Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Шитов Егор Александрович

Система для анализа библиометрических показателей открытого репозитория CEUR Workshop Proceedings

Курсовая работа

Научный руководитель:
Чернышев Г. А.

Санкт-Петербург
2019

Оглавление

Введение

Любой современный исследователь в процессе своей деятельности неизбежно сталкивается с проблемой поиска научных статей. Решить такую проблему помогают сервисы-агрегаторы научных статей. На сегодняшний день существует несколько таких сервисов. Они условно бесплатные: в них можно бесплатно найти статью, используя, например, библиометрические показатели, однако, скачать статью можно только платно. Примером таких сервисов могут быть ACM Digital Library [1], Springer Link [16], IEEE Xplore Digital Library [11]. У некоторых сервисов отсутствует функционал предоставления доступа к статьям: например у DBLP [8].

С другой стороны, на сегодняшний день существуют площадки, на которых ученые выкладывают свои статьи в публичный доступ. Пример такой площадки — CEUR [6]. Однако сейчас для CEUR не существует отдельного агрегатора статей, отчего искать какие-либо статьи на CEUR сложно.

Отдельно можно отметить сервис arXiv.org [4] — это полностью бесплатная площадка, на которую выкладываются труды по физике, математике, астрономии, информатике и биологии. На данную площадку было выложено уже более миллиона статей. Также там присутствует поиск статей по разным параметрам: по названию, по автору, по DOI и др. Однако в данном сервисе нет библиометрической информации статей и авторов.

Рассмотрим некоторые ситуации, в которых агрегатор научных статей для CEUR может быть полезен.

Во-первых, с учетом того, что все больше исследователей выкладывают свои труды на CEUR, становится актуальной проблема поиска выдающихся работ за последнее время. В решении данной проблемы может помочь библиометрическая информация. Такие показатели как количество цитирований, место работы автора, конференция, на которой была опубликована статья, могут упростить поиск.

Во-вторых, зачастую, ученые не ограничиваются одной публикаци-

ей по какой-то предметной области. Следовательно, почти всегда возникает потребность посмотреть другие работы автора. В поиске таких работ может помочь агрегатор статей, который предоставляет функционал просмотра всех работ конкретного автора.

В-третьих, при исследовании какой-то новой предметной области полезно будет найти общепризнанные хорошие работы в этой области с большим количеством цитирований, так как такие работы могут послужить хорошей отправной точкой. В решении данной проблемы может помочь библиометрические показатели, например, количество цитирований.

В заключение, библиометрическая информация, построенная по данным с CEUR, позволит понять, как часто исследователи, которые выкладывают свои труды на CEUR, цитируют таких же исследователей. Отчасти это поможет определить рейтинг (импакт-фактор)[27] CEUR.

1. Постановка задачи

Цель работы - разработать прототип сервиса для анализа библиографической информации открытого репозитория CEUR-WS. Задачи, поставленные для решения в рамках данной работы:

- реализовать забор данных с сайта
- спроектировать структуру базы данных
- извлекать информацию из pdf-файла статьи
- реализовать модуль дедупликации статей
- разработать пользовательский интерфейс

2. Обзор инструментов

В данном разделе описан набор инструментов, который используется для реализации поставленных задач.

2.1. Выбор фреймворка

На сегодняшний день существует множество разных фреймворков для написания web сервисов. В том числе существует большое количество языков, на которых можно реализовать данные задачи. Рассмотрим наиболее популярные подходы:

- Python и Django [9]
- Javascript и Node.js [14]
- Java и Spring Framework [15]
- C# и ASP.NET [5]
- PHP

Все они достаточно популярны и используются во многих сервисах. Однако, для конкретной задачи какой-то из них может подходить лучше остальных.

К примеру, ввиду того, что с ростом проекта увеличивается вероятность появления ошибки, на динамически типизированных языках сложнее писать большие проекты, так как статический анализ присутствует в меньшей мере. Поэтому Python, Javascript, PHP было решено не использовать для данной задачи.

