Санкт-Петербургский государственный университет

Прикладная математика и информатика

Нелинейная динамика, информатика и управление

Долотов Виктор Юрьевич

Поиск архетипа в выборках похожих строк с визуализацией средствами Python

Выпускная квалификационная работа

Научный руководитель:
д.т.н., проф. Кознов Д. В.

Рецензент:
ст. преп. Луцив Д. В.

Санкт-Петербург

2018

Saint Petersburg State University

Applied Mathematics and Computer Science

Nonlinear Dynamics, Informatics and Control

Dolotov Viktor

Finding an archetype in samples of similar lines with visualization by means of Python

Graduation Project

Scientific supervisor:
prof. Dmitry Koznov

Reviewer:
senior lecturer Dmitry Luciv

Saint Petersburg

2018

Оглавление

Введение

Информационные технологии уже давно являются неотъемлемой составляющей в жизни современного человека. Использование компьютеров, мобильных телефонов и различных программных продуктов стало незаменимым и даже обыденным. При этом сложность программного обеспечения (ПО) для этих систем постоянно возрастает. Одной из существенных компонент современного ПО является проектная документация, описывающая программные продукты с разных точек зрения и на разном уровне детализации. В связи с этим появляется задача поддержания должного качества процесса разработки и сопровождения документации.

Существует два вида документации ПО: техническая (проектная) и пользовательская (всевозможные руководства по использованию ПО). В данной работе мы рассматриваем техническую документацию. Она помогает разработчикам ПО понимать созданные ими или другими разработчиками продукты, лучше в них разбираться, а также проводить их дальнейшее усовершенствование. При этом техническая документация из-за сложности описываемого продукта может иметь значительные размеры и нетривиальную структуру.

Документация, подобно самому программному обеспечению, также постоянно совершенствуется и модернизируется. Поэтому одной из причин проблем с качеством документации является наличие неконтролируемых повторов текста, что значительно усложняет сопровождение документа, так как при модификации документации новую информацию нужно вводить не в одно место, а сразу в несколько [1]. И это очень часто не делается должным образом из-за недостатка времени и технических средств. В результате в документации накапливаются ошибки и противоречия. Для примера можно

указать руководство программиста для ядра ОС Linux (Linux Kernel Documentation), качество которого достаточно часто обсуждается в Linux-сообществе [2, 3]. Таким образом, важной задачей становится упрощение и частичная автоматизация процесса поиска и рефакторинга повторов в документации.

Одной из проблем выявления повторов является то, что в большинстве случаев они не являются точными. Раскопированные фрагменты текста обычно впоследствии изменяются — как содержательно, так и стилистически, — и такие повторы уже не найти обычными средствами текстового поиска. И если поиску и анализу точных повторов посвящено значительное количество исследований [4, 5, 6, 7], то неточные повторы исследуются крайне мало. Одним из немногочисленных проектов, посвященных этому вопросу, является проект DocLine [8, 9, 10, 11, 12, 13, 14].

Проект DocLine [15, 16, 17], созданный на кафедре системного программирования более 10 лет назад, нацелен на создание инструмента для разработки и сопровождения электронной документации со значительными повторами в тексте, а также для поддержки повторного использования документации. В рамках проекта DocLine были созданы средства поиска нечётких повторов (алгоритмы, методика, метод улучшения документации на основе поиска и анализа неточных повторов) [5, 6, 12, 14]. Также был реализован программный инструмент Duplicate Finder [8, 9, 10, 11] (исходные коды и примеры выложены в свободный доступ [18]). В том числе в инструменте имеется частичная поддержка функции рефакторинга документов: пользователь может отметить нужную ему группу неточных повторов для автоматического создания шаблона этой группы. Этот шаблон включает в себя общий текст всех повторов (архетип) с так называемыми *точками расширения* — вариативными фрагментами текста, разными для

каждого отдельного элемента группы. Для реализации данной функции появляется потребность в создании алгоритма выделения архетипа из имеющейся, то есть найденной, группы неточных повторов и визуализации данной информации. Таким образом, Duplicate Finder требует дальнейшего совершенствования и доработки.

Постановка задачи

Целью данной работы является разработка и реализация алгоритма поиска архетипа набора текстовых строк и визуализация результатов работы этого алгоритма с помощью средств Python в рамках инструмента Duplicate Finder. Для достижения этой цели были сформулированы следующие задачи.

1. Проанализировать текстовые повторы в документации программного обеспечения (ПО), обнаруженные при помощи инструмента Duplicate Finder.
2. Разработать и реализовать алгоритм выделения архетипа в выборках похожих строк на основе существующих алгоритмов выравнивания последовательностей.
3. Провести апробацию реализованного алгоритма на реалистичных данных.

1. Обзор

1.1. Средства нечёткого поиска

1.1.1. Технология DocLine

DocLine — это технология, позволяющая создавать и сопровождать сложную документацию в программных проектах, с возможностью поиска, конфигурирования и дальнейшего комбинированного использования вариативных повторов фрагментов текста [15, 16, 17]. Для представления документации DocLine расширяет язык DocBook, изначально предназначенный для создания объёмной технической документации со сложной структурой, до языка DRL, позволяющего определять повторно используемые фрагменты текста.

