Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
Национальный исследовательский университет
"Высшая школа экономики"

Отделение программной инженерии
Кафедра Управления разработкой программного обеспечения

УТВЕРЖДАЮ
Зав. кафедрой УРПО

_____ С.М. Авдошин
«__» _____ 2014 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА по направлению 231000.62 Программная инженерия подготовки бакалавра

На тему «Программа анализа текстов на основе модифицированного метода аннотированных суффиксных деревьев»

Студент группы №471ПИ _____ Дубов Михаил Сергеевич
подпись

_____
(Дата)

Научный руководитель

_____ _____ Миркин Борис Григорьевич
(должность, звание) подпись

_____
(Дата)

Москва, 2014 г.

Реферат

Отчет 35 страниц, 4 главы, 7 иллюстраций, 2 таблицы, 35 источников, 5 приложений.

Ключевые слова: *анализ текстов, алгоритмы над строками, аннотированные суффиксные деревья, суффиксные массивы, выделение синонимов, концептуальные графы.*

Объектом разработки является программа статистического анализа текстов на основе модифицированного метода аннотированных суффиксных деревьев (АСД). Будучи многообещающей методологией анализа текстов с многочисленными оригинальными применениями, традиционный метод АСД показал свою недостаточную эффективность в смысле потребления его базовыми алгоритмами вычислительных ресурсов (времени и памяти) при использовании в работе с большими по объему коллекциями текстов. Кроме того, эксперименты на реальных текстах показали, что метод АСД в своем базовом виде нередко дает нерелевантные оценки степени вхождения ключевых словосочетаний в анализируемые корпуса текстов из-за отсутствия в нем поддержки учета языковых особенностей анализируемых текстов, прежде всего – синонимов.

Целью работы является разработка модификации исходного метода АСД, позволяющей преодолеть указанные выше его недостатки, а также создание программного продукта, реализующего соответствующие алгоритмы. Ряд экспериментов показал, что новая программная реализация метода АСД превзошла использовавшиеся до этого аналоги в отношении как объема используемой памяти (достигается десятикратная ее экономия), так и времени работы основных алгоритмов. Эти результаты были достигнуты за счет использования в программной реализации вместо суффиксных деревьев более эффективной альтернативной структуры данных – суффиксных массивов.

Программный продукт распространяется с открытым исходным кодом под лицензией MIT как Python-пакет EAST. Он также официально зарегистрирован в системе Python Package Index. Пакет не только может использоваться в качестве Python-библиотеки в других проектах, но и предоставляет конечному пользователю удобный интерфейс командной строки для работы с анализируемыми текстами.

В настоящее время продукт успешно внедрен в систему сбора и анализа газетных статей, разрабатываемую НУГ «Методы анализа и визуализации текстов» НИУ ВШЭ.

Дальнейшие направления работы включают в себя уточнения конфигурации алгоритма выделения синонимов, а также расширение круга практических приложений метода АСД, поддерживаемых нашим ПО.

Содержание

Введение

Метод *аннотированных суффиксных деревьев* (АСД) был впервые предложен в 2006 г. как оригинальный подход к классификации текстов с применением к задаче фильтрации спама [15]. В основе метода лежит первоначальная индексация текста (или коллекции текстов) путем построения для него суффиксного дерева [4, с. 89-93] специального вида с последующим вычислением оценки степени вхождения в данный текст некоторого набора ключевых словосочетаний. Метод АСД при этом отличается от большого количества широко используемых сегодня подходов к анализу текста тем, что он рассматривает входные тексты как последовательности символов, а не как последовательности слов. В то время как вторая из упомянутых выше стратегий довольно широко встречается как в литературе, так и в практических приложениях (например, в виде модели «мешка слов» [17]), первая имеет преимущество в виде большей независимости от особенностей языков, на которых написаны анализируемые тексты [25]. Действительно, работа с текстами как с последовательностями символов сильно понижает (но не уничтожает полностью) необходимость в ряде шагов их предобработки, например, лемматизации (выделения основ), так как результаты работы соответствующих алгоритмов, как правило, мало зависит от грамматических форм встречающихся в тексте слов. Точно так же уходит необходимость в нередко весьма трудоемком шаге подготовки списка стоп-слов и фильтрации текстов на его основе – метод АСД, работая с отдельными символами текста, мало чувствителен к встречающемуся в нем «шуму». Наконец, важным свойством такого подхода является тот факт, что он учитывает последовательную природу анализируемых текстов [16], что отсутствует, например, в выше упомянутой модели представления текстов в виде «мешка слов».

Интерес к методу АСД повысился с тех пор, как в литературе был предложен ряд оригинальных его приложений. Среди наиболее значимых разработок в этой области назовем использование аннотированных суффиксных деревьев в проблеме классификации текстов [16] и извлечения из них наиболее значимых слов, кроме того, основанный на этой структуре данных подход к иерархической кластеризации текстов, а также использование ее для построения графов связей между ключевыми словосочетаниями на основе анализа входной коллекции текстов [25]. Алгоритм построения такого «концептуального графа» основывается на анализе совместной встречаемости ключевых словосочетаний в текстах данного корпуса (ключевое словосочетание считается встречающимся в некотором тексте, если оценка степени его вхождения в этот текст, вычисленная с помощью соответствующего АСД, превышает некоторый порог).

