных подсистем с акцентом на метаболизм, регуляцию экспрессии генов и передачу сигнала. С другой стороны, регуляторные сети охватывают гены и белки, представляющие процессы, которые позволяют клеткам контролировать уровни экспрессии генов в различных ситуациях (например, разные типы клеток или разные стимулы), в то время как сигнальные сети представляют собой белковые каскады, обычно участвующие в трансдукции сигналов снаружи клетки во внутренние регуляторные и метаболические системы. Для анализа биологических сетей используются: степень распределения, поиск кратчайшего пути, коэффициенты кластеризации и много другое почерпнутого из теории графов.

Сборка генома – процесс объединения большого количества коротких фрагментов ДНК (ридов) в одну или несколько длинных последовательностей в целях восстановления последовательностей ДНК хромосом, из которых возникли эти фрагменты в процессе секвенирования.

Секвенирование генома много раз сравнивалось с чтением книги, книги жизни. Тем не менее, секвенирование генома технически совсем не процесс чтения. Действительно, в настоящее

время нет технологий, позволяющих секвенировать полные геномы, поскольку эти методы ограничены последовательностью небольших фрагментов молекул ДНК, состоящей не более чем из нескольких сотен пар оснований. Эти перекрывающиеся фрагменты, называемые ридами, необходимо затем собрать вместе, чтобы восстановить исходную последовательность (или геном), почти так же, как можно собрать кусочки головоломки.

Сложность этой проблемы может быть огромной, что легко понять, если учесть, что размер считываний обычно составляет несколько сотен нуклеотидов, в то время как полные геномы обычно содержат от нескольких до тысяч миллионов нуклеотидов. До 1990-х годов это действительно считалось невыполнимой задачей, и даже сегодня некоторые более крупные геномы все еще недоступны для современных программно-аппаратных комплексов.

Помимо размера геномов, другие факторы делают это очень сложной проблемой на практике. Некоторые из них перечислены ниже:

– молекулы ДНК имеют две комплементарные цепи, и невозможно определить, из какой цепи был взят данный фрагмент;

– все типы оборудования для секвенирования генома имеют относительно высокую частоту ошибок, и, таким образом, во многих случаях фрагменты могут отличаться от исходной последовательности, что затрудняет вывод наложения с другими смежными фрагментами;

– существуют области генома, которые очень трудно или даже невозможно упорядочить, и поэтому они не охватываются никакими ридами.

Наиболее эффективные алгоритмы для этих сложных задач основаны на графах, если точнее на концепциях гамильтоновых и эйлеровых путей, и связанных с ними алгоритмах.

Графы перекрытия. Одним из вариантов решения этих проблем является теория графов. Действительно, фрагменты для этой последовательности могут быть организованы в особый тип графа, называемый графом перекрытия, который может привести к другой постановке задачи и альтернативным алгоритмам. Граф перекрытия определяется следующим образом:

– узлы (вершины) соответствуют входным фрагментам, то есть для каждого фрагмента будет одна вершина (рид);

– ребра созданы между двумя узлами v и w, если $\text{suffix}_{k-1}(v) = \text{prefix}_{k-1}(w)$, т.е. последовательностей узлов v и w имеют перекрытие K - 1 нуклеотидов.

Решением задачи восстановления последовательности можно представить в виде пути по этому графу. Чтобы решение было правильным, этот путь должен проходить ровно один раз по каждому узлу графа.

Гамильтонов цикл. В ориентированном графе гамильтонов цикл – это путь, который проходит через все узлы графа ровно один раз. Как мы уже упоминали выше, проблема восстановления строки последовательности с учетом ее состава может быть определена как проблема нахождения гамильтоновой схемы в графе перекрытия. Действительно, из пути, заданного гамильтоновым циклом, мы можем легко получить «исходную» последовательность. Это делается простым извлечением

последовательности из первого узла в пути и конкатенацией последнего символа последовательностей в оставшихся узлах пути в соответствии с его порядком. При увеличении размера последовательности алгоритм начинает вести себя все хуже и хуже с точки зрения времени выполнения. Фактически этот алгоритм нельзя использовать для больших последовательностей и соответствующих графиков перекрытия. Действительно, проблема нахождения гамильтоновых графов довольно сложна и относится к классу NP-полных задач оптимизации, что подразумевает отсутствие эффективных алгоритмов при масштабировании задачи, в данном случае, когда граф становится больше [10].