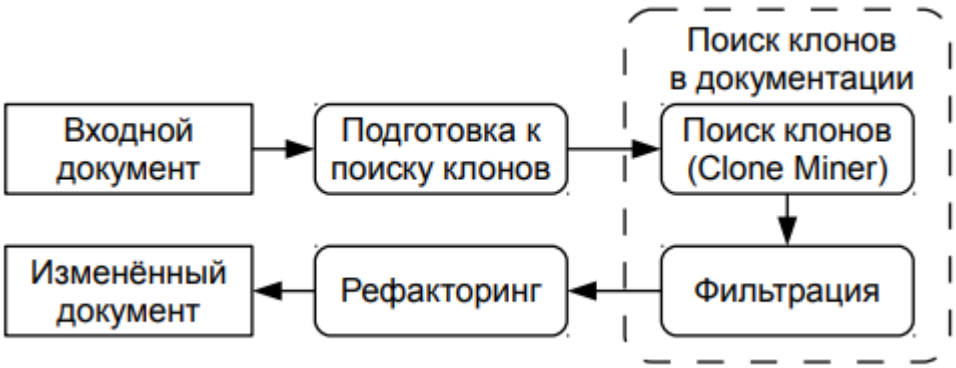

Рис. 1.1. Поиск клонов в DocLine (Рисунок взят из работы [10])

В DocLine для выполнения этих целей используется имеющий графический интерфейс инструмент Duplicate Finder [18]. На Рис. 1.1 представлен алгоритм работы с документацией: выбор файла для редактирования, его подготовка, поиск повторов, фильтрация результатов, и далее пользователь может выполнить автоматический рефакторинг любой из найденных групп.

Этот алгоритм позволяет искать в тексте неточные повторы, выделяя группы с помощью Duplicate Finder. Он, в свою очередь, использует клоны, найденные инструментом Clone Miner [19], компонируя из них неточные повторы.

1.1.2. Инструмент Duplicate Finder

Как было сказано выше, поиск неточных повторов выполняется именно с помощью Duplicate Finder [8, 9, 10, 11], и выделение архетипа в группах неточных повторов поможет в совершенствовании этого инструмента, поэтому стоит остановиться на нем подробнее.

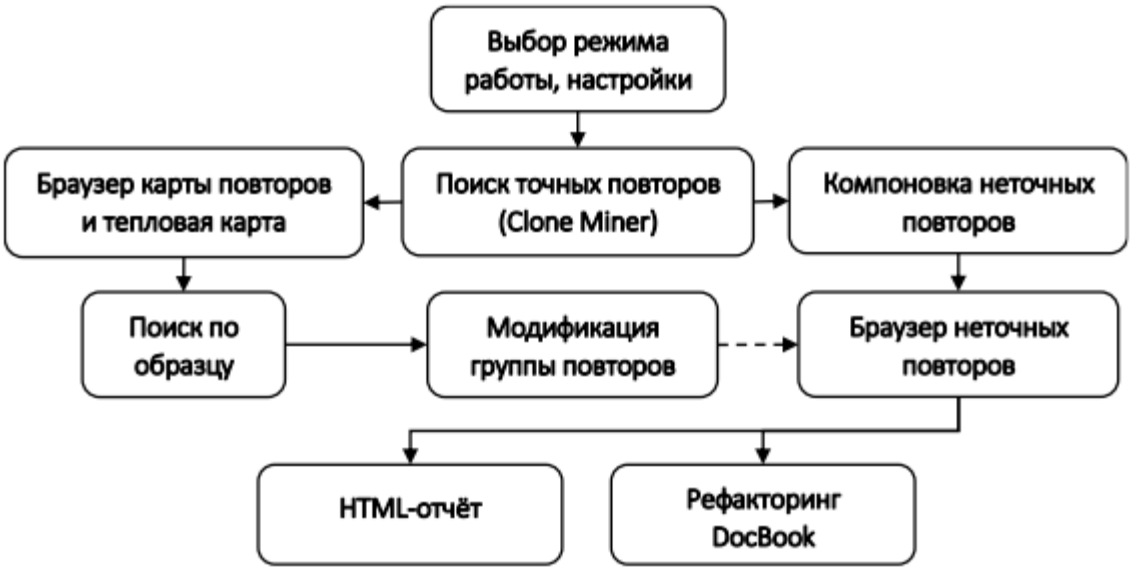

Рис. 1.2. Функциональная архитектура Duplicate Finder
(Рисунок взят из работы [11])

На Рис. 1.2 представлена функциональная архитектура Duplicate Finder: прямоугольниками представлены блоки функциональности инструмента, а стрелки задают основной сценарий использования.

Duplicate Finder поддерживает два режима работы: автоматический и интерактивный, каждый из которых имеет свои индивидуальные настройки. Автоматический режим предназначен для быстрого нахождения повторов, использующегося для предварительного анализа документа. Интерактивный режим подразумевает взаимодействие с пользователем: он формирует образец для поиска неточных повторов и редактирует полученную выдачу. На данный момент идет модификация данного алгоритма поиска, который составляет группу неточных повторов по указанному пользователем образцу. После нахождения группы результат нужно представить каким-то удобным для пользователя образом. Именно для этого нужно выделение архетипа и его визуализация в уже реализованном в автоматическом режиме браузере неточных повторов, который предоставляет наиболее удобный вариант использования инструмента, так как включает в себя всю нужную пользователю информацию о найденных группах неточных повторов.

1.2. Алгоритмы выравнивания последовательностей

После изучения области применения выделения архетипа из нескольких фрагментов текста естественно рассмотреть различные существующие алгоритмы, решающие эту или схожую задачу.

