

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
КАФЕДРА КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ И МНОГОПРОЦЕССОРНЫХ СИСТЕМ

**Васильев Игорь Сергеевич**

**Выпускная квалификационная работа бакалавра**

**Анализ социальных сетей с помощью технологий  
больших данных**

Направление 010300

Фундаментальные информатика и информационные технологии

Научный руководитель,

PhD,

доцент

Корхов Владимир Владиславович

Санкт-Петербург

2016

# Содержание

Введение. . . . .	3
Постановка задачи. . . . .	4
Обзор литературы. . . . .	5
Глава 1. Социальные графы и их анализ. . . . .	7
1.1. Обзор области. . . . .	7
1.2. Определение социального графа. . . . .	8
1.3. Характеристики социальных графов. . . . .	8
1.4. Основные алгоритмы анализа графов. . . . .	12
Глава 2. Граф соискатели — работодатели. . . . .	14
2.1. Определение основных понятий. . . . .	14
2.2. Разбор резюме и вакансий. . . . .	17
2.3. Использование Word2Vec и Doc2Vec. . . . .	18
Глава 3. Подготовка данных. . . . .	23
3.1. Описание структуры данных. . . . .	23
3.2. Работа с данными в Apache Spark. . . . .	25
Глава 4. Реализация. . . . .	26
4.1. Классификация документов и ключевых навыков. . . . .	26
4.2. Анализ резюме. . . . .	28
4.3. Анализ вакансии. . . . .	30
4.4. Определение связей между соискателями и вакансиями. . . . .	31
4.5. Построение графа. . . . .	31
Выводы. . . . .	34
Заключение. . . . .	35
Дальнейшая работа. . . . .	35
Список литературы. . . . .	36

## **Введение**

В связи с ростом популярности интернета, мобильных устройств, и т. д. наблюдается значительный рост объемов информации. Эта информация представлена в самом разном виде, как структурированная, так и без какой-либо определенной структуры. Хранение таких данных и последующий их анализ может привести к полезным, а иногда довольно неожиданным, результатам. Но для работы с таким количеством информации требуются новые технологии и методы, которые обозначаются термином Big Data.

## Постановка задачи

Имея коллекцию резюме и вакансий, имеющих слабоструктурированную форму, наиболее эффективно сопоставить соискателей работодателям. Для этого извлечь необходимую информацию с помощью методов обработки естественного языка, после чего, на её основе, построить двудольный граф, где одним множеством будут кандидаты, а другим — вакансии.

Задачу можно разбить на несколько подзадач:

1. Сбор данных;
2. Разработка методов обработки данных, их сравнение и применение;
3. Определение связи между сущностями и построение графа;
4. Определение оптимального назначения;
5. Перенос алгоритмов на платформу Big Data.

## Обзор литературы

В статье [5] от разработчиков фреймворка Apache Spark рассказывается о концепции главной его абстракции — RDD, что расшифровывается как Resilient Distributed Dataset. В ней объясняется за счет чего Spark получил такое увеличение производительности. Выделены приложения, где стоит применять RDD, и, наоборот, в которых RDD не принесет увеличения производительности. Особенностью RDD является выполнение вычислений in-memory, то есть внутри оперативной памяти, не выгружая промежуточные результаты на диск, в отличие от, например, Hadoop. В статье сказано, что сильной стороной данной абстракции являются итеративные алгоритмы, что может быть использовано в данной работе, потому что большое количество алгоритмов на социальных графах являются именно такими.

В статье [8] описываются особенности библиотеки GraphX и её взаимодействие с RDD. Рассказывается о модели распределенных вычислений на графах, о приложениях, в которых эта модель может оказаться полезной. Также в статье есть информация об основных методах, которые предоставляет эта библиотека, указания к тому, как наиболее эффективно их применять.

Целью статьи [6] являлось сравнение известных семантических моделей в контексте автоматизации резюмирования документов. В качестве таких моделей были выбраны Латентное размещение Дирихле, Латентно-семантический анализ, Word2Vec, Doc2Vec и TF-IDF. Результаты затем сравнивались с значениями, полученными экспертами.

В данной статье рассказано о структуре и особенностях каждой из 5 моделей, описан подход, с помощью которого они оценивались. Результатом

данной работы стал вывод о том, что Doc2Vec является лучшей моделью для данного типа задач. Однако результаты пока далеки от идеальных.

В статье [7] описывается реализация Doc2Vec, который способен предоставлять векторное представление для кусков текста. Большим плюсом данной модели является то, что она способна обучаться без учителя, что очень удобно в данной работе, потому что в свободном доступе нет необходимых маркированных данных. В статье есть ряд экспериментов, результаты которых позволяют сделать вывод о том, в каких случаях Doc2Vec может использоваться, а также о том, как лучше обучать модель.

Целью статьи [9] является разработка алгоритма, который помог бы автоматизировать поиск навыков соискателя в резюме. В отличие от подходов, где применяется векторное представление слов, авторы здесь попытались выделить особые признаки, которые указывают на то, что данное словосочетание определяет навык. Также в этой статье предложено разбить навыки по категориям, что в дальнейшем упростит поиск и сделает алгоритм более точным. Показан весь процесс, включая процесс обработки документов, определения веса определенного навыка, выделение и организация навыков.

# Глава 1. Социальные графы и их анализ

## 1.1. Обзор области

Анализ социальных графов — это междисциплинарная область, которая фокусируется на изучении взаимосвязанных сущностей, включая социальные, биологические, коммуникационные и компьютерные сети. Главной целью этой области является получение объясняющих и предсказывающих моделей для физических, социальных, технологических и биологических феноменов.

В социальных сетях существует множество типов отношений между сущностями. Их исследование — это шаг в сторону лучшего понимания социума. Оно включает в себя, например, разработку аналитических мер (analytical measures), алгоритмов для выделения сообществ, предсказывание связей между сущностями и анализ сильно связанных узлов (хабов).