И Spring Framework, и ASP.NET предоставляют широкий функционал для реализации проектов любой сложности. Однако в рамках данной работы было решено использовать Spring Framework, так как у него большее количество готовых решений. К тому же, при широком наборе возможностей, уровень вхождения в Spring Framework достаточно низок, из-за Spring Boot, что является важным фактором при разработке прототипа.

2.2. Выбор языка

Как основной язык разработки был выбран Kotlin. У него есть несколько плюсов:

- код в среднем пишется быстрее, из-за лаконичности языка
- код является более безопасным
- данный язык предоставляет иной подход для параллельного программирования — корутины
- в экспериментальной версии языка есть каналы, которые позволяют безопасно обмениваться информацией между корутинами

Минусы:

- неполная совместимость с Spring Boot
- сложность в понимании некоторых конструкций
- меньшая стабильность, чем у Java

Описанные выше минусы не являются критичными для данного проекта.

Как второстепенный язык для некоторых вспомогательных модулей был выбран Golang.

Плюсы:

- целевые приложения — серверные приложения
- богатая стандартная библиотека
- скорость выполнения
- простота написания кода, из-за отсутствия поддержки сложных конструкция
- благодаря единому синтаксису, отсутствию классов и нестандартному подходу наследования проекты на Golang проще поддерживать, чем проекты на классических языках (Java, C#, C++)

2.3. Выбор базы данных

Ввиду того, что у Spring Framework есть готовая ORM-модель для SQL баз данных, рассматривались именно SQL базы.

Наиболее популярные базы данных:

- MySQL
- PostgreSQL
- MS SQL

Ввиду того, что для малого проекта различие между этими базами данных не так критично, а Spring Framework интегрирован с MySQL лучше, чем с остальными, было решено остановиться именно на MySQL.

3. Архитектура сервиса

В данном разделе будет рассмотрена общая архитектура сервиса, архитектура сайта и структура базы данных.

3.1. Общая архитектура

Ввиду того, что некоторые поставленные задачи слабо коррелируют друг с другом (например, дедупликация и пользовательский интерфейс), было решено использовать модульную архитектуру. Основной сайт и модуль дедупликации реализованы как два разных приложения, взаимодействие которых происходит по локальной сети. Отсюда следует ряд преимуществ:

- отказоустойчивость. При возникновении ошибки на стороне сайта в процессе дедупликации, процесс дедупликации корректно закончит свою работу. Следовательно, все внешние ресурсы, в частности база данных, не останутся в неконсистентном состоянии.

- простота замены модуля дедупликации. Реализация модуля дедупликации никак не зависит от реализации сайта и наоборот. Следовательно, замена одного модуля дедупликации другим происходит незаметно для сайта. Такой подход может быть полезен, например, для тестирования новых алгоритмов. Приведем пример. Предположим, что возникла необходимость проверить новый алгоритм, основанный на машинном обучении. На данный момент, самый простой способ его реализовать — на языке Python. Допустим, что данный алгоритм работает лучше предыдущего. Однако у реализации на языке Python могут быть проблемы с производительностью. Поэтому, в дальнейшем этот алгоритм может быть реализован на более быстром, но сложном языке — например, на C++. То есть, такой подход позволяет быстро и с минимальными затратами проверить какие-то новые гипотезы.

- распределение нагрузки. Так как модули слабо связаны между

собой, они могут выполняться на разных машинах. В таком случае, нагрузка на сайт не будет никак коррелировать с нагрузкой на модуль дедупликации.

3.2. Архитектура сайта

За основу взята трехуровневая архитектура.

Рис. 1: Трехуровневая архитектура

На уровне данных отправляются запросы для получения данных, например запросы в базу данных и различные сетевые запросы. Результаты передаются на уровень логики, цель которого — преобразовать полученные данные в соответствии с поставленными задачами. Далее результаты работы уровня логики передаются уровню представления, основная задача которого — отображать данные конечному пользователю. Стоит заметить, что хорошим тоном при реализации данного подхода является то, что классы более высокого уровня агрегируют классы более низкого уровня, в то время как классы более низкого уровня отправляют результаты классам высокого через интерфейсы. То есть реализация более высокого уровня сокрыта от более нижнего уровня.