Рассмотрим применяемый в биоинформатике метод изучения *последовательностей* мономеров ДНК, РНК или белков [20], представимых в виде строки символов, каждый из которых соответствует определённому мономеру. Метод основан на размещении последовательностей друг под другом таким образом, чтобы легко увидеть похожие участки в них. Этот метод называется *выравниванием последовательностей* [21], и его результатом является структура с визуально выделенной общей составляющей, что похоже на поиск архетипа у фрагментов текста.

Рассмотрим ряд известных алгоритмов выравнивания — Нидлмана-Вунша [22] и Смита-Ватермана [23], а также алгоритм сопоставления шаблонов Ратклиффа-Обершельпа [24].

1.2.1. Алгоритм Нидлмана-Вунша

В 1970 году Солом Нидлманом (Saul Needleman) и Кристианом Вуншем (Christian Wunsch) был предложен алгоритм для выравнивания двух аминокислотных или нуклеотидных последовательностей [22].

Алгоритм состоит из двух этапов:

1) составление матрицы N×M (где N и M — это длины последовательностей);
2) обратное прохождение матрицы от нижнего правого края к верхнему левому.

Составление матрицы (оценочной матрицы) происходит путём выбора наилучшего из трёх действий: пропустить элемент первой последовательности, пропустить элемент второй последовательности или записать оба элемента, если они совпадают. Пропуск элементов влечёт за собой некоторый числовой штраф, а запись — поощрение. Таким образом, наилучшее действие выбирается как максимум из 3 возможных чисел, полученных применением штрафов или поощрений к полученным на прошлом шаге числам. Результат (оценка) записывается в соответствующий элемент матрицы. Обратное же прохождение воспроизводит те действия, что были признаны лучшими при составлении матрицы, и применяет их, получая итоговое выравнивание.

Из недостатков алгоритма можно выделить то, что в памяти нужно хранить матрицу размера N×M. Для решения этой проблемы можно воспользоваться алгоритмом Хиршберга (Daniel Hirschberg) [25], который

позволяет вычислять оптимальное выравнивание, используя меньше памяти. Однако для этого понадобится примерно вдвое большее время, поэтому конкретно в решении поставленной задачи лучше воспользоваться исходной версией алгоритма.

1.2.2. Алгоритм Смита-Ватермана

Алгоритм был впервые предложен Темплом Смитом (Temple Smith) и Майклом Ватерманом (Michael Waterman) в 1981 году [23] и является вариацией алгоритма Нидлмана-Вунша. Его исходная версия оказалась слишком требовательной, и поэтому в период с 1982 г. по 1988 г. для оптимизации алгоритма были предложены три модификации [26, 27, 28]. В последствии его вычислительная сложность стала линейной относительно длины самой короткой последовательности.

Опишем нововведения данного алгоритма. Во-первых, при инициализации матрицы в алгоритме Нидлмана-Вунша первая строка и столбец подлежат штрафу за пропуски, а в алгоритме Смита-Ватермана они заполняются нулями. Во-вторых, при дальнейшем составлении матрицы в алгоритме Смита-Ватермана не может быть отрицательных значений — вместо них ставятся нули. В-третьих, отличается обратное прохождение полученной матрицы. В алгоритме Смита-Ватермана оно начинается с наивысшего значения и заканчивается на нуле, тем самым выделяя итоговую подпоследовательность.

Однако важно отметить, что алгоритмы Нидлмана-Вунша и Смита-Ватермана отличаются также областью выравнивания — глобальной и локальной соответственно. *Глобальное* выравнивание — это использование последовательностей целиком, то есть в результате будут присутствовать все мономеры каждой последовательности. *Локальное* выравнивание

подразумевает выбор участка в каждой из последовательностей и выравнивание между этими локальными участками. Поэтому алгоритм Смита-Ватермана не подходит для решения поставленной задачи, так как архетип нужно выделять во всем неточном повторе целиком.

1.2.3. Алгоритм Ратклиффа-Обершельпа

Также в 1983 году Ратклиффом (John Ratcliff) и Обершельпом (John Obershelp) был опубликован алгоритм [24]. Он не относится к биоинформатике и выравниванию последовательностей, однако этот алгоритм можно использовать для решения поставленной задачи, более того, на языке Python уже существует его программная реализация (модуль difflib, класс SquenceMatcher [29]).

Алгоритм Ратклиффа-Обершельпа позволяет определять похожесть двух строк с помощью некоторой величины, которая получается при делении удвоенной длины архетипа на сумму длин этих строк. То есть в ходе работы алгоритма выделяется архетип двух строк. Это происходит по следующему принципу: совершается поиск максимальной смежной подпоследовательности двух строк, после чего алгоритм запускается рекурсивно для оставшихся строк, распложенных слева и справа от найденной.

Таким образом, есть возможность использовать уже существующую программную реализацию рассмотренного алгоритма, решающего задачу поиска архетипа двух строк. Однако вычислительная сложность алгоритма Ратклиффа-Обершельпа гораздо выше, чем у алгоритма Нидлмана-Вунша, так как поиск только первой наибольшей подпоследовательности уже имеет сложность, как у всего алгоритма выравнивания. Поэтому для решения задачи выделения архетипа у группы неточных повторов наиболее подходящим вариантом является использование модифицированного под работу со

строками алгоритма выравнивания двух последовательностей Нидлмана-Вунша.

1.3. Используемые технологии и программные средства

Для работы был выбран популярный в научном сообществе при решении задач анализа и обработки текста язык Python [30], так как на нём реализован инструмент Duplicate Finder, а также потому что этот язык имеет ряд достоинств, таких как кроссплатформенность, множество готовых программных библиотек, реализующих различные алгоритмы, открытая лицензия и легкость в освоении.