Некоторые из теоретических достижений нашли своё приложение в следующем:

1. Мониторинг и анализ мобильных сетей (сети подвижности), используя траектории как рёбра, соединяющие разные географические районы и «точки интереса»;
2. Идентификация возможных заблуждений (ошибок) в обеспечении приватности и безопасности;
3. Вирусный маркетинг в социальных сетях;
4. Система, помогающая принимать решения по поимке преступников, анализирующая преступные тенденции;
5. Отслеживание и предсказание вспышек заболеваний с помощью Твиттера.

Стоит отметить, что анализ социальных сетей находит свое применение в огромном количестве областей, связанных с обработкой больших данных (везде, где можно представить данные как связанные между собой сущности и где нужно исследовать отношение между ними или их влияние друг на друга).

## **1.2. Определение социального графа**

Для понятия «социальный граф» нет единого определения. Говоря о нём, подразумевают контекстуальную социограмму, которая описывает участников, организации, группы внутри какой-то социальной сети, а также отношения между ними.

Социальная сеть — это структура, состоящая из множества социальных акторов (людей или организаций), а также множества связей между ними.

## **1.3. Характеристики социальных графов**

Говоря о задачах на социальном графе, употребляют термин метрики, которые в числовой форме отображают характеристики социальных объектов, сегментов/групп объектов и их связей. Эти метрики используются при проведении анализа социальных сетей. Самые общие из них:

1. Количество вершин;
2. Количество уникальных ребер;
3. Количество петель;
4. Количество связных компонент.



Метрики, характеризующие связи:

1. Гомогенность — степень, с которой схожие участники формируют связи между собой в сравнении с несхожими;
2. Множественность — количество форм, содержащихся в связи;
3. Обоюдность — степень, с которой двое участников отвечают друг другу взаимностью в сфере каких-либо взаимодействий.

Метрики, характеризующие распределение:

1. Мост — индивид, чьи слабые связи заполняют структурные пробелы, обеспечивая единственное соединение между двумя другими индивидами или кластерами;
2. Центральность — определяет значение или влияние определенного узла или группы в сети;
3. Структурные пробелы — отсутствие связи между двумя частями сети.

Самыми полезными и часто используемыми являются следующие метрики:

1. Degree Centrality (степень вершины) — как много людей (сущностей) связано с этим человеком напрямую;
2. Betweenness (промежуточность) — насколько вероятно, что этот узел будет элементом наикратчайшего пути, связывающего две другие сущности в сети;
3. Closeness (близость) — как быстро человек может достичь всех остальных участников сети;
4. Eigenvector — насколько хорошо этот человек связан с другими «сильно-связанными» людьми;

5. Shortest Path (кратчайший путь)— минимальное количество связей, требуемых для установления наличия взаимосвязи между двумя отдельными пользователями;
6. Set of key players.

Иллюстрация данных метрик представлена на рисунке 1.

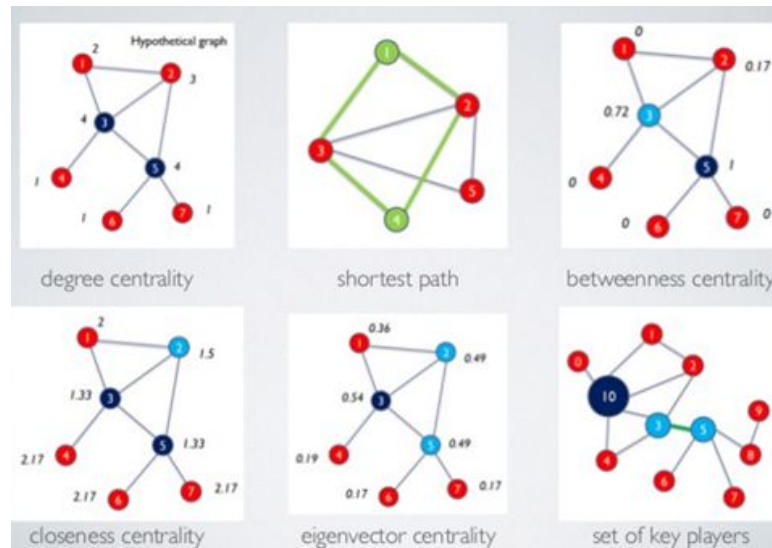


Рис. 1 Примеры метрик социальных графов [4]

Применение данных метрик можно посмотреть в таблице на рисунке 2:

Case	Questions	SNA Tools
Leader Selection	Who is the central in the trust and respect network?	Degree Centrality
Ranks	How do we rank our top performer individuals in the organization?	Eigenvector Centrality, Pageranks
Task Force Selection	How do we put together a team that maximally connect through out the	Closeness Centrality
Mergers and Acquisition	How to merge separate cultures/networks?	Homophily, Reciprocity, Mutuality, Transitivity
Competitive Advantages	What is the missing links between supply and demand?	Structural Holes
Advertising Attachment	How strong the impact of our advertisement effort?	Tie Strength
Market Segmentation	How segmented our market is?	Clustering Coefficient, Clique, Modularity
Information Dissemination	How is the information/knowledge spreading?	Random Walks, Hits Algorithm
Dynamics of Organization	How dynamics our organization is?	Temporal Networks

Рис. 2 Сравнительная таблица метрик [4]

Составными частями социальных сетей являются четыре основных компонента:

- Хабы — узлы, имеющие особенно высокую степень, пример на рисунке 3



Рис. 3 Хабы в социальном графе

- Мосты — тип связи, который соединяет две разные группы в одной сети, пример на рисунке 4