Плюсы данной архитектуры:

- структурированный код.
- простота тестирования отдельных классов ввиду простоты создания среды для тестирования данных классов — для необходимых зависимостей можно создать макет [13].
- данная архитектура является широко распространенной. Следовательно, упрощается вхождение новых людей в проект. Следовательно, снижаются затраты на поддержку кода и на добавление новой функциональности.

Ввиду того, что существуют задачи, которые выполняются достаточно долго, появляется следующая проблема: после запуска долгой задачи пользователь, не дожидаясь окончания выполнения, производит попытку запуска еще раз. При отсутствии решения данной проблемы в зависимости от реализации могут произойти следующие события:

- одновременно выполняются две одинаковых задачи
- выполнение первой задачи экстренно прерывается, заново запускается такая же задача

В случае, если задача взаимодействует с внешними ресурсами, то есть вероятность, что данные ресурсы могут остаться в неконсистентном состоянии. Каналы и корутины в языке kotlin предоставляют инструментарий для решения данной проблемы.

Рассмотрим следующий код:

```
val channel: ReceiveChannel<T>? = null
val mtx = Mutex()
...
fun runTaskIfNeeded(result: (T) -> Unit){
    val ch = mtx.withLock{ channel }

    val taskChannel = if(ch == null || ch.isClosedForReceive){
        val taskChannel = task()
        mtx.withLock { channel = taskChannel }
        taskChannel
    } else ch
```

```
    for(element in taskChannel)
        result(element)
}
```

Сначала производится проверка, не запущена ли уже данная задача. Если задача не запущена, то она запускается, канал, который она возвращает, запоминается, указатель дублируется в переменную taskChannel. Если задача запущена, то канал этой задачи уже известен, указатель хранится в переменной channel. В таком случае он копируется в переменную taskChannel. Если задача успешно завершила свое выполнение и данные, которые в процессе были отправлены в канал, считаны, то канал закрывается, и при следующем вызове функции задача запускается заново. В конце происходит обработка возвращаемых данных задачи. Таким образом, удается избежать проблему повторного запуска работающей задачи.

Такой подход добавляет еще один слой-прослойку между уровнем представления и уровнем логики.

3.3. Структура базы данных

В базе данных существует три основных таблицы — authors, papers, volumes. Таблица authors содержит в себе информацию об авторах статей и разделов CEUR. Таблица papers содержит информацию о статьях. Таблица volumes — информацию о разделах CEUR.

Для обеспечения связей многие-ко-многим в базе данных присутствуют вспомогательные таблицы:

- authors_linked_to_papers — таблица отношения авторы-статьи
- editors_linked_to_volumes — таблица отношения редакторы-разделы
- paper_references — таблица отношения ссылок статей
- submitters_linked_to_volumes — таблица отношения тех, кто загрузил разделы на CEUR, к разделам

Ниже приведена диаграмма связей в базе данных.