Серьезным недостатком рассматриваемого метода является его низкая производительность в случае наивной программной реализации. В последнее время была отмечена квадратичная (относительно длины входной совокупности текстов) сложность предлагаемого в оригинальной работе [15] алгоритма построения аннотированного суффиксного дерева и квадратичный же объем памяти ЭВМ, необходимый для его хранения. Также был предложен алгоритм, позволяющий осуществлять построение дерева за линейное время и таким же образом снижающий требования к объему используемой памяти [24]. Предложенный алгоритм является модификацией широко используемого в программных реализациях алгоритма Укконена [20]. Вслед за предложенной модификацией этого алгоритма последовала ее программная реализация, которая была опубликована [23]. Однако даже будучи линейным по сложности, данный подход все еще страдает от избыточного потребления памяти (около 30 байт в среднем на один символ входного текста), а также от плохой локальности в памяти, так как суффиксное дерево является не последовательной, но связной структурой данных. Второй недостаток, в частности, не позволяет воспользоваться преимуществами современных процессорных архитектур, предполагающих широкое использование кэшей [1] и, таким образом, сильно снижает производительность используемых алгоритмов. В результате данный подход к программной реализации метода АСД представляется недостаточно эффективным для применения исследователями на практике при работе с большими текстовыми корпусами.

Общепризнанной альтернативой суффиксным деревьям является структура данных под названием *«суффиксный массив»* [13]. Она не только хранится в памяти последовательно, но также значительно снижает требования к объемам используемой памяти (4 байта на один символ входного текста в простейшем случае). Относительно недавно было показано, что при оснащении простейшего суффиксного массива несколькими дополнительными массивами (при аккуратной программной реализации позволяющими сохранять требования к памяти на уровне около 10 байт на один символ) возможно адаптировать к этой структуре данных любой алгоритм, использующий суффиксные деревья [1]. Возможно это и в случае аннотированных суффиксных деревьев, где реализация соответствующих алгоритмов обработки текста с использованием суффиксных массивов позволяет более чем в два раза снизить объем используемой памяти и улучшить скорость их работы за счет последовательного хранения используемых структур данных в памяти. Именно это и сделано в данной работе.

Другим важным направлением улучшения метода АСД является повышение его чувствительности к языковым особенностям анализируемых текстов, в первую очередь – к

встречающимся в них синонимам. Ряд экспериментов [26] показал, что неучет встречающихся в тексте синонимов обычно сильно понижает релевантность строимых предложенным в [25] алгоритмом графов связей между анализируемыми ключевыми словосочетаниями. Действительно, легко можно представить себе текст, посвященный, скажем, классификации растений, автор которого изредка использует в качестве синонима слову *«классификация»* понятие *«таксономия»*. Оценка вхождения в такой текст словосочетания *«таксономия растений»* будет в этом случае скорее всего значительно ниже, чем для словосочетания *«классификация растений»*, что, однако, неправильно с семантической точки зрения.

Методы выделения синонимов из текстов варьируются начиная с алгоритмов, основанных на анализе контекста, в котором встречаются соответствующие слова [11], и заканчивая подходами, обрабатывающими тезаурусы [21]. В нашей работе мы используем первый подход, так как он, на наш взгляд, повышает релевантность выделяемых синонимов относительно предметной области анализируемых текстов, а также позволяет избежать выделения избыточного числа синонимов, что снизило бы производительность метода АСД.

Таким образом, в нашу задачу входит разработка такой модификации метода АСД, которая бы позволяла использовать преимущества метода суффиксных массивов и учета синонимов одновременно.

Цель работы

Целью работы является создание программного продукта, реализующего указанную модификацию метода АСД и ориентированного на работу с русскоязычными текстами. Помимо программной реализации «ядра» рассматриваемой методологии (индексации текстов и последующего вычисления оценки степени вхождения в них ключевых словосочетаний) конечный продукт также должен предоставлять пользователю возможность осуществлять построение графов связи между ключевыми словосочетаниями [25], реализуя таким образом одно из наиболее интересных практических приложений данного метода.

Помимо собственно разработки приложения, целью работы является его внедрение в программное обеспечение для сбора и анализа текстов новостных публикаций из Web-ресурсов, работа над которым ведется в рамках Научно-учебной группы НИУ ВШЭ «Методы анализа и визуализации текстов» [27]. Данное программное обеспечение, в

частности, планируется использовать для построения графов связей между ключевыми словосочетаниями (заданными пользователем или же автоматически выделенными из анализируемого корпуса статей) по методу АСД с целью их последующей интерпретации экспертами.

1. Обзор

1.1. Структуры данных и алгоритмы над строками

Суффиксное дерево является одной из центральных структур данных в области построения и анализа алгоритмов над строками [4, с. 89-93]. Ее значимость проистекает из того факта, что она не только является основой для линейного по времени работы решения одной из классической задач обработки строк – задачи *точного сопоставления с образцом* – но также позволяет проектировать весьма элегантные алгоритмы для решения многих задач в различных областях компьютерных наук, например, в такой актуальной и активно развивающейся области как биоинформатика [7, с. 320-324]. Формально суффиксное дерево для строки S длины n определяется как корневое направленное дерево, кодирующее все суффиксы этой строки: оно строится таким образом, что конкатенация его реберных меток на каждом из путей от корня к одному из листьев образует один из суффиксов входной строки, т.е. подстроку $S[i..n]$. Дополнительные ограничения, накладываемые на данную структуру данных, сводятся к тому, что каждый внутренний узел дерева должен иметь два или более дочерних, а каждое из ребер – должно быть помечено непустой подстрокой S [3, с. 90]. Суффиксное дерево может быть построено для заданной входной строки за время, линейно зависящее от ее длины (например, алгоритмом Укконена [20]) и требует для хранения также линейного количества памяти [22]. *Обобщенным суффиксным деревом* называется суффиксное дерево, построенное для нескольких строк. Оно также может быть построено за время, линейно зависящее от суммы длин этих строк [4, с. 116].