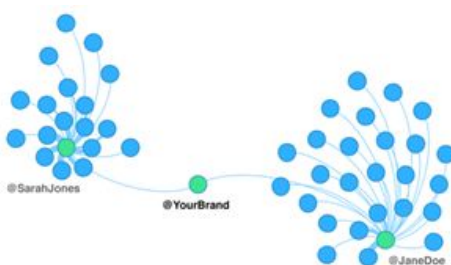


Рис. 4 Мосты в социальном графе

- Острова — слабосвязанные узлы или группы узлов, пример на рисунке 5

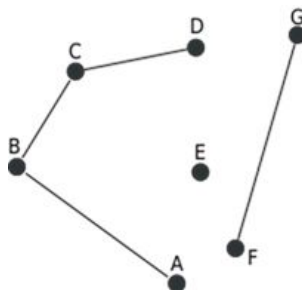


Рис. 5 Острова в социальном графе

- Кластеры — группы сильно связанных между собой узлов (обозначены разными цветами), пример на рисунке 6

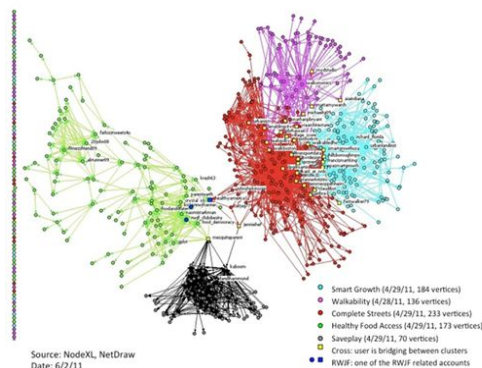


Рис. 6 Кластеры в социальном графе

В данной работе объектом анализа является рынок труда, а именно, отношение работодатель — соискатель. Их взаимодействие можно представить ориентированным двудольным графом.

## 1.4. Основные алгоритмы анализа графов

Самыми распространенными задачами анализа социальных графов являются выявление особенностей структуры. Например, можно выделить сообщества, то есть группы узлов, которые имеют сильную связность между

собой и слабую с узлами, не входящими в данное сообщество. Также довольно часто требуется выделить самых влиятельных членов группы. Широкое применение находят алгоритмы, предсказывающие возможные связи между сущностями, а также алгоритмы рекомендаций.

Одним из классических и универсальных алгоритмов является PageRank. В контексте социальных графов он может использоваться для выделения наиболее авторитетных вершин графа, а также для предсказания связей и предоставления рекомендаций. Например, персонализированный PageRank используют в Twitter, чтобы предлагать пользователям аккаунты, которые им могут быть интересны. Кроме того, относительно недавно его стали использовать, чтобы ранжировать публичные места и улицы для предсказания того, как много пешеходов или велосипедистов придут туда.

Другим классическим примером является поиск (сильно) связных компонент. Это алгоритм поиска подмножеств вершин графа таких, что между любыми двумя вершинами из конкретного подмножества существует путь, и не существует путей между вершинами разных подмножеств. Он может использоваться для выделения тесно взаимодействующих групп людей, чтобы затем предлагать членам этой группы, например, страницы, которые посчитали интересными другие члены данной группы.

Также стоит отметить алгоритм подсчета кратчайших путей в графе, который находит свое применение в поиске самых влиятельных узлов и выделении сообществ.

Большинство алгоритмов являются итеративными.

## Глава 2. Граф соискатели — работодатели

### 2.1. Определение основных понятий

Возможности Apache Spark позволяют проводить анализ над большим количеством неструктурированных данных. К данной задаче это можно применить, например, автоматически учитывая в каждом резюме не только ключевые навыки, а также предыдущий опыт работы и информацию о соискателе. Также можно более детально рассматривать информацию о вакансии, а именно её описание в свободной форме.

Полученный граф можно анализировать, извлекая следующую информацию:

- Самые востребованные профессии
- Профессии, для которых нужна определенная совокупность навыков
- Наиболее подходящие вакансии для соискателя
- Различные характеристики состояния рынка труда и его изменение
- Выделение кластеров профессий и соискателей

Граф  $G = (W, E)$  — двудольный, если множество его вершин можно разбить на два непересекающихся подмножества:  $A \cup B = W$ ,  $|A| > 0$ ,  $|B| > 0$ , причем каждое ребро имеет начало в  $A$ , а конец в  $B$ .

Граф  $G = (W, E)$  — ориентированный и двудольный, если его неориентированный двойник — двудольный граф.

В данной задаче элементами множества вершин  $A$  будут соискатели, а элементы множества вершин  $B$  — работодатели. Основной задачей является оптимальное распределение вакансий между кандидатами. Так как граф является двудольным, то можно сказать, что стоит проблема о марьяже (stable marriage problem): Элементы каждого множества ранжируют элементы

другого множества числом от 1 до  $n$  в зависимости от своих предпочтений, затем пару связывают таким образом, что нет элементов из другого множества, которые хотели бы иметь партнера из этой пары вместо своего настоящего. Такие связи называют устойчивыми (stable). Задачу можно поставить следующим образом:

Найти максимальное паросочетание в двудольном графе, то есть выбрать наибольшее количество ребёр, таких что ни одно из них не имело бы общей вершины с другим ребром.

Для решения этой задачи можно использовать итеративный алгоритм Гейла-Шепли [8]. Сначала необходимо определить способ ранжирования элементов из другого множества. Затем для каждой вершины формируется список предпочитаемых элементов другого множества. После этого начинает работу непосредственно алгоритм, каждая итерация включает в себя стадию предложения, согласия и отказа:

1. Каждый несвязанный ни с кем элемент из множества соискателей предлагает свою кандидатуру, на основе своих интересов, наиболее предпочитаемому работодателю, к которому он ещё не обращался (независимо от того, связан ли этот работодатель с кем-то из соискателей или нет);
2. Каждый работодатель отвечает согласием кандидату, которого он наиболее предпочитает (причем такой кандидат может быть выбран на предыдущей итерации) и отклоняет предложения остальных.

Сложность такого алгоритма  $O(n^2)$ , где  $n$  – число работодателей или соискателей.

Число работодателей и соискателей не равно друг другу. Кроме того, в результате работы алгоритма может получиться так, что на вакансию претендуют два кандидата с одинаковой оценкой. Из-за этого может получиться так, что не все вершины одного из множеств не будут связаны с другим. Алгоритм Гейла-Шепли гарантирует, что количество паросочетаний будет, как минимум, равен половине размера оптимального паросочетания.