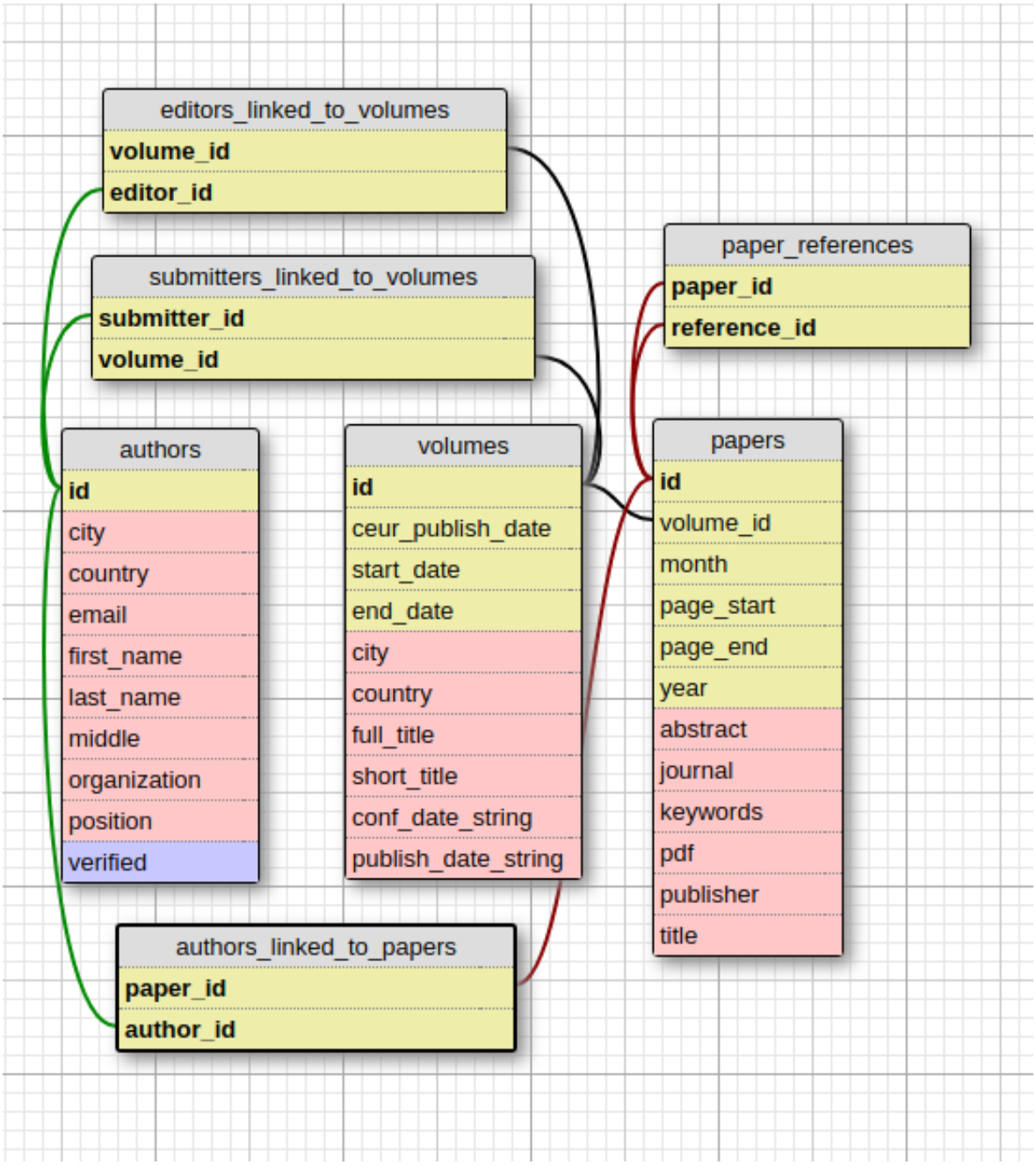

Рис. 2: Структура базы данных

4. Извлечение информации с сайта CEUR

В данном разделе будет описан механизм извлечения информации о статьях с сайта CEUR-WS.org.

На данной площадке статьи группируются в разделы. Нумерация разделов идет по-порядку с нуля. Каждый раздел доступен через FTP-сервер как архив в формате zip. После извлечения файлов из архива

нужная информация о статьях находится в файле index.html, а сами статьи предоставляются как pdf-файлы.

Таким образом, алгоритм для извлечения информации выглядит следующим образом:

- загрузить zip архив раздела с FTP-сервера CEUR
- распаковать архив
- извлечь мета-информацию о статьях данного раздела из файла index.html
- извлечь информацию о цитировании, abstract и ключевые слова из pdf-файла статьи

Существует большое количество библиотек, которые позволяют реализовать первые три пункта. Для загрузки файлов с FTP-сервера был выбран инструментарий из Apache Commons [3]. Для распаковки архива — библиотека zip4j [17]. Для быстрой навигации по HTML документу была выбрана библиотека jsoup [12], которая позволяет удобно обращаться к тегам и атрибутам HTML документа.

5. Извлечение информации из статей

В данном разделе описаны инструменты, которые были испробованы в данной работе для извлечения информации из статей.