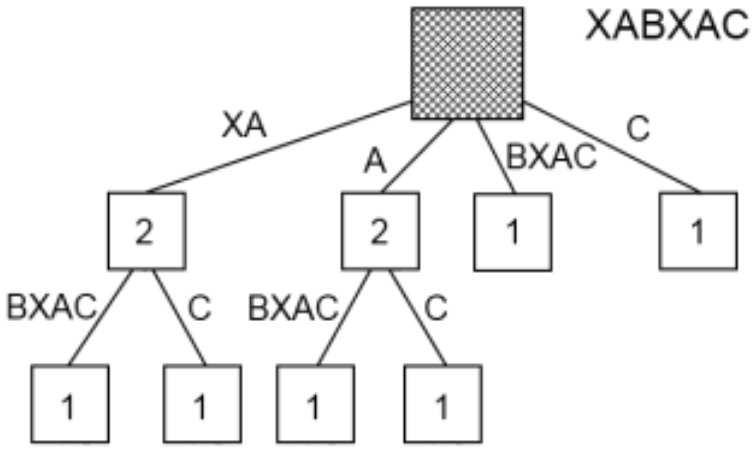

Рисунок 1. Аннотированное суффиксное дерево для строки "XABXAC"

Аннотированное суффиксное дерево является расширением оригинальной структуры данных [15]. При построении данной структуры данных в дополнение к меткам ребер рассчитываются также частоты встречаемости в исходном тексте подстрок,

закодированных в метках ребер на пути от корня к каждому из узлов. Эти частоты сохраняются в узлах дерева как целочисленные значения (на рис. 1 приведен пример построенного АСД для строки “XABXAC”). Такие аннотации узлов позволяют затем вычислять оценку степени вхождения входных строк в исходный текст (оценка является действительным числом в $[0; 1]$). Данная процедура называется *наложением* словосочетания на аннотированное суффиксное дерево [25]. В случае работы с коллекцией текстов осуществляется построение обобщенного аннотированного суффиксного дерева. Важным свойством аннотаций является то, что в каждом из узлов они являются суммой аннотаций дочерних узлов. В [24] было показано, что при соблюдении условия окончания каждого из текстов из коллекции уникальным символом (что легко достигается путем предобработки поступающей на вход совокупности текстов) возможно использование этого свойства для адаптации одного из линейных алгоритмов построения обычного суффиксного дерева (например, алгоритма Укконена). Это позволяет строить аннотированные суффиксные деревья также за линейное время.

Так называемый *метод аннотированных суффиксных деревьев* (метод АСД) является подходом к анализу текстов, состоящем из двух основных шагов. На первом шаге входная коллекция текстов индексируется путем построения для нее аннотированного суффиксного дерева. Это дерево затем используется для вычисления оценок степени вхождения в исходную коллекцию текстов набора ключевых словосочетаний (определенного заранее). Например, при решении задачи фильтрации спама можно построить аннотированное суффиксное дерево для корпуса писем, содержащих спам. На этапе анализа поступающих e-mail сообщений на предмет наличия в них спама их тексты можно разбить на ряд словосочетаний, которые затем накладываются на уже построенное суффиксное дерево. В зависимости от полученных оценок принимается решение, следует ли пометить анализируемое письмо как спам [15].

Суффиксные массивы были предложены спустя 20 лет после появления в литературе суффиксных деревьев как более эффективная в плане использования памяти альтернатива [13]. В то время как суффиксное дерево даже в самой аккуратной своей реализации требует не менее 20 байт на каждый из символов входной строки, суффиксный массив в чистом виде требует всего 4 байта на символ. Формально суффиксным массивом для строки S длины n называется массив из n целых чисел от 0 до $(n-1)$, упорядоченный таким образом, что он перечисляет начальные позиции всех суффиксов строки S в лексикографическом порядке (по возрастанию) [4, с. 149]. Суффиксный массив может быть получен путем обхода уже построенного суффиксного дерева для той же строки; известны

также и алгоритмы, позволяющие строить суффиксные массивы непосредственно на основе одной входной строки за время, линейно зависящее от ее длины (алгоритмы Кярккяйнена и Сандерса [8], а также алгоритм Ко и Алуру [10]), являющиеся, однако, несколько более сложными для восприятия, чем известные линейные алгоритмы построения суффиксных деревьев.