Для того чтобы найти как можно больше паросочетаний, можно модифицировать алгоритм:

1. Вершины, которые предпочитает соискатель, будут пройдены дважды;
2. Соискатель будет предпочитать незанятую позицию занятой, если он имеет к ним одинаковый интерес;
3. Если работодатель принял кандидата, у которого есть более предпочитаемая вакансия, он будет дальше принимать новые предложения и, в дальнейшем, может от него отказаться. В этом случае соискатель не уберет данную вакансию из своего списка.

Используя эти модификации, алгоритм гарантирует, как минимум,  $2/3$  от размера оптимального паросочетания.

Исходя из написанного выше, следует, что для работы алгоритма нужно создать RDD, в котором будут храниться список предпочитаемых вакансий, а также статус соискателя. Это же нужно сделать для работодателей.

Так как алгоритму неважен порядок, в котором соискатели предлагают свои кандидатуры или работодатели выбирают наиболее предпочтительного работника, он может выполняться в два этапа: запроса и ответа, что упрощает его запуск на Apache Spark.



Этапы работы алгоритма Гейла-Шепли:

1. Все непринятые соискатели отправляют свои предложения на наиболее предпочитаемую вакансию, а работодатели обрабатывают их, выбирая наиболее предпочитаемое и меняя свой статус;
2. Работодатели отправляют свои решения, а соискатели, если нужно, выбирают из них наиболее предпочтительное и меняют свой статус.

Для того чтобы добавить сюда модификации, описанные выше, не нужно менять общую схему работы алгоритма. Изменения вносятся в то, как обновляются статусы участников, и как они делают предложение и отвечают на него. Так, ещё не выбравший кандидата работодатель отправляет сообщение всем элементам из его списка, оповещая их о том, что вакансия свободная. В свою очередь, соискатель отправляет специальный запрос на вакансию, которую он предпочитает выбранной ранее. Данные изменения не несут большой вычислительной нагрузки, поэтому сложность алгоритма остается такой же.

В статье [11] рассмотрена работа такого алгоритма на данных, которые состоят из двух множеств по 1000 элементов. Для полного завершения работы ему понадобилось примерно 700 итераций.

## **2.2. Разбор резюме и вакансий**

Связь соискателя и работодателя главным образом определяется наличием или отсутствием нужных навыков. Исходя из структуры данных, представленной выше, для них есть отдельное поле, как в вакансиях, так и в резюме. Таким образом, можно проверить наличие у кандидата навыков,

которые требуются для данной работы, после чего поставить ему оценку и, если она выше пороговой, добавить в список предпочтений для дальнейшего рассмотрения. Однако довольно часто работодатели оставляют такое поле пустым, а все необходимые навыки находятся в описании вакансии или в поле с требованиями, причем как часть предложения, а не списком. То же самое относится к резюме. Это сильно усложняет задачу, так как для её решения необходимо применять методы NLP, в частности Text Mining.

Для того чтобы решить эту проблему, можно использовать векторное представление слов. Его идея заключается в том, чтобы сопоставить словам и фразам элементы векторного пространства размерности  $n$ , где  $n$  значительно меньше количества уникальных слов в данном корпусе. Для построения такого представления существует несколько подходов:

Получая на вход большую коллекцию документов, в данном случае файлы с текстом резюме и вакансий, Word2Vec возвращает представление уникальных слов этого корпуса в векторном пространстве. Эти вектора располагаются в нём таким образом, что слова, появившееся в похожем контексте, находятся в непосредственной близости. Такая особенность позволяет сравнивать предложения и термины, используя удобные метрики, например, cosine simalrity.

### **2.3. Использование Word2Vec и Doc2Vec**

Word2Vec — это набор моделей, принимающих на вход текст и получающих в результате работы представление слов в векторном пространстве на основе контекста. Эти модели (Continous-Bag-Of-Words и skip-gram) представляют собой нейронную сеть, задачей которой является реконструкция контекста слов [8]. Так, задача CBOW – предсказание слова

на основании контекста, а задача skip-gram – предсказать контекст на основе единственного слова. Их архитектуру можно посмотреть на рисунке 7

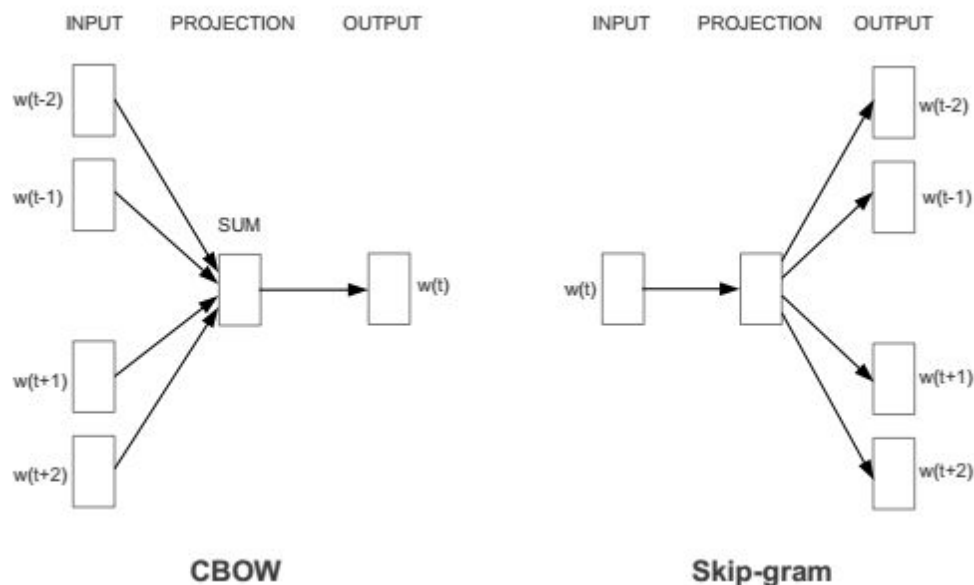


Рис. 7 Архитектура CBOW и skip-gram

Размер векторного пространства  $R_n$  задается вручную, обычно  $n$  находится в диапазоне от 100 до 400. Таким образом, данный метод имеет неоспоримое преимущество в виде небольшой размерности векторов. В отличие, например, от методов, которые работают со словарями, где размерность может достигать нескольких тысяч.