5.1. AnyStyle parser

AnyStyle parser [2] — проект, написанный на языке Ruby. Доступен через CLI и как web сервис. Данная утилита на вход принимает заранее изъятую из pdf-файла ссылку на статью. На выходе выдает сегментированную информацию о статье: авторов статьи, название статьи, год публикации и т.д. Данный проект легко разворачивается, так как устанавливается через пакетный менеджер, который автоматически загружает все необходимые зависимости. Более того, данный проект прост в использовании, так как взаимодействие основного модуля разрабатываемого сервиса и данного проекта может происходить по сети, что упрощает интеграцию.

Однако, у anystyle parser есть существенный недостаток — плохое качество сегментирования. Как показывает статья [26], F1-мера данного сегментатора — 54%.

5.2. FreeCite

FreeCite [10] — проект, также написанный на языке Ruby. Взаимодействие осуществляется либо через CLI, либо через web сервис. Аналогично, принимает на вход заранее изъятую из pdf-файла ссылку на статью. На выходе возвращает сегментированную информацию. FreeCite запустить сложнее, чем AnyStyle, ввиду неподдерживаемости проекта (последнее изменение было в апреле 2009 года). Однако в использовании он так же прост, так как взаимодействие с ним можно производить по сети. Субъективно, качество сегментирования у FreeCite лучше, чем у AnyStyle, но присутствует один дефект. Если в ссылке на статью авторы указаны как фамилия и инициалы, то инициалы последнего автора зачастую сегментировались как начало названия статьи. По наблюде-

ниям в ходе данной работы, такая ошибка встречалась примерно в 160 случаях из 500. Из-за такой ошибки некоторые алгоритмы дедупликации могли работать значительно хуже.

5.3. GROBID

GROBID [25] — проект, который до сих пор поддерживается. Написан на языке Java. Доступен как отдельное приложение и как библиотека для jvm-проекта. В отличие от рассмотренных выше проектов GROBID принимает на вход pdf файл статьи. На выходе выдает информацию об авторах (ФИО, место работы), о статье (ключевые слова и abstract), ссылки на другие статьи. Качество извлечения информации у GROBID значительно выше, чем у рассмотренных выше проектов. Как показано в [26], F1-мера GROBID — 89%.

Вывод

GROBID предоставляет больший функционал по сравнению с рассмотренными выше аналогами. Причем, ввиду того, что GROBID работает с pdf файлами, он является более простым в использовании. Недостатком GROBID является то, что он работает медленнее, чем его аналоги. Однако так как известно, что количество запросов на извлечение информации из pdf файлов статей не будет большим и контролируется доверенным лицом, то скорость работы GROBID не является существенным недостатком для данного проекта. Следовательно, было решено выбрать GROBID для извлечения информации из pdf файлов статей.

6. Дедупликация

В данном разделе будут описаны методы дедупликации библиографических записей в базе данных.

На данный момент существует несколько подходов для решения данной задачи. Рассмотрим некоторые из них.

6.1. Генетическое программирование

В статьях [22, 20] авторы попробовали применить генетическое программирование для построения функции, которая определяет, указывают ли записи на одну и ту же статью или нет. Плюсы данного подхода:

- автоматически выбираются наиболее важные признаки
- выводится большое количество функций, следовательно, можно выбрать наиболее подходящую
- простота в реализации
- на 12% лучше базового метода Fellegi & Sunter [23]

Минусы:

- F-мера данного подхода на Cora [7] датасете 82%. Данный результат достигают и другие подходы, более легкие в реализации.

6.2. Эвристический алгоритм

В статье [19] описан алгоритм, который последовательно сравнивает некоторые поля библиографической информации, затем выносит вердикт: совпадают статьи или нет.

Общий принцип алгоритма:

- Сравнить года публикаций. Если они различаются больше, чем на определенный порог, то считать статьи разными. Обычно в качестве порога берется 1.

- Сравнить авторов статей. Если показатели различаются больше, чем на определенный порог, то считать статьи разными.

- Сравнить название статей с помощью расстояния Левенштейна. Считать статьи разными, если расстояние больше, чем определенный порог.

Такой подход позволяет отсеять большое количество несовпадений на раннем этапе, не запуская алгоритм сравнения строк, который вычислительно затратный.