В качестве интегрированной среды разработки была выбрана среда PyCharm от компании JetBrains [31], потому что она позволяет удобно и быстро разрабатывать на Python, предоставляя средства для анализа кода, графический отладчик и инструмент для запуска юнит-тестов.

2. Специфика повторов, находимых Duplicate Finder

Как было отмечено выше, решение поставленной задачи поможет в дальнейшем совершенствовании инструмента Duplicate Finder, а именно, в автоматическом выделении архетипа групп повторов, находимых по образцу и при помощи алгоритма автоматического поиска повторов, самостоятельно не выделяющего архетип. Так как выделение архетипа происходит у групп неточных повторов, будет полезным узнать о них подробнее, рассмотрев статистику, собранную с использованием инструмента Duplicate Finder.

Для сбора статистики были использованы 19 документов, каждый из которых состоит в среднем из 889103 символов (Тестовый набор, описанный в [11]). Всего в них было найдено 12307 групп повторов, что в среднем составляет 648 групп на документ, большинство из которых (74,4%) являются группами точных повторов. Таким образом, в среднем лишь четверть всех групп состоит из групп с неточными повторами.

Далее подробней рассмотрим сами группы повторов. Общее число повторов из всех найденных групп оказалось равным 31166, то есть в среднем 1640 повторов в документе или по 2–3 (2,53) повтора в группе. Однако в каждой группе точных повторов содержится только по 1 уникальному повтору. Взяв во внимание информацию о том, что группы неточных повторов не всегда состоят только из уникальных повторов, и о количестве групп точных повторов, уникальными оказалась лишь половина всех повторов. Таким образом, получается в среднем по 1–2 (1,285) уникальных повтора в группе.

Также для каждой группы был выделен архетип и получена информация о точках расширения. Формально, в архетипах оказалось в среднем по 0,4 точки расширения в каждом, однако нужно учесть то, что в группах, состоящих из точных повторов, архетип совпадает со всеми повторами, то есть точки

расширения в таких группах отсутствуют. Таким образом, в архетипах групп с неточными повторами имеется в среднем 1–2 (1,545) точки расширения, а максимальное количество точек расширения, найденных в одной группе, составило 24. Отсюда можно сделать вывод, что в большинстве случаев в рамках одной группы у повторов, чья длина в среднем составляет 126 символов, отличается небольшое количество фрагментов.

Итак, в результате сбора и рассмотрения статистики было сделано несколько выводов о группах неточных повторов, находимых инструментом Duplicate Finder.

1. Лишь около четверти всех найденных групп являются неточными.
2. Неточные группы состоят, в среднем, из небольшого количества уникальных повторов (1–2).
3. Повторы в рамках одной неточной группы в большинстве случаев имеют небольшое количество точек расширения.

3. Алгоритм поиска и визуализации архетипа

Теперь, после изучения специфики находимых неточных повторов, опишем созданный в рамках работы алгоритм поиска архетипа в выборках похожих строк. Он основывается на парном выравнивании с учетом выводов, сделанных в прошлой главе. Алгоритм ищет по множеству неточных повторов, входящих в одну группу (упорядоченное множество строк G), архетип (упорядоченное множество строк A). В качестве алгоритма парного выравнивания был выбран алгоритм Нидлмана-Вунша.

3.1. Описание алгоритма

Основная идея алгоритма — применить парное выравнивание для первых элементов множества, далее последовательно результат прошлого выравнивания выравнивать со следующим элементом. Спецификация алгоритма представлена на листинге 1.

```
/* G — Входные данные
/* n — Количество повторов (#G)
/* S — Множество перестановок размера n
/* {p ϵ S | ∀ i > m p^(i+1)>p^i} = S_m ⊂ S
/* G_p — Перестановка упорядоченной группы повторов (p ϵ S)
/* A — Результат
1       Initiate ()
2       G ← {g ϵ G | ∀ i,j g_i≠g_j}
3       M ← sup {m | #S_m≤120}
4       A ← ∅
5       for p ϵ S_M
6          A_p ← pairwise_alignment( G_p^1, G_p^2 )
7          for i ϵ {3,…,n}
8                A_p ← pairwise_alignment( A_p, G_p^i )
9          if  #A < #A_p
10               then  A ← A_p
```

Листинг 1. Алгоритм поиска архетипа

В функцию Initiate() входит преобразование в нужный вид исходных данных для упорядоченного множества G, получаемых при помощи инструмента Duplicate Finder (строка 1). Далее множество G фильтруется для исключения из него точных повторов — это делается для оптимизации работы алгоритма, поскольку точные повторы одинаково влияют на итоговый архетип (строка 2). Затем подбирается такое максимальное число M, для которого множество перестановок S_M первых M элементов упорядоченного множества G не является слишком большим (строка 3). Получение перестановок нужно из-за последовательного применения парного выравнивания к элементам множества G, потому что в таком случае итоговый результат (архетип) будет зависеть от того, в каком порядке выравнивались элементы. А число M подбирается для оптимизации — после изучения специфики повторов, находимых Duplicate Finder, был сделан вывод о том, что группы неточных повторов в большинстве случаев состоят из небольшого количества уникальных повторов. Поэтому чаще всего будут перебираться все возможные перестановки, а ограничение M вводится для оставшихся случаев, однако и этого будет хватать для достаточно точного нахождения архетипа.