Графы де Брёйна для сборки генома. В поиске эффективных алгоритмов для сборки генома исследовательское сообщество искало другие пути решения. Действительно, подходя к аналогичным проблемам, хотя в то время не относящийся к сборке генома, математик де Брёйн предложил отличное решение, в котором также используются ориентированные графы. Идея заключается в том, чтобы представлять фрагменты задачи как ребра, а не как вершины. Таким образом, здесь решениями этой задачи могут быть пути по графу, содержащие все ребра ровно один раз. В случае графов *де Брёйна* вершины содержат последовательности, которые соответствуют либо $prefix_{k-1}$, либо $suffix_{k-1}$ одного из фрагментов. Таким образом, каждое ребро, соответствующее фрагменту, соединяет узел, представляющий его $prefix_{k-1}$, с узлом, представляющим его $suffix_{k-1}$. В этом случае, в отличие от предыдущих графиков перекрытия, префиксы и суффиксы длины k-1, которые имеют одинаковую последовательность, представлены в одном узле. Таким образом, повторяющиеся фрагменты представлены в виде нескольких ребер, соединяющих одну и ту же пару узлов.

Обсуждение. Мы рассмотрели в данной статье основные алгоритмы биоинформатики: поиск паттернов в цепочке ДНК, сравнение и выравнивание последовательностей, алгоритм множественного выравнивания последовательностей, алгоритмы филогенетического анализа, алгоритмы обнаружения мотива, а также вероятностные мотивы и стохастические алгоритмы. Определили потенциальную сложность некоторых алгоритмов, что показывает важность этих алгоритмов и потенциал проблем, для решения которых используются эти алгоритмы. Рассмотрели скрытые марковские модели, графы и биологические сети. Для решения вопросов сборки генома рассмотрели графы перекрытия, гамильтонов цикл и графы де Брёйна. Понимая идеи и алгоритмы, лежащие в основе различных методов, можно оптимизировать их использование или разработать новые методы, отвечающие потребностям современной науки.

Работа выполнена за счет средств грантового финансирования научных исследований на 2018-2020 годы по проекту AP05132044 «Разработка аппаратно-медицинского комплекса оценки психофизиологических параметров человека».

ЛИТЕРАТУРА

1. S.L. Salzberg, D.B. Searls, S. Kasif. Computational Methods in Molecular Biology. – Elsevier Science, 1998. – 368 c.

2. Cock P.J., Antao T., Chang J.T., Chapman B.A., Cox C.J., Dalke A., Friedberg I.,Hamelryck T., Kauff F., Wilczynski B., de Hoon M.J. Biopython: freely available python tools for computational molecular biology and bioinformatics. – Bioinformatics, 2009.

3. Jeff Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck, Michiel de Hoon, Peter Cock, Tiago Antao, Eric Talevich, Bartek Wilczyński. Biopython Tutorial and Cookbook.http://biopython.org/DIST/docs/tutorial/Tutorial.html.

4. Miguel Rocha, Pedro G. Ferreira. Bioinformatics Algorithms. Design and Implementation in Python. – Academic Press is an imprint of Elsevier,2018. – 400 p.

5. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter. Molecular Biology of the Cell, 4th edition.– Garland Science, New York, USA, 2002.

6. S. Bassi, Python for bioinformatics,2th edition. – CHAPMAN & HALL/ CRC,Mathematical and Computational Biology Series

7. Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, David J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs, Nucleic Acids Research,1997.

8. R. Andersson, et al. An atlas of active enhancers across human cell types and tissues, Nature 507 (Mar 2014).–P.455-461.

9. K. Asai, S. Hayamizu, K. Handa, Prediction of protein secondary structure by the hidden Markov model, Computer Applications in the Biosciences 9 (2) (Apr 1993) – P. 141-146.

10. John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd edition, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

11. C.E. Lawrence, S.F. Altschul, M.S. Boguski, J.S. Liu, A.F. Neuwald, J.C. Wootton, Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment, Science 262 (5131) (Oct 1993) – P. 208–214.

12. Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition//Proceedings of the IEEE, 1989.-P. 257–286.

Экономикалық ғылымдар

Мерейтой күні

СОДЕРЖАНИЕ

Технические науки

Редактор: М.К. Оспанова
Верстка на компьютере: Молдажанова И.К.
Сдано в набор 26.12.2019 г. Подписано в печать 30.12.2019 г.
Офисная бумага 80 г/м2. Печать цифровая.
Тираж 300 экз.
Отпечатано в типографии «Филин»
e-mail: filin_ip@mail.ru, тел.: +7 (7172) 792 777
ул. Кунаева 8, «Изумрудный квартал»