Принцип работы Word2Vec можно описать следующим образом: максимизация косинусной близости для векторного представления слов, которые появляются в похожих контекстах, и, наоборот, её минимизация для слов, не встречающихся в похожих контекстах.

После того, как векторные представления получены, появляется возможность, например, находить близость между двумя словами, получать список наиболее близких элементов в векторном пространстве и так далее. Кроме того, можно получать вектора для целых предложений, используя, например, усредненный вектор всех слов в нём. Однако данный подход

игнорирует порядок слов. Поэтому для работы с предложениями, параграфами или целыми документами, следует использовать Doc2Vec.

В отличие от Word2Vec, Doc2Vec использует две модели: Distributed Memory и Distributed-Bag-Of-Words. Distributed Memory сопоставляет каждому параграфу или предложению вектор, который содержит в себе вектор этого параграфа или предложения, а также все слова, содержащиеся в нём. Таким образом, этот метод предсказывает слово на основании вектора параграфа, что позволяет учесть порядок слов, и по предшествующим словам. DBOW, в свою очередь, предсказывает появление случайных слов в параграфе только на основании вектора параграфа. Архитектура обеих моделей показана на рисунке 8 и 9 [7].

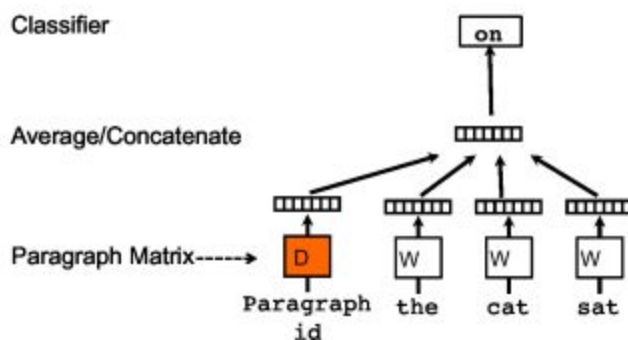


Рис. 8 Архитектура DM

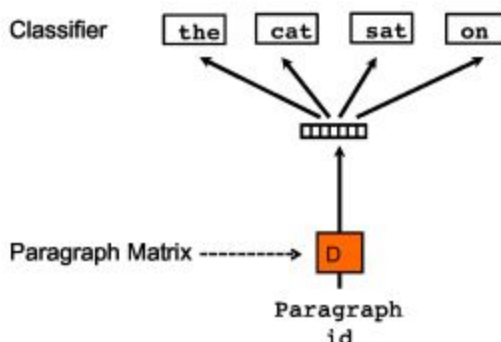


Рис. 9 Архитектура DBOW

Исходя из информации, представленной выше, следует, что для данной

задачи Doc2Vec подходит гораздо лучше, потому что слова (навыки) придется сравнивать с предложениями (требованиями) или абзацами (описание вакансии, опыт кандидата).

Для того чтобы использовать Doc2Vec, можно взять модель, обученную, например, на корпусе Википедии, или же обучить её самому. Недостаток уже готовой модели в том, что она может быть слишком общей и, соответственно, считать ненужные слова близкими друг к другу. Это показано в [1]: обучение на специальном корпусе и обучение на нем же с добавлением страниц Википедии не дало ощутимой разницы. Таким образом, исходным корпусом можно считать все собранные вакансии и резюме.

В качестве входных данных Doc2Vec может принять как предложения (одна строчка входного файла – предложение), так и целые документы (одна строчка входного файла – документ). После обучения модели можно получить векторное представление для слова, предложения или документа, которых не было в тренировочном корпусе. Однако при довольно большом количестве новых вакансий и резюме, её все-таки стоит переобучить.

Таким образом, имея готовую модель, можно получать векторное представление как слов и предложений, так и целых документов. При сравнении текста вакансии и резюме это может использоваться следующим образом:

1. При отсутствии в тексте резюме явного указания имеющихся навыков, можно добавлять их в ассоциирующийся с ним массив, если таковые были выделены при сравнении текста резюме с текстом вакансии;
2. Обработка предыдущего опыта кандидата и выделение из него навыков, требуемых для данной вакансии;
3. Прохождение по тексту резюме, представленному в свободной форме, осуществляя поиск указанных в профессии навыков.

Все это может быть осуществлено благодаря возможности сравнения между собой как предложений, так и слов с предложением или документом.

## Глава 3. Подготовка данных

### 3.1. Описание структуры данных

Так как базу данных резюме не удалось найти в свободном доступе, было решено собрать такую информацию с сайта hh.ru. Для этого был написан парсер, который собирает все доступные в данный момент резюме и сохраняет их в формате JSON.

Для каждого резюме существуют следующие поля:

1. `about` — информация о соискателе в свободной форме, например, его особенности, желаемые условия, ключевые навыки и т. д.
2. `skills` — список ключевых навыков
3. `position_salary` — желаемая зарплата
4. `position_specialization` — специализация
5. `position_title` — желаемая должность
6. `position_schedule` — желаемый график работы
7. `position_time` — занятость: полная, неполная
8. `personal` — пол, возраст, место жительства
9. `cit_time` — желательное время пути до работы
10. `cit_cit` — гражданство
11. `cit_permission` — разрешение на работу
12. `experience` — опыт работы: должность, время работы, краткое описание
13. `education` — образование
14. `link` — ссылка на резюме

Вакансии для тестирования также брались с сайта hh.ru с помощью API. Основным ограничением было 100 результатов на запрос. Вакансии представлены в формате JSON, который имеет следующие поля:

1. описание
2. требования к кандидату
3. обязанности кандидата
4. адрес
5. информация о компании
6. з/п
7. ссылка на полный текст

Для каждой вакансии можно посмотреть её описание, условия работы, занятость и т. д. Исходя из формата полученной информации, были выбраны методы её обработки.