Основная часть алгоритма является циклом, в котором происходит выделение архетипа для строк из множества G_p, полученного из G перестановкой первых M элементов (строки 5–10). Цикл повторяется столько раз, сколько есть перестановок для данного M. Внутри этого цикла алгоритм парного выравнивания, функция pairwise_alignment(duplicate_a, duplicate_b), применяется для первых двух элементов множества G_p (строка 6), выделяя их парный архетип A_p. Далее алгоритм перебирает оставшиеся элементы из G_p, и для каждого из них последовательно применяет парное выравнивание с найденным ранее архетипом A_p (строка 8). Результат последнего выравнивания становится новым архетипом A_p. Таким образом, первые два элемента

практически задают архетип группы, который в дальнейшем проходит фильтрацию оставшимися элементами. В самом конце из всех архетипов A_p, найденных для конкретных перестановок исходного множества G, в качестве итогового результата выделяется самый больший по мощности (строка 10).

3.2. Реализация алгоритма

Описанный выше алгоритм был реализован на языке Python в виде библиотеки с двумя функциями: функцией выделения архетипа и функцией парного выравнивания.

3.2.1. Функция выделения архетипа

Функция **`archetype_search(group)`** является основной, именно внутри неё происходит выделение архетипа из группы повторов по вышеописанной схеме. На вход она получает группу повторов — список строк group, а на выходе выдаёт список списков строк (архетип). Сама функция состоит из пяти блоков (см. рис. 3.1): форматирование повторов, исключение точных групп и повторов, получение перестановок, поиск архетипа и преобразование результата в нужный вид.

В первом блоке все повторы форматируются для дальнейшего использования (Листинг 1, строка 1) — все разделительные символы заменяются пробелами, после чего лишние пробелы удаляются. Происходит преобразование текста к нижнему регистру, а также разделение слов и небуквенных символов.

Во втором блоке преобразованная группа проходит фильтрацию (Листинг 1, строка 2), после которой остаются только уникальные повторы. Если остаётся всего один повтор, то группа является точной. То есть архетипом

является весь оставшийся повтор, поэтому сразу же формируется и выдаётся результат.

```
5   def archetype_search(group):
6       # Форматирование повторов
7       work_group = group
8       for i in range(len(work_group)):...
12      #
13
14      # Исключение точных групп и повторов
15      work_group = set(work_group)
16      num_dup = len(work_group)
17      if num_dup == 1:
18          return [[dup] for dup in group]
19      #
20
21      # Получение перестановок
22      max_num_perms = max(120, num_dup + 1)
23      M = 0
24      num_perms = 1
25      while M <= num_dup and num_perms < max_num_perms:...
28      M -= 1
29      work_group = [dup.split(' ') for dup in work_group]
30      perms = list(itertools.permutations(work_group, M))
31      #
32
33      # Поиск архетипа
34      archetype = []
35      for perm in perms:...
78      #
79
80      # Преобразование результата
81      result = [[] * num_dup]
82      start_index = [0 * num_dup]
83      end_index = [0 * num_dup]
84      for word in archetype:...
94      #
95      return result
```

Рис. 3.1. Функция выделения архетипа из группы неточных повторов

В третьем блоке создается множество перестановок исходной группы (Листинг 1, строка 3). Сперва по заданному ограничению размера множества

перестановок подбирается оптимальное число M, после чего это множество создается с помощью библиотеки itertools [32].

Блок «Поиск архетипа» в точности реализует логику, представленную выше, в разделе с описанием алгоритма (Листинг 1, строки 4–10). Результатом является максимальный упорядоченный набор строк, содержащихся в одном и том же порядке во всех уникальных неточных повторах.

В последнем блоке из исходных повторов исключаются точки расширения, составляя список с выделенным в каждом повторе архетипом. Такой формат нужен для дальнейшей визуализации результатов.

3.2.2. Функция парного выравнивания

Функция **`pairwise_alignment(duplicate_a, duplicate_b)`** реализует модифицированный алгоритм Нидлмана-Вунша (см. рис. 3.2). Она используется в блоке «Поиск архетипа» по описанной в начале главы схеме.

```
141  def alg(duplicate_a, duplicate_b):
142      mulct = -1
143      num_words = max(len(duplicate_a), len(duplicate_b))
144
145      # Заполнение оценочной матрицы
146      matrix_score = [[0] * (len(duplicate_a) + 1) for i in range(len(duplicate_b) + 1)]
147      for i in range(1, len(duplicate_a) + 1):
148          matrix_score[0][i] = mulct * i
149      for i in range(1, len(duplicate_b) + 1):
150          matrix_score[i][0] = mulct * i
151      for i in range(1, len(duplicate_b) + 1):...
160      #
161
162      # Обратный проход матрицы
163      res = []
164      i = len(duplicate_b)
165      j = len(duplicate_a)
166      while i > 0 and j > 0:...
180      res.reverse()
181      #
182      return res
```

Рис. 3.2. Функция парного выравнивания

Реализация, как и исходный алгоритм, разделена на две части: заполнение оценочной матрицы и её обратный проход. Реализация повторяет алгоритм, описанный в главе «Обзор», за исключением нескольких деталей. Изменению в нем подверглись поощрение при совпадении элементов и обратный проход матрицы. При совпадении элементов теперь учитываются длина совпавшего элемента и общее количество элементов, благодаря чему предпочтение всегда отдаётся наиболее длинному конечному результату, что позволяет выделить наиболее максимальный архетип. А в обратном проходе вместо применения действий записываются элементы из совпадений, которые были выбраны в качестве наилучшего действия, то есть записывается архетип без каких-либо лишних действий.