Еще через 10 лет после появления первых результатов, связанных с использованием суффиксных массивов, было показано, что любой алгоритм, использующий в своей работе суффиксные деревья, может быть переписан с использованием суффиксных массивов без потерь в асимптотической временной сложности и объеме используемой памяти [1]. Такая систематическая замена суффиксных деревьев массивами возможна, однако, только при использовании нескольких вспомогательных массивов, в частности, так называемого *lcp-массива*, который хранит информацию о длине наибольших общих префиксов для «соседних» суффиксов и также может быть построен за линейное время. Если в [9] было показано, как использовать lcp-массив для симуляции обхода суффиксного дерева снизу вверх, то в [1] эта идея была развита до концепта так называемого *lcp-дерева*, узлы которого однозначно соотносятся с узлами соответствующего суффиксного дерева. Само *lcp*-дерево фактически при этом не строится и используется лишь как виртуальная структура данных. Обход такого дерева возможен не только снизу вверх, но и сверху вниз – при условии хранения в памяти еще одного вспомогательного *child*-массива. Собственно сама возможность симуляции обхода суффиксного дерева как снизу вверх, так и сверху вниз при использовании суффиксного массива позволяет замещать им эту структуру данных в любых алгоритмах. Вместе с исходным суффиксным вспомогательные массивы (*lcp* и *child*) – при аккуратной программной реализации – требуют не более *10* байт памяти на один символ входной строки, являясь таким образом в 2 раза более эффективным с точки зрения памяти решением, чем суффиксные деревья. Такая структура данных была названа своими авторами *расширенным суффиксным массивом* (*enhanced suffix array*) [1].

Замещение суффиксных деревьев суффиксными массивами является не единственным путем решения проблемы избыточного потребления первыми оперативной памяти ЭВМ. Другим возможным подходом является реализация эффективных, но весьма сложных алгоритмов построения и использования суффиксных деревьев во внешней памяти [2, 14]. Кроме того, разработаны алгоритмы для сжатия самого суффиксного дерева в оперативной памяти [18].

Наконец, отметим существование модификаций предложенного в [1] метода симуляции обхода суффиксных деревьев сверху вниз при использовании суффиксных

массивов – например, в [19] предлагается вместо вспомогательного *child*-массива использовать так называемые *Range Minimum Queries* (RMQ). Данный подход позволяет при некоторой потере во временной эффективности алгоритмов еще более снизить объем потребляемой ими памяти.

В целом, значительным преимуществом решений, основанных на использовании суффиксных массивов, перед подходами, работающими с суффиксными деревьями, является то обстоятельство, что суффиксные массивы являются компактной структурой данных, которая хранится в памяти последовательно – в отличие от суффиксных деревьев, которые являются связной структурой данных. Это позволяет современным процессорам, многие из которых обладают архитектурой, активно использующей кэш при подгрузке данных из памяти, использовать ряд низкоуровневых оптимизаций при работе с суффиксными массивами. Алгоритмы, использующие суффиксные массивы, таким образом, сохраняют высокую эффективность в плане времени их выполнения при более экономном использовании памяти.

1.2. Выделение синонимов

Синонимия является одним из основных лексико-семантических отношений между словами в одном языке (наряду с антонимией, гиперонимией, меронимией и др.). Два слова считаются синонимами, если они обладают (приблизительно) одним значением [21]. Процесс *извлечения синонимов* направлен на автоматическое создание тезауруса какой-либо заранее заданной структуры (например, основанной на графах), к которому можно делать запросы на получение множества синонимов (возможно, с оценкой степени их синонимичности) для данного слова. Как правило, извлечение синонимов осуществляется либо на базе какого-нибудь словаря (например, толкового), либо же на основе входного корпуса текстов (нередко ограниченного текстами в какой-либо одной предметной области).

Стратегия выделения синонимов, основанная на анализе словарей, как правило, включает в себя детальный анализ структуры текст определений в нем слов. В результате этого анализа выявляются связанные между собой определения и затем определяются пары синонимичных слов. Для решения этой задачи можно, например, построить на основе анализа толкового словаря граф определений и применить к нему какой-либо алгоритм ранжирования, получая при этом меры связи между парами вершин [3]. Эти меры связи можно принять за коэффициент синонимичности соответствующих вершинам слов.

Контекстно-ориентированный подход к извлечению синонимов базируется на предположении, что схожие слова должны появляться в схожих контекстах. Примером работы, реализующей эту идею, является [11], где контексты моделируются так называемыми *тройками зависимостей* (w_1, r, w_2), где w_1 и w_2 – слова, а r – грамматическое отношение между ними. На основе частотности встречаемости каждой из этих троек в исходном корпусе текстов рассчитывается специальная метрика связности между парами слов. В результате эта метрика принимает высокие значения для тех пар, слова из которых часто встречаются в одних и тех же тройках зависимостей.

Два описанных выше подхода – основанный на анализе словарей и основанный на анализе контекстов, в которых встречаются слова (статистический) – могут быть скомбинированы для получения метрики семантической связности слов, которая учитывает как энциклопедические знания о них, так и образцы использования этих слов в анализируемых текстах [5].

2. Описание алгоритмов

2.1. Реализация метода АСД с использованием суффиксных массивов

Подход к преобразованию алгоритмов, основанных на использовании суффиксных деревьев, к алгоритмам, работающим с суффиксными массивами, излагается в [1] применительно к суффиксным деревьям в их оригинальном виде. Для того чтобы адаптировать этот подход к аннотированным суффиксным деревьям, которые являются расширением оригинальной структуры данных и содержат больше информации об индексируемом ею тексте, необходимо в первую очередь спроектировать способ хранения аннотаций узлов дерева при переходе к использованию вместо них массивов.