#### **Вакансии:**

Для таких полей, как адрес, должность, з/п, профессиональная область и т. д. дополнительной обработки не требуется, потому что они сохраняются в стандартном виде.

Поля с описанием вакансии, требований к кандидатам, их обязанностями и условиями были обработаны следующим образом. Сначала они были разбиты на предложения и сохранены в виде списков, где каждый элемент — отдельное предложение. Затем в каждом элементе (предложении) была удалена пунктуация, он был разбит на отдельные слова. После был применен стемминг.

К ключевым навыкам также был применен стемминг, и они были сохранены как отдельный список.

#### **Резюме:**

Файлы с резюме были обработаны тем же способом, что и вакансии, за исключением поля с опытом соискателя. Если оно не пустое, то из него извлекаются должности, на которых раньше работал соискатель, название компании, профессиональная область и описание.



## 3.2. Работа с данными в Apache Spark

Apache Spark предоставляет удобный инструмент для работы с данными такого типа — DataFrame. Это распределенная коллекция, где данные сохраняются в именованные столбцы, поэтому работа с Apache Spark DataFrame похожа на работу с таблицами в реляционных базах данных.

Для того, чтобы создать DataFrame из JSON, необходимо:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
vacancies = sqlContext.read.json("vacancies.json")
```

Стоит отметить, что на JSON файл накладываются определенные ограничения, так, например, попытка считать файл, в котором значение одного из полей занимает несколько строчек, скорее всего закончится неудачей.

После того, как DataFrame создан, можно обращаться к его столбцам, например, следующим образом:

```
vacancies .select("position_title").show() — вывести все значения столбца
'position_title'
```