3.3. Визуализация результатов

Конечная визуализация группы неточных повторов с выделенным архетипом с помощью вышеописанного алгоритма была реализована в виде HTML-отчета браузера неточных повторов (см. рис. 3.3), который использует инструмент Duplicate Finder, работая в автоматическом режиме. В таблице содержится вся нужная информация о группе неточных повторов. В первой колонке содержится номер группы, который всегда будет равен единице, так как в данном отчете будет всего одна группа, во второй — количество повторов в группе, в третьей — количество точек расширения. В последней колонке отображается сама группа повторов, состоящая из точек расширения и архетипа. Точки расширения ограничены двойными треугольниками, между которыми указан номер соответствующей точки. Внутри этой области разными цветами указываются возможные значения соответствующей точки расширения для каждого повтора в группе. Внизу, под таблицей, находится окно с полным содержимым всех повторов из группы. При выборе возможного

значения точки расширения в этом окне выделяется соответствующий повтор. Для просмотра HTML-отчета поддерживаются браузеры FireFox и Chrome.

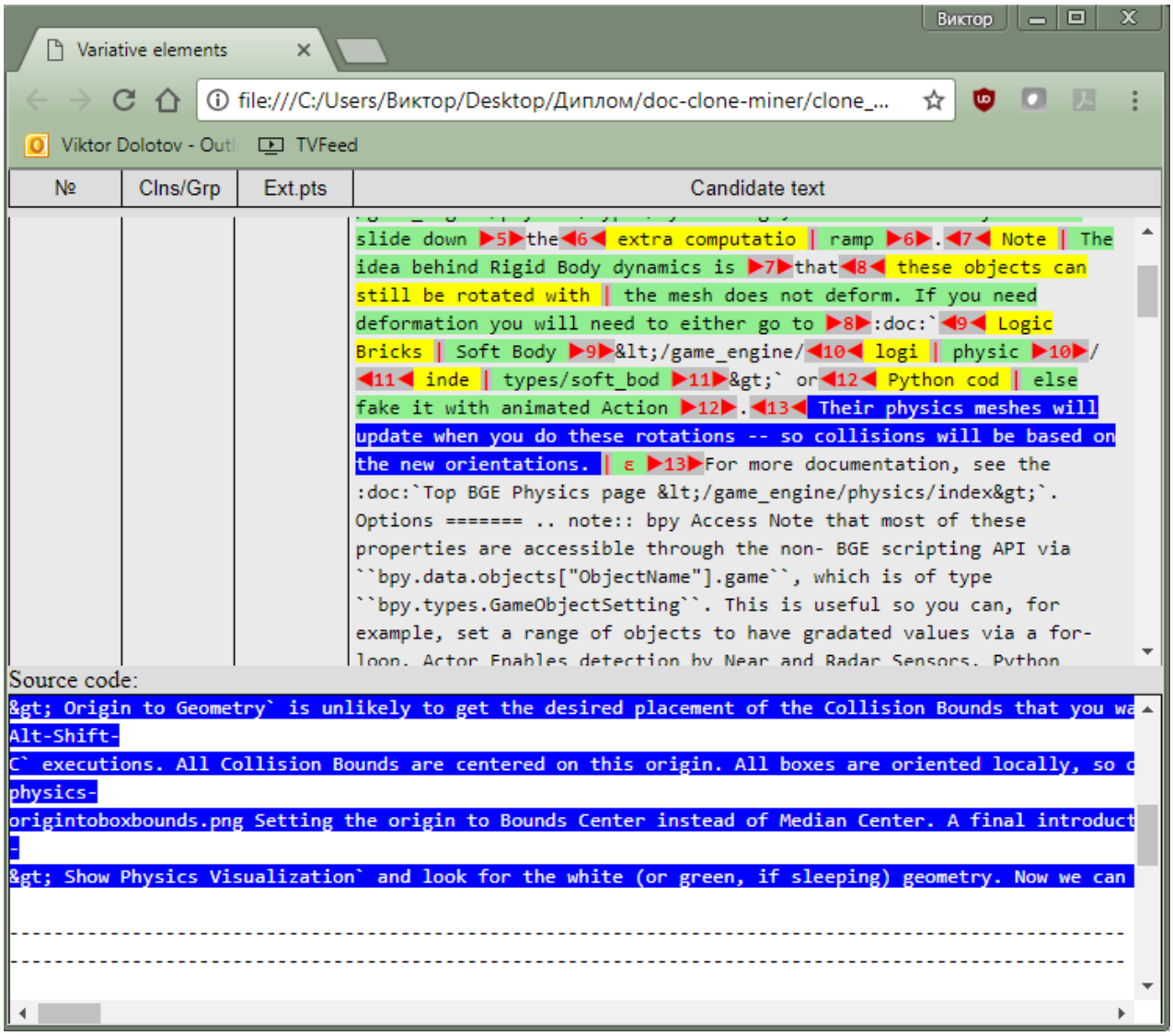

Рис. 3.3. HTML-отчет

Для программной реализации визуализации в библиотеку была добавлена ещё одна функция (get_html(group, archetype)), принимающая на вход группу неточных повторов и найденный с помощью описанного ранее алгоритма архетип. Так как браузер неточных повторов вместе с созданием HTML-отчета уже реализованы в рамках инструмента Duplicate Finder, от функции

потребовалось лишь запустить нужный метод (модуль util, функция write_variative_report(clones, candidates, report_file_name)). Для этого неточные повторы с выделенным в них архетипом преобразовываются в специальный класс VariativeElement (модуль clones), который подаётся на вход методу создания HTML-отчета.