Напомним, что, согласно определению суффиксных деревьев, каждый из их внутренних узлов должен иметь два или более дочерних [4, с. 90]. Кроме того, суффиксное дерево, построенное для строки длины n, по определению имеет ровно n листьев. Из этих двух фактов непосредственно следует ограничение на число внутренних узлов в суффиксном дереве: их число не может превышать $(n-2)$. При соблюдении условия окончания каждого из текстов входной коллекции уникальным символом (что легко достигается путем ее предобработки) каждый из суффиксов строки (совокупности строк) будет встречаться в ней ровно один раз; соответственно, частоты встречаемости (аннотации) в листьях дерева заведомо будут равны 1, и необходимость хранить эти единичные аннотации в явном виде отпадает. Таким образом, необходимо хранить не более $(n-1)$ аннотаций для внутренних узлов и корня дерева. Для хранения этих аннотаций мы можем ввести еще один массив (назовем его *annotation*-массив), который вслед за вспомогательными *lcp*- и *child-массивами*, используемыми при реализации алгоритмов над суффиксными деревьями с использованием суффиксных массивов, будет занимать n ячеек памяти.

В уже упомянутой модели *lcp-дерева*, которое соответствует исходному суффиксному дереву и которое можно восстановить на основе информации из *lcp-массива*, узлы дерева представляются в виде так называемых *lcp-интервалов*, которые задаются тремя параметрами: *lcp-значением* l, а также левой и правой границей интервала (i и j). Нетрудно показать (как это сделано, например, в [1]), что каждому из таких *lcp-интервалов* v (одновременно узлов дерева) можно однозначно сопоставить индекс $index(v)$, равный первому по возрастанию индексу k такому, что $k > i$ и, кроме того, $lcp[k] = l$. Именно этим фактом мы пользуемся для того, чтобы перенести аннотации внутренних узлов суффиксного дерева в ячейки нового *annotation-массива* по соответствующим этим узлам индексам.

Таблица 1

Расширенный суффиксный массив с аннотациями для строки “XABXAC”

i	***суффиксный массив***	***lcp-массив***	***child-массив***			***annotation-массив***	***S[suf[i]:]***
			1.	***2.***	***3.***		
0	1	0		1	2	6	ABXAC
1	4	1				2	AC
2	2	0	1		3		BXAC
3	5	0			4		C
4	0	0					XABXAC
5	3	2				2	XAC

В табл. 1 представлен расширенный суффиксный массив с *annotation*-массивом для строки “XABXAC” (для той же строки нами было построено аннотированное суффиксное дерево на рис. 1). Суффиксы строки в последнем столбце приведены для наглядности (в соответствии с определением суффиксного массива они идут в лексикографическом порядке) и в действительности не хранятся в памяти. Три *child*-массива приведены нами для сохранения консистентности с аналогичными иллюстрациями в [1]; в эффективной же программной реализации возможно сжать их до одного массива. Отдельно отметим, что числа в ячейках annotation-массива совпадают с аннотациями узлов из АСД на рис. 1 (два внутренних узла с частотой 2 и корень с частотой 6. Частоты листьев равны 1 и, напомним, в *annotation*-массиве не хранятся).

Приведем предложенный нами в [24] алгоритм построения АСД для коллекции строк, имеющий асимптотически линейную сложность относительно длины строк на входе (где $f(v)$ – частота (аннотация) узла v). Алгоритм использует то свойство АСД, что аннотация любого из внутренних его узла равна сумме аннотаций дочерних узлов. Первым шагом коллекция текстов соответствующим образом подготавливается так, чтобы каждая из строк в ней заканчивалась уникальным символом, а после построения неаннотированного суффиксного дерева аннотации узлов вычисляются простым его обходом (частоты листьев дерева принимаются равными 1, так как они после предобработки строк гарантированно хранят уникальные суффиксы):

Алгоритм **LinearASTConstruction**(C)
Вход. Коллекция строк $C = \{S_1 \dots S_m\}$.
Выход. Обобщенное аннотированное суффиксное дерево для C.

1. Построить $C' = \{S_1\$_1 \dots S_m\$_m\}$, где $\$_i$ - уникальные символы.
2. Построить обобщенное суффиксное дерево T для коллекции C', используя алгоритм с линейной сложностью (например, *алгоритм Укконена*).
3. **for** l **in** $leaves(T)$
4. **do** присвоить $f(l) \leftarrow 1$
5. Выполнить **постфиксный обход в глубину** дерева T; в каждом внутреннем узле v присвоить $f(v) \leftarrow \sum_{u \in T:\, parent(u)=v} f(u)$.

При переходе к от суффиксных деревьев к суффиксным массивам алгоритм построения соответствующих структур данных в целом сохраняет ту же последовательность действий, что и ранее. Строки из входной коллекции вновь снабжаются уникальными завершающими символами, но массивы теперь строятся не для коллекции строк, а для их конкатенации. После построения собственно суффиксного и двух вспомогательных массивов сами аннотации точно так же, как и ранее, вычисляются за один постфиксный обход дерева в глубину, но дерево на этот раз является лишь уже упоминавшимся нами виртуальным *lcp-деревом* (работу с которым несложно реализовать с помощью *lcp-массива* [9]), а обрабатываются только его внутренние узлы (но не листья):

Алгоритм **LinearEASAConstruction**(C)
Вход. Коллекция строк $C = \{S_1 \dots S_m\}$.
Выход. Аннотированный суффиксный массив для C.