```
vacancies .filter(vacancies ["position_title"] = "Аналитик").show()
```

## Глава 4. Реализация

### 4.1. Классификация документов и ключевых навыков

Некоторые резюме и вакансии написаны на иностранном языке. Сделать систему, которая сможет сравнивать два документа на разных языках очень сложная задача. В этом направлении было проведено несколько исследований [2], [3]. Однако в данной работе документы, содержащие определенный процент языка отличного от русского, будут игнорироваться.

Для начала были созданы два `DataFrame IndexesVac` и `IndexesRes`, в которых будут два столбца: индекс и документ. Сначала загружаются все вакансии в `RDD`, затем к нему применяется функция `zipWithIndex`. Те же действия проводятся для файлов с резюме.

Перед тем, как начать обработку текста, необходимо собрать все доступные ключевые навыки и классифицировать их, чтобы ускорить дальнейшую работу. В качестве категорий можно взять данные поля `specializations.profarea_name`, которые есть в вакансиях и резюме, причем вакансиям стоит отдавать предпочтение. Таких категорий существует 28 штук. После того, как собран список ключевых навыков для каждой категории, можно разбить его на три класса:

1. Общие для всех категорий навыки;
2. Общие для кластера навыки;
3. Особые навыки отдельной профессии.

Решение этой задачи можно провести в несколько этапов:

1. Вручную создать список, в котором будут содержаться все 28 профессиональных областей;
2. Прочитать все файлы с вакансиями;

3. Каждый элемент `vacancies` проверить на наличие информации в поле с ключевыми навыками и, если они там есть, то определить к какой области и должности они относятся;
4. Получить RDD, количество элементов которого равно вакансиям, состоящий из идентификатора профессиональной области, должности и списка навыков;
5. Из RDD, полученного на предыдущем шаге, получить список всех профессий для каждой из профессиональных областей;
6. Используя `join`, получить RDD, в которых будут все навыки для данной профессиональной области или профессии;
7. На основании RDD с навыками профессиональной области, отфильтровать список всех ключевых навыков, используя `join`. В результате останутся те из них, которые есть во всех профессиональных областях;
8. Повторить действия 7 шага, но для RDD с навыками профессиональной области и отдельной должности;
9. Найти частоту появления навыков в документах.

Данная классификация позволяет эффективнее сравнивать вакансии и резюме, исключая ненужные навыки и документы.

Следующим этапом будет классификация всех документов по профессиональным областям и должностям. Для этого необходимо создать два новых `DataFrame` на основе `IndexesVac` и `IndexesRes`. Перебирая их элементы, происходит обращение к полям документа, и в новые столбцы добавляются данные о профессиональной области и должности. Эти два `DataFrame` называются `VacSpecPos` и `ResSpecPos`.

Таким образом, при дальнейшем анализе документов ограничивается множество всевозможных навыков, исключая те из них, которые характерны для других профессиональных областей и должностей.

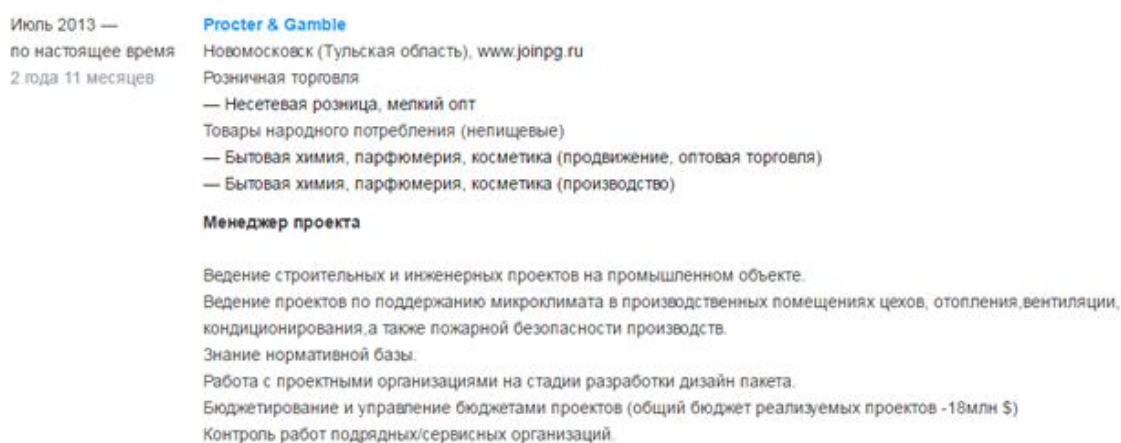
Кроме того, имея данную классификацию и частоту появления навыков в вакансиях, можно отслеживать изменение трендов рынка труда.

## 4.2. Анализ резюме

Структура для хранения извлеченных данных для резюме и вакансии, имеет свои сходства и различия. Общим будет наличие таких полей, как дата размещения, специализация, должность, адрес, з/п, тип занятости. Их несоответствие сразу говорит о том, что кандидат вряд ли подойдет для данной работы.

Полученная в предыдущем параграфе классификация навыков и документов будет использоваться в дальнейшем разборе резюме. Целью является выделение из полей с информацией об опыте работы, личной информацией ключевых навыков соискателя.

Для резюме особенно важно анализировать предыдущий опыт работы кандидата, учитывая время, проведенное на какой-либо должности и его обязанности. Пример того, как оформляется опыт соискателя, можно найти на рисунке 10.



Июль 2013 — по настоящее время 2 года 11 месяцев	<b>Procter &amp; Gamble</b> Новомосковск (Тульская область), <a href="http://www.joinpg.ru">www.joinpg.ru</a> Розничная торговля — Несетевая розница, мелкий опт Товары народного потребления (непищевые) — Бытовая химия, парфюмерия, косметика (продвижение, оптовая торговля) — Бытовая химия, парфюмерия, косметика (производство) <b>Менеджер проекта</b> Ведение строительных и инженерных проектов на промышленном объекте. Ведение проектов по поддержанию микроклимата в производственных помещениях цехов, отопления, вентиляции, кондиционирования, а также пожарной безопасности производств. Знание нормативной базы. Работа с проектными организациями на стадии разработки дизайн пакета. Бюджетирование и управление бюджетами проектов (общий бюджет реализуемых проектов -18млн \$) Контроль работ подрядных/сервисных организаций.
--	--

Рис. 10 Опыт соискателя

Полезной информацией здесь является компания, её специализация, должность, время работы на ней, а также описание. Стоит отметить, что для поля с описанием нет каких-либо стандартов, текст в нём представлен в свободной форме, поэтому для его анализа нужно использовать векторное представление ключевых навыков.

Поле «О себе» также может содержать информацию об опыте, навыках и личностных качествах. Оно не имеет стандартов, поэтому для его анализа также будет использована модель Doc2Vec.

Извлечение информации из документа производится в несколько шагов во время прохода по IndexesRes: Осуществляется доступ к строке в DataFrame и преобразование её в RDD, где один элемент — массив, состоящий из:

1. Индекса документа;
2. Личной информации;
3. Даты размещения;
4. Профессиональной области;
5. Должности;
6. Адрес;
7. З/п;
8. Типа занятости;
9. Образования;
10. Информация о знании языков;
11. Указанных ключевых навыков;
12. Массива с опытом работы, элементы которого — это массивы, хранящие в себе время работы на данной должности, должность, компанию, профессиональную область и описание в виде списка предложений;
13. Массива со списком предложений из поля “О себе”

14. Навыков, которые удалось извлечь из опыта работы;

15. Навыков, которые удалось извлечь из поля “О себе”;

Стоит отметить, что элементы с 1 по 10 не нужно никак обрабатывать, они сохраняются в начальном виде. К ключевым навыкам, указанным в специальном поле, был заранее применен стемминг. Элементы 12 и 13 получены в результате совмещения информации из нескольких полей JSON файла. Элементы 14 и 15 получаются в результате применения к каждому предложению функции, в основе которой лежит нахождение косинусного сходства между этим предложением и всеми ключевыми навыками, характерными для данной профессиональной области и должности.

### **4.3. Анализ вакансии**

Анализ вакансии происходит по той же схеме, что и анализ резюме.

Отличием здесь являются только поля, полученные в результате разбора:

1. Индекс документа
2. Дата размещения;
3. Профессиональная область;
4. Должность;
5. Адрес;
6. З/п;
7. Тип занятости;
8. Опыт работы;
9. Ключевые навыки;
10. Список предложений с описанием вакансии;
11. Список предложений с требованиями к кандидату;
12. Навыки, которые удалось извлечь из описания вакансии;
13. Навыки, которые удалось извлечь из требований к кандидату;