Рассмотрим функции, которые используются для сравнения авторов.

$$IniSim(a,b) = \begin{cases} & a_1 == b_1 \wedge a_m == b_n \vee \\ 1, & a_1 == b_2 \wedge a_m == b_1 \vee \\ & a_1 == b_1 \wedge a_2 == b_2 \vee \\ & a_1 == b_n \wedge a_2 == b_1 \\ 0, & \end{cases} \quad (1)$$

IniSim - функция, которая сравнивает инициалы авторов. Она покрывает некоторую вариативность написания ФИО автора. Как важный эвристический признак тут используется тот факт, что ошибок в написании первой буквы какой-либо части ФИО гораздо меньше, чем ошибок в написании других букв. Из недостатков можно выделить то, что данная функция не покрывает случаи, когда у ФИО есть какие-то приставки или суффиксы, указывающие на должность автора.

Алгоритм NameMatch сравнивает каждого автора с каждым, используя функцию IniSim. Если количество совпадений больше, чем определенный порог, то считается, что статью написали одни и те же авторы.

Проблемы данного подхода:

- в названии статьи могут быть пропущены слова, следовательно, из-за расстояния Левенштейна статьи будут считаться разными

Algorithm 1 NameMatch

```
1: m ← LENGTH(K)
2: n ← LENGTH(L)
3: counter ← 0
4: for i ← 1 to m do
5:     for j ← 1 to n do
6:         if IniSim(K_i, L_j == 1) then
7:             counter ← counter + 1
8:             L_j ← null
9:         end if
10:    end for
11: end for
12: if counter/Max(m, n) ≥ t_N then
13:    return 1
14: else
15:    return 0
16: end if
```

- в библиографической информации может отсутствовать год публикации
- существуют дубликаты, в которых год публикации расходится больше, чем на 1
- префиксы и суффиксы в именах обрабатываются функцией IniSim некорректно
- пропуски имени обрабатываются некорректно
- некоторые перестановки ФИО обрабатываются некорректно
- количество указанных авторов может быть сильно разным

F-мера данного подхода на Cora датасете достигает 82.6%, что сравнимо с F-мерой подхода, основанного на генетическом программировании. Однако реализация данного алгоритма значительно проще. К тому же, некоторые проблемы данного подхода можно решить. Например, добавить подход мешка слов для сравнения названий статей и для сравнивания ФИО авторов, не отсеивать статьи с неизвестным годом публикации.

6.3. Подход, основанный на дереве решений

В статье [18] рассмотрен подход, основанный на дереве решений. Формируются некоторые признаки схожести или различия двух статей, затем по ним формируется дерево решений. Наилучшие показатели получились у алгоритма C4.5 - 89.7% F-меры на Cora датасете, что больше, чем у выше рассмотренных методов.

Из плюсов данного метода можно выделить то, что он выделяет наиболее важные признаки, которые можно в дальнейшем анализировать.

Из минусов, как видно в статье, наиболее важный признак - различия в названии статей, которые вычисляются с помощью расстояния Левенштейна. Ввиду того, что он наиболее важный, он будет вычисляться для каждой пары статей, что приведет к медленной работе алгоритма по сравнению с эвристическим подходом.

6.4. Подходы, основанные на получении информации из сторонних сервисов

Существуют некоторые подходы [21, 24], которые предлагают делать дедупликацию, предварительно взяв информацию со сторонних сервисов, например DOI или MAS. Такие подходы, действительно, упрощают дедупликацию, но для данной работы не подходят ввиду нарушения ToS данных сервисов.

Вывод

Так как целью данной работы является разработка прототипа, то был выбран эвристический подход дедупликации статей. При простоте реализации его показатели примерно такие же, как и у других алгоритмов. К тому же, ввиду модульности системы, в дальнейшем можно будет более подробно исследовать другие подходы дедупликации, удобно их протестировать и внедрить в данную систему.

7. Заключение

В ходе данной работы реализован прототип сервиса по анализу библиометрических показателей открытого репозитория CEUR Workshop Proceedings, а именно:

- сделан модуль забора данных с сайта CEUR-WS.org
- спроектирована структура базы данных
- найден и применен эффективный инструмент для извлечения информации из pdf файла статей
- реализован модуль дедупликации статей
- реализован базовый пользовательский интерфейс

Список литературы

[1] Acm digital library. https://dl.acm.org/. Online; accessed 14.05.2019.