1. Построить строку $S = S_1\$_1 + \ \dots \ + S_m\$_m$, где $\$_i$ - уникальные символы, а "+" – оператор конкатенации строк.
2. Построить суффиксный массив A для строки S, используя алгоритм с линейной сложностью (например, *алгоритм Кярккяйнена*), а также два вспомогательных массива: *lcp-массив* и *child-массив*.
3. Выполнить **постфиксный обход в глубину** соответствующего массиву A виртуального *lcp-дерева*; в каждом внутреннем узле, соответствующем *lcp-интервалу* $v = \langle l, i, j \rangle$, где $i < j$, присвоить $annotation[index(v)] \leftarrow (\sum_{u \in A:\, parent(u)=v} annotation[index(u)]) + \#(\langle l, i, j \rangle : i = j)$.

Если длина i-й строки во входной коллекции из m строк равна n_i, то трудоемкость данного алгоритма построения индекса для такой коллекции составляет $\Theta(n_1 + \dots + n_m)$ или же $\boldsymbol{O(mn_{max})}$.

Упомянем важную деталь реализации: при построении как АСД, так и расширенного суффиксного массива для какого-либо входного текста мы рассматриваем этот текст не как одну большую строку, но разбиваем его на последовательность строк из трех слов. Данный подход был предложен в [25] и позволяет добиться большей точности при вычислении оценок степени вхождения в исходный текст ключевых словосочетаний, так как они, как правило, также состоят из двух-трех слов.

При вычислении оценки вхождения в проиндексированную АСД коллекцию текстов некоторого словосочетания (*«наложении»* словосочетания на АСД) используется понятие условной вероятности узла, которая для некоторого узла v принимается равной $\hat{p}(v) = f(v)/f(parent(v))$, где $f(v)$ – частота узла v, $f(parent(v))$ – частота узла-родителя v [24]. Если длина максимального совпадения некоторой строки s с символами в АСД на любом пути от корня до листьев равна k, то вспомогательная оценка $score$ считается как:

$$score(s) = \frac{\sum_{i=1}^{k} \hat{p}(v_i)}{k}$$

Непосредственно сама оценка степени вхождения строки в АСД вычисляется как сумма вспомогательных оценок для ее суффиксов, нормированная по длине строки:

$$SCORE(S) = \frac{\sum_{i=1}^{|S|} score(S[i:])}{|S|}$$

Данная процедура наложения ключевого словосочетания на дерево легко реализуется и в случае с расширенным вспомогательными структурами данных суффиксным массивом: *child-массив* позволяет симулировать не только постфиксный, но префиксный обход дерева [1], который и необходим для реализации данной процедуры.

Мы получили, таким образом, реализацию метода АСД, использующую в качестве базовой структуры данных суффиксный массив с тремя вспомогательными массивами: *lcp-массивом*, *child-массивом* и *annotation-массивом* – и осуществляющей индексацию текста за линейное время. Будем называть данную совокупность структур данных *расширенным аннотированным суффиксным массивом (enhanced annotated suffix array).*

Алгоритм наиболее интересующего нас практического приложения метода АСД – построения графов связей между ключевыми словосочетаниями согласно анализируемому корпусу текстов – не претерпевает изменений по сравнению с оригиналом [25], т.к. использует лишь базовый API структуры данных АСД, который не претерпевает изменений при предлагаемых нами модификациях в его реализации. Первым шагом этого алгоритма для каждого ключевого словосочетания K определяется множество текстов $F(K)$, в которые это словосочетание входит с оценкой выше некоторого порога (в ходе экспериментов с русскоязычными текстами нами использовался порог, равный 0.25). Значимость связи между ключевыми словосочетаниями A и B (точнее говоря, степень, с которой словосочетание A влечет за собой словосочетание B) принимается равной доле множества $F(B)$ в $F(A)$. Собственно граф ключевых словосочетаний является направленными графом, вершины которого – это сами словосочетания, а ребра соединяют те из них, связь между которыми является значимой (в [25], например, предлагается считать значимыми связи начиная с уровня в 60%).

2.2. Учет синонимов в методе АСД

Для повышения точности результатов метода АСД предлагается использовать контекстно-ориентированный алгоритм выделения синонимов, основанный на

моделировании связи между словами (т.е. контекста, в котором они встречаются) через так называемые *тройки грамматических отношений* [11]. Решающим преимуществом такого подхода нам видится тот факт, что синонимы извлекаются на основе исключительно того же самого корпуса текстов, на суффиксное дерево для которого затем будет накладываться набор ключевых словосочетаний. Это позволяет, с одной стороны, рассчитывать на выделение алгоритмом специфичных для предметной области синонимов, которые далеко не всегда возможно извлечь из толковых словарей, а с другой – не извлекает избыточных синонимов, которых вообще нет в анализируемых текстах, не имея таким образом негативного влияния на производительность алгоритма наложения словосочетаний по методу АСД.