Преобразование состоит из нескольких этапов. Во-первых, все повторы из группы записываются по порядку в один новый общий файл, чтобы отображать его внизу отчета (модуль clones, класс InputFile). Во-вторых, каждый фрагмент выделенного в неточных повторах архетипа представляется в виде двух индексов, которые соответствуют начальному и конечному символу этого фрагмента в ранее созданном общем файле. В-третьих, из получившихся наборов пар индексов составляется лист специальных классов ExactCloneGroup (модуль clones), содержащих информацию о точных повторах фрагментов текста, то есть о кусочках архетипа. И, наконец, из этого листа простым вызовом конструктора создаётся экземпляр класса VariativeElement.

Таким образом, пользователь, работая в интерактивном режиме инструмента Duplicate Finder, сможет после использования поиска неточных повторов по образцу наглядно увидеть, изучить и проанализировать полученный результат.

4. Апробация алгоритма

4.1. Формирование тестовых данных

Для проверки корректности работы алгоритма, а также с целью обнаружения недостатков был сформирован набор тестовых данных. Для его формирования было отобрано 15 групп неточных повторов из реальной документации ПО 7 проектов. Рассматривались самые интересные для эксперимента группы: с наибольшим средним количеством символов в повторе, уникальных повторов и точек расширения. Были выбраны именно такие критерии, так как они больше всего влияют на время работы алгоритма.

4.2. Схема экспериментов

Перед началом эксперимента был применён инструмент Duplicate Finder. Результатом работы автоматического режима инструмента являются группы неточных повторов с выделенными архетипами, поэтому по найденным им повторам легко проверить алгоритм поиска архетипа. Таким образом, сперва для тестовых данных выделялся правильный архетип с помощью автоматического режима инструмента Duplicate Finder. После чего архетип выделялся уже с помощью реализованного алгоритма. Условием успешного прохождения теста являлось нахождение алгоритмом правильного архетипа.

4.3. Анализ результатов

Тестовые данные были разбиты на 2 блока: с наибольшим количеством точек расширения (5 групп) и с наибольшим количеством символов и уникальных повторов (10 групп) соответственно. Эксперименты проводились на ноутбуке с центральным процессором Intel Core i7 7700HQ (тактовая частота 3,8 ГГц, кэш 6 Мб) и 16 Гб ОЗУ.

Результаты тестов с первым блоком групп (см. табл. 1, номера с 1 по 5) показали некоторое расхождение с архетипами, которые были найдены с помощью инструмента Duplicate Finder. Дело в том, что алгоритм, реализованный в рамках данной работы, выделяет архетип точнее — внутри некоторых вариативных точек выделяются схожие фрагменты текста, и они делятся на более мелкие по сравнению с вариативными точками, выделенными Duplicate Finder. Таким образом, вместе с точностью возрастает и количество точек расширения, а архетипы, найденные с использованием алгоритма парного выравнивания, целиком содержат в себе правильные архетипы, дополняя их.

№	Точки расшир.(Duplicate Finder)	Точки расшир.	Уник. повторы	Ср. длина	Время
1	7	13	2	4266	2.218092918395996
2	9	15	2	2960	0.3630621433258056
3	10	13	2	271	0.0059831142425537
4	18	25	2	819.5	0.061834335327148
5	3	13	2	773	0.0309355258941650
6	1	1	10	564.8	9.145504951477050
7	1	1	10	213.8	1.412255048751831
8	1	1	67	416.2	10.44035216140747
9	1	1	11	132.9	0.361002445220947
10	1	1	12	206.8	0.161567687988281
11	0	0	1	47235	0.031914472579956
12	0	0	1	27882	0.016954660415649
13	3	19	2	7372.5	4.893900394439697
14	11	27	2	7379	7.181818008422852
15	2	18	2	7051	7.852022409439087

Табл. 1. Результаты работы алгоритма

Тесты второго блока данных проводились для закрепления результатов тестов первого блока и для изучения времени работы алгоритма. Выделенные архетипы также целиком содержали правильные архетипы, полученные с помощью инструмента Duplicate Finder. Максимальное время работы

алгоритма составило 10,44 сек (группа с 67 уникальными повторами). Как видно из результатов тестов (см. табл. 1, номера с 6 по 15), на время работы оба параметра (длина повторов и количество уникальных повторов) влияют примерно одинаково.

Из результатов тестов можно сделать вывод о том, что алгоритм корректно выделяет архетип у групп неточных повторов. Среднее время выделение у группы архетипа составило 0,0096 сек. (рассматривались 12307 групп из 19 реальных документаций). Однако, работая с некоторыми большими (с большим количеством уникальных повторов) и качественными (с длинными повторами и большим количеством точек расширения) группами, алгоритм может выделять архетип до 11 секунд.

Заключение

В рамках данной работы были достигнуты следующие результаты.

1. Используя инструмент Duplicate Finder, были получены текстовые повторы в документациях программного обеспечения, анализ которых в дальнейшем был использован для оптимизации алгоритма.
2. Разработан и реализован алгоритм на базе алгоритма Нидлмана-Вунша для выделения архетипа в выборках похожих строк с визуальным представлением результатов работы в виде браузера неточных повторов.
3. Проведена апробация с составлением статистики реализованного алгоритма на найденных в документации ПО 15 группах повторов, обоснована корректность результатов работы алгоритма.