#### **4.4. Определение связей между соискателями и вакансиями**

Определение того, что вакансия подходит данному кандидату осуществляется в несколько этапов. Сначала внутри каждой профессиональной области происходит сравнение каждого соискателя с каждой вакансией по следующим полям: должность, адрес, тип занятости, наличие необходимого опыта работы в данной сфере (только срок), з/п. Если хотя бы по одному из этих полей обнаружено несовпадение, то эта пара больше не рассматривается. Если же все поля удовлетворяют друг другу, то индекс данной вакансии добавляется в RDD, состоящий из индекса резюме соискателя и подходящих вакансий. Такой же RDD формируется для каждой вакансии.

Следующим этапом будет прохождение по каждому элементу списка в RDD и определение оценки того, насколько вакансия подходит кандидату, и наоборот. Если эта оценка переходит определенный порог, то вакансия или кандидат добавляются в список предпочитаемых связей, который будет в дальнейшем использован для нахождения оптимального паросочетания. Оценку и порог можно менять в зависимости от того, что более важно работодателю. В данной работе все найденные навыки, вне зависимости от того, как они указаны в документе, учитывались одинаково. Оценка выставлялась как отношение размера множества пересечения навыков резюме и вакансии к размеру множества объединения навыков резюме и вакансии. За порог взято число 0.6.

## 4.5. Построение графа

Для работы с графами Apache Spark предоставляет библиотеку GraphX. Она позволяет создавать и работать с ориентированными мультиграфами, в которых каждой вершине или каждому ребру может быть поставлено в соответствие какое-то свойство.

Плюсами этой библиотеки являются:

1. Вычисления над графом выполняются параллельно, распределенные между узлами кластера;
2. Удобство работы;
3. Возможность рассматривать данные как граф и как коллекции;
4. Выполнение операций над графами с той же эффективностью, что и над RDD;
5. Готовые алгоритмы.

Минусами, в свою очередь, можно назвать:

1. Невозможность динамически обновлять граф, добавлять или удалять вершины и ребра;
2. Граф существует, пока он загружен в память.

GraphFrames является пакетом для Spark, который, в отличие от GraphX, позволяет строить графы на основе DataFrame, а не RDD. Можно сказать, что он расширяет возможности GraphX, делая доступными сериализацию, основанную на DataFrame, а также выразительные запросы к графу. Однако, его минусом, как и в GraphX, остается невозможность динамически обновлять граф.

Невозможность динамически обновлять граф компенсируется тем, что операция его создания из DataFrame требует довольно немного ресурсов.



В результате сравнения этих библиотек была выбрана библиотека GraphX, потому что её возможностей хватает для решения данной задачи. К тому же она имеет более богатую документацию. GraphFrames, хотя и являются перспективным проектом, пока находится в начальном состоянии, поэтому с ним могут возникнуть проблемы.

Для того, чтобы создать граф, требуются два DataFrame, один из которых будет содержать в себе вершины, а другой — ребра. Вершинами в данном случае являются соискатели и компании, а связи между ними определяются тем, насколько кандидат подходит компании, и наоборот.

На основании списков предпочтения, полученных на предыдущем этапе, можно построить двудольный граф, описанный в главе 2. Для этого используется библиотека GraphX. Узлами графа являются соискатели и вакансии, у каждого из которых есть свойство — список предпочитаемых узлов из другого множества.

После того, как граф построен, к нему применяется алгоритм, также описанный в главе 2. Результатом его работы является оптимальное паросочетание, которое должно распределить соискателей по вакансиям максимально эффективно для обеих сторон.

Благодаря тому, что ранее были построены списки предпочтения, работодатели могут вручную выбрать кандидатов, которые остались незаняты после работы алгоритма.

## Выводы

На основе полученных результатов можно заявить, что метод нахождения оптимальных связей между соискателями и работодателями, рассмотренный в данной работе, довольно хорошо справляется со своей задачей. В случае, когда имеется большое количество текстовых данных без какой-либо обучающей выборки, использование векторного представления для слов и предложений имеет неоспоримое преимущество. С его помощью можно, не проводя перед этим сложную обработку данных, разбить документы на заранее определенные классы. Точность данной модели является допустимой для решения рассмотренной здесь задачи.

Было показано, что алгоритм для нахождения множества оптимальных паросочетаний, описанный в главе 2, может применяться в рассмотренной области. В корректировке нуждается формула оценки важности извлеченных навыков.

Реализация описанных выше алгоритмов с использованием Apache Spark является рабочей, но требует оптимизации. В частности, при извлечении полезной информации из документов, используется слишком много промежуточных переменных. Для оптимизации использования памяти требуется обозначать какие из них должны оставаться в памяти, а какие можно удалить.

## **Заключение**

В результате работы было разработано программное обеспечение для сбора данных о вакансиях и резюме. Полученные данные были обработаны, приведены к более удобной для дальнейшего анализа форме и сохранены в формате JSON. Затем были определены возможные подходы извлечения полезной информации из слабоструктурированных данных, проведено их сравнение и выбран наиболее подходящий для данной работы вариант. Был разработан алгоритм, основанный на векторном представлении слов, который позволяет анализировать обработанные данные и находить связи между сущностями. На основе найденных связей был построен граф, на котором был выполнен алгоритм по нахождению наиболее оптимального множества паросочетаний. Все эти алгоритмы рассматривались в контексте фреймворка для обработки Big Data — Apache Spark. Можно считать, что все поставленные задачи в результате работы выполнены.

## **Дальнейшая работа**

Для дальнейшего улучшения эффективности рассмотренных здесь алгоритмов необходимо провести работу в следующих направлениях: решение вопроса о хранении полученных данных; применение методов машинного обучения над полученной моделью; рассмотрение социальных контактов соискателей для более эффективного и успешного поиска вакансии.

## Список литературы

1. Carsten Lygteskov Hansen, Melanie Tosik, Gerard Goossen, Chao Li, Lena Bayeva, Florence Berbain, Mihai Rotaru How to Get the Best Word Vectors for Resume Parsing // 2013.
2. Enda Liu, Tomoki Ito, Kiyoshi Izumi Comparison among multilingual financial words using the word2vec and clustering with news resources for automatic creation of financial dictionaries // 2016.
3. Manaal Faruqui, Chris Dyer Improving Vector Space Word Representations Using Multilingual Correlation // 2014.
4. Marc A. Smith NodeXL: Simple network analysis for social media // 2013.
5. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing // 2012.
6. Michal Campr, Karel Jezek Comparing Semantic Models for Evaluating Automatic Document Summarization // 2015.
7. Quoc Le, Tomas Mikolov Distributed Representations of Sentences and Documents // 2014.
8. Reynold S. Xin, Joseph E. Gonzalez, Michael J. Franklin, Ion Stoica GraphX: A Resilient Distributed Graph System on Spark // 2013.
9. Sumit Maheshwari, Abhishek Sainani, P Krishna Reddy An Approach to Extract Special Skills to Improve the Performance of Resume Selection // 2010.
10. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean Distributed Representations of Words and Phrases and their Compositionality // 2013.

11. Yilong Geng, Mingyu Gao Distributed Stable Marriage with Incomplete List and Ties using Spark // 2015.