В ходе работы алгоритма тройки грамматических отношений выделяются из текста в соответствии с некоторыми *грамматическими шаблонами*, заданными разработчиком заранее. Такими шаблонами могут быть, к примеру, *«прилагательное+существительное»*, *«глагол+наречие»* и другие. Соответствующими тройками отношений могли бы быть, например, $(\text{хорошая}, adj + noun, \text{работа})$, $(\text{сделал}, verb + adv, \text{быстро})$ и т.п. После выделения таких троек для всех возможных пар слов из токенизированного текста считается мера их сходства [11], принимающая значения в $[0; 1]$:

$$sim(w_1, w_2) = \frac{\sum_{(r,w)\in T(w_1)\cap T(w_2)}(I(w_1, r, w) + I(w_2, r, w)}{\sum_{(r,w)\in T(w_1)} I(w_1, r, w) + \sum_{(r,w)\in T(w_2)} I(w_2, r, w)}$$

– где $I(w_1, r, w_2) = log\frac{\|w_1,r,w_2\|\times\|*,r,*\|}{\|w_1,r,*\|\times\|*,r,w_2\|}$, $T(w) = \{(r, w'): I(w, r, w') > 0\}$, а $\|w_1, r, w_2\|$ – частота встречаемости соответствующей тройки во входных текстах. Отношение между словами в тройке, задаваемое этой мерой, является симметричным, но не транзитивным. Отметим, что в нашем случае мы вычисляем оценку схожести только для тех пар, слова в которых имеют длину более *2* символов и встречаются в тексте чаще некоторого порога (экспериментально определенного в нашей работе как $(\frac{\#(\text{тексты в коллекции})}{50})$). Это позволяет значительно оптимизировать время работы алгоритма.

Наконец, синонимами из проанализированных пар слов считаются те, оценка сходства для которых превышает некоторый заранее заданный порог. Ряд проведенных нами экспериментов позволяет предложить нам в качестве такого порога для русскоязычных текстов величину $sim(w_1, w_2)$, равную 0.3.

Сам алгоритм для вычисления оценки степени вхождения словосочетания в тексты предлагается модифицировать следующим образом. Для каждого входного ключевого

словосочетания S сначала необходимо найти множество синонимов (используя некоторый экспериментально определенный порог оценки степени связности слов) $syn(S) = \{w_1, w_2, \dots, w_k\}$. Степень вхождения словосочетания S в текст определяется как $\max_{w \in syn(S) \cup \{S\}} score(w)$, где $score(w)$ – это оценка степени вхождения словосочетания w в коллекцию текстов, вычисляемая как в оригинальном методе АСД [25].

В качестве средства для выделения троек грамматических связей между словами в русскоязычных текстах предлагается использовать ПО *Томита-парсер*, разработанное компанией *Яндекс* [28]. Будучи средством, ориентированным прежде всего на задачу выделения фактов из текстов, Томита-парсер при аккуратной конфигурации своих входных грамматик хорошо справляется также и с задачей выделения подчиненных связей вроде согласования, управления или примыкания. Программа обладает интерфейсом, позволяющим легко провести ее интеграцию с нашей системой в виде съемного компонента.

3. Программная реализация

3.1. Общее описание

Описанные выше алгоритмы, составляющие основу улучшенного метода АСД, были реализованы нами в виде программного продукта *EAST* (аббревиатура расшифровывается как *Enhanced Annotated Suffix Trees*). *EAST* представляет собой Python-пакет [13] с открытым исходным кодом [23], официально зарегистрированный в системе *Python Package Index* [29] и распространяемый под лицензией *MIT* [31]. Он может быть, таким образом, установлен на любой машине, на которой развернут Python, командой:

$ pip install EAST

Альтернативой такому способу установки пакета является загрузка исходного кода из GitHub [23] и непосредственная работа с ним. После установки пакета необходимо также загрузить приложение *Яндекс.Томита-парсер* [28] и поместить исполняемый файл в соответствующую поддиректорию пакета *EAST*. В случае отсутствия этого файла программа будет работать с текстами только в режиме без учета синонимов.

После установки пакета *EAST* он доступен в качестве как python-библиотеки (т.е. может быть включен в код любой другой Python-программы инструкцией *"import east"*), так и консольного приложения (команда *"east"* в консоли) с удобным пользовательским интерфейсом командной строки, позволяющим легко выполнять основные операции анализа текстов по методу АСД и его приложений.

Отдельно отметим, что программное средство позиционируется как ориентированное на работу с русскоязычными текстами (хотя в режиме без учета синонимов оно работает и с любыми другими языками). Входные тексты при этом должны быть представлены в кодировке UTF-8 [33].

Продукт снабжен программной документацией в соответствии с ГОСТ ЕСПД (см. Приложения А-Д).

3.2. Архитектура программного средства

Код пакета *EAST* организован в соответствии с общепринятыми при работе над проектами с открытым исходным кодом практиками. В качестве одного из образцов для него упомянем проект *OpenStack Rally* [30], также разрабатываемый с участием автора. Пакет *EAST* состоит из следующих поддиректорий:

- *analysis* – набор скриптов для проведения экспериментов с доступными реализациями алгоритмов. Эти скрипты были использованы нами при составлении материала главы 4 данной работы;
- *doc* – вспомогательные материалы для конечного пользователя программы:
  - *samples* - набор примеров входных файлов, с которыми может работать программа;
  - *source* – документация на английском языке в формате rst, а также автогенерируемая (на основе комментариев в коде) документация программы.
- *east* – код программы:
  - *asts* – различные реализации алгоритмов построения АСД;
  - *synonyms* – код, связанный с задачей выделения синонимов из текстов.
- *tests* – код юнит-тестов к программе;
- *tools* – использующиеся в программе сторонние программные продукты (в нашем случае – Яндекс.Томита-парсер и его конфигурационные файлы).

Основным принципом при организации модулей программы является четкость в разделении между ними функционала. Например, все специальные классы исключений в *EAST* вынесены в отдельный модуль *exceptions.py*; то же сделано и с использующимися в программе константами (*consts.py*). Различные подпакеты программы содержат в себе модули *utils.py*, всегда содержащие набор вспомогательных методов, использующихся другими модулями в том же пакете.

В целях обеспечения возможности проведения экспериментов по сравнению эффективности новой реализации метода АСД через суффиксные массивы со старыми (линейным и наивным алгоритмами построения аннотированного суффиксного дерева) в *EAST* присутствуют программные реализации всех трех из них. Соответствующая структура классов имеет вид, представленный на рис. 2.

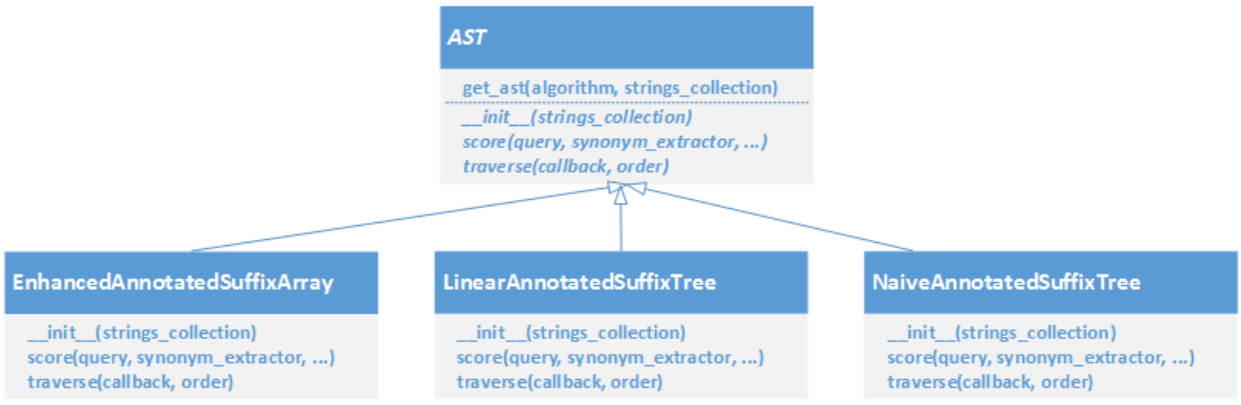

Рисунок 2. Диаграмма классов различных реализаций метода АСД

Все реализации метода АСД унаследованы от одного базового абстрактного класса *AST* и реализуют один и тот же довольно простой интерфейс:

- *__init__(strings_collection)* – инициализация структуры данных коллекцией текстов;
- *score(string)* – вычисление оценки наложения строки на построенную структуру данных;
- *traverse(callback, order)* – обход дерева (или же его симуляция) в заданном порядке.

Сам базовый класс *AST* реализует шаблон проектирования *«абстрактная фабрика классов»*: его метод *get_ast(algorithm, strings_collection)* принимает на вход имя желаемой реализации метода АСД (одно из *“easa”*, *“ast_linear”*, *“ast_naive”*), подгружает соответствующий класс и инициализирует его коллекцией строк. Программист, таким образом, должен инстанцировать классы-наследники *AST* не напрямую, но инструкцией вида *base.AST.get_ast(“easa”, [“String1”, “String2”, ...])*.

Работа с юнит-тестами в *EAST* происходит с использованием встроенного в Python фреймворка *unittest* с расширением *testtools* [32]. Он позволяет организовать код юнит-тестов в классы, наследующиеся от базового класса *testtools.TestCase*, реализует автообнаружение тестов по маске имен тестовых модулей (имеющих в нашем случае имена вида *test_*.py*), а также выводит в командную строку результаты работы тестов в удобном для восприятия формате. Отдельно отметим, что с целью упрощения работы с тестами структура их модулей полностью повторяет структуру соответствующих тестируемых модулей программы. Например, для модуля *east/synonyms/utils.py* соответствующий модуль с юнит-тестами располагается в *tests/synonyms/test_utils.py*. Для запуска юнит-тестов достаточно вызвать специально реализованный нами для этих целей скрипт:

```
$ ./run_tests.sh
```

При вызове данного скрипта автоматически происходит обнаружение тестов, их запуск, вывод информации в консоль и, в случае ошибок, вывод трассировки стека.

Код программного обеспечения снабжен комментариями (называемыми в Python также docstrings [12]), которые унифицированы по своей структуре и могут легко быть использованы для автогенерации документации к коду.

В ходе своей работы программа принимает от пользователя команду на индексацию по методу АСД некоторой коллекции текстов, а также список ключевых словосочетаний, степень присутствия которых в тексте необходимо вычислить. Опционально при этом может осуществляться выделение из текста синонимов и их учет при наложении на АСД

ключевых словосочетаний. UML-диаграмма взаимодействия соответствующих компонент системы представлена на рис. 3. Коллекция текстов, поступающая на вход, сначала передается в объект, отвечающий за выделение синонимов, а затем на ее основе строится АСД. При вычислении оценки вхождения в АСД ключевого словосочетания в метода *score()* также передается объект-выделитель синонимов.