Список литературы

1. Романовский, К.Ю. Метод повторного использования документации семейств программных продуктов / К.Ю. Романовский. Диссертация на соискание научной степени кандидата физико-математических наук. — Санкт-Петербургский государственный университет. — 2010. — 111 с.
2. Corbet J. The present and future of formatted kernel documentation, 2016. — URL: https://lwn.net/Articles/671496/
3. Nikula J. Kernel documentation with Sphinx, part 1: how we got here, 2016. — URL: https://lwn.net/Articles/692704/
4. Horie M., Chiba S. Tool Support for Crosscutting Concerns of API Documentation. Proceedings of the 9th International Conference on Aspect-Oriented Software Development. p. 97–108, 2010.
5. Juergens E., Deissenboeck F., Feilkas M., Hummel B., Schaetz B., Wagner S., Domann C., Streit J. Can clone detection support quality assessments of requirements specifications? Proceedings of ACM/IEEE 32nd International Conference on Software Engineering vol.2. p. 79–88, 2010.
6. Nosál' M., Porubän J. Preliminary report on empirical study of repeated fragments in internal documentation. Proceedings of Federated Conference on Computer Science and Information Systems. p. 1573–1576, 2016.
7. Wagner S., Mendez Fernandez D. Analyzing Text in Software Projects. The Art and Science of Analyzing Software Data, Elsevier. p. 39–72, 2015.
8. Koznov D.V., Luciv D.V., Chernishev G.A., Terekhov A.N. Detecting Near Duplicates in Software Documentation. ArXiv EPrint, 2017.

9. Кантеев Л.Д., Костюков Ю.О., Луцив Д.В., Кознов Д.В., Смирнов М.Н. Обнаружение неточно повторяющегося текста в документации программного обеспечения. Труды Института системного программирования РАН, № 4. с. 303–314, 2017.

10. Луцив Д.В., Кознов Д.В., Басит Х.А., Терехов А.Н. Задача поиска нечетких повторов при организации повторного использования документации. Программирование, № 4. с. 39–49, 2016.

11. Луцив Д.В. Поиск неточных повторов в документации программного обеспечения / Д.В. Луцив. Диссертация на соискание научной степени кандидата физико-математических наук. — Санкт-Петербургский государственный университет. — 2018. — 122 с.

12. Луцив Д.В., Кознов Д.В., Басит Х.А., Ли О.Е., Смирнов М.Н., Романовский К.Ю. Метод поиска повторяющихся фрагментов текста в технической документации. Научно-технический вестник информационных технологий, механики и оптики No4 (92), СПб НИУ ИТМО. с. 106-114, 2014.

13. Koznov D.V., Luciv D.V., Chernishev G.A. Duplicate management in software documentation maintenance. Proceedings of V International conference Actual problems of system and software engineering (APSSE), CEUR Workshop Proceedings. p. 195–201, 2017.

14. Koznov D.V., Luciv D.V., Basit H.A., Lieh O.E., Smirnov M.N. Clone Detection in Reuse of Software Technical Documentation. Lecture Notes in Computer Science. p. 170–185, 2016.

15. Романовский К.Ю., Кознов Д.В. DocLine: метод разработки документации семейств программных продуктов. Программирование №4, 2008.

16. Minchin L., Romanovsky K., Koznov D. Refactoring the Documentation of Software Product Lines. Lecture Notes in Computer Science. Vol. 4980 c. 158–170, 2011.

17. Кознов Д.В., Шутак А.В., Смирнов М.Н., Смажевский М.А. Поиск клонов при рефакторинге технической документации. Компьютерные инструменты в образовании, № 4. с. 30–40, 2012.

18. Инструмент Duplicate Finder. — URL: http://www.math.spbu.ru/user/kromanovsky/docline/duplicate-finder-ru.html

19. Basit H. A., Jarzabek S. Efficient Token Based Clone Detection with Flexible Tokenization. Proceedings of the 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers. p. 513-516, 2007.

20. Jenkins A. D., Kratochvíl P., Stepto R. F. T., Suter U. W. Glossary of basic terms in polymer science. IUPAC Recommendations. p. 2289, 1996.

21. Mount DM. Bioinformatics: Sequence and Genome Analysis. Cold Spring Harbor Laboratory Press: Cold Spring Harbor, NY., №2, 2004.

22. 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 48 (3). p. 443–453, 1970.

23. Smith T. F., Waterman M. S. Identification of Common Molecular Subsequences. Journal of Molecular Biology 147. p. 195–197, 1981.

24. Ratcliff J.W., Metzener D. Pattern Matching: The Gestalt Approach. Dr. Dobb's Journal. p. 46, 1988.

25. Hirschberg D. S. A linear space algorithm for computing maximal common subsequences. Communications of the ACM 18 (6). p. 341–343, 1975.

26. Osamu Gotoh. An improved algorithm for matching biological sequences. Journal of Molecular Biology 162. p. 705-708, 1982.

27. Stephen F. Altschul, Bruce W. Erickson. Optimal sequence alignment using affine gap costs. Bulletin of Mathematical Biology 48. p. 603–616, 1986.

28. Miller Webb, Myers Eugene. Optimal alignments in linear space. Bioinformatics 4. p. 11–17, 1988.

29. Difflib, Python Documentation. — URL: https://docs.python.org/3/library/difflib.html

30. Python, официальная страница. — URL: https://www.python.org/

31. JetBrains PyCharm, официальная страница. — URL: https://www.jetbrains.com/pycharm/

32. Itertools, Python Documentation. — URL: https://docs.python.org/3/library/itertools.html