[2] Anystyle parser. https://github.com/inukshuk/anystyle. Online; accessed 14.05.2019.

[3] Apache commons. https://commons.apache.org/. Online; accessed 15.05.2019.

[4] Arxiv.org. https://arxiv.org. Online; accessed 15.05.2019.

[5] Asp.net. https://dotnet.microsoft.com/apps/aspnet. Online; accessed 14.05.2019.

[6] Ceur workshop proceedings. http://ceur-ws.org/. Online; accessed 14.05.2019.

[7] Cora dataset. https://people.cs.umass.edu/~mccallum/code-data.html. Online; accessed 14.05.2019.

[8] Dblp. https://dblp.uni-trier.de/. Online; accessed 14.05.2019.

[9] Django. https://www.djangoproject.com/. Online; accessed 14.05.2019.

[10] Freecite. http://freecite.library.brown.edu/. Online; accessed 14.05.2019.

[11] Ieee xplore digital library. https://ieeexplore.ieee.org/Xplore/home.jsp. Online; accessed 14.05.2019.

[12] Jsoup: Java html parser. https://jsoup.org/. Online; accessed 15.05.2019.

[13] Mockito framework. https://site.mockito.org/. Online; accessed 14.05.2019.

[14] Node.js. https://nodejs.org/. Online; accessed 14.05.2019.

[15] Spring framework. https://spring.io/. Online; accessed 14.05.2019.

[16] Springer link. https://link.springer.com/. Online; accessed 14.05.2019.

[17] Zip4j. http://www.lingala.net/zip4j/download.php. Online; accessed 15.05.2019.

[18] Eduardo Borges, Karin Becker, Carlos Heuser, and Renata Galante. A classification-based approach for bibliographic metadata deduplication. pages 221–228, 01 2011.

[19] Eduardo N. Borges, Moisés G. de Carvalho, Renata Galante, Marcos André Gonçalves, and Alberto H. F. Laender. An unsupervised heuristic-based approach for bibliographic metadata deduplication. *Inf. Process. Manage.*, 47(5):706–718, September 2011.

[20] Moisés G. Carvalho, Albero H. F. Laender, Marcos André Gonçalves, and Altigran S. da Silva. Replica identification using genetic programming. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 1801–1806, New York, NY, USA, 2008. ACM.

[21] Miew Keen Choong, Sarah Thorning, and Guy Tsafnat. Citation enrichment improves deduplication of primary evidence. In *Revised Selected Papers of the PAKDD 2015 Workshops on Trends and Applications in Knowledge Discovery and Data Mining - Volume 9441*, pages 237–244, New York, NY, USA, 2015. Springer-Verlag New York, Inc.

[22] Moisés G. de Carvalho, Marcos André Gonçalves, Alberto H. F. Laender, and Altigran S. da Silva. Learning to deduplicate. In *Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '06, pages 41–50, New York, NY, USA, 2006. ACM.

[23] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of American Statistical Association*, pages 1183–1210, 1969.

[24] Tin Huynh, Hiep Luong, and Kiem Hoang. Integrating bibliographical data of computer science publications from online digital libraries. In *Proceedings of the 4th Asian Conference on Intelligent Information and Database Systems - Volume Part III*, ACIIDS'12, pages 226–235, Berlin, Heidelberg, 2012. Springer-Verlag.

[25] Lopez P. GROBID: combining automatic bibliographic data recognition and term extraction for scholarship publications. *Research and Advanced Technology for Digital Libraries*, pages 473–474, 2009.

[26] Dominika Tkaczyk, Andrew Collins, Paraic Sheridan, and Joeran Beel. Machine learning vs. rules and out-of-the-box vs. retrained: An evaluation of open-source bibliographic reference and citation parsers. In *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries*, JCDL '18, pages 99–108, New York, NY, USA, 2018. ACM.

[27] Левин В. И. Библиометрические показатели или экспертные оценки: как оценивать результаты научной деятельности. *Современное образование*, (4):11–28, 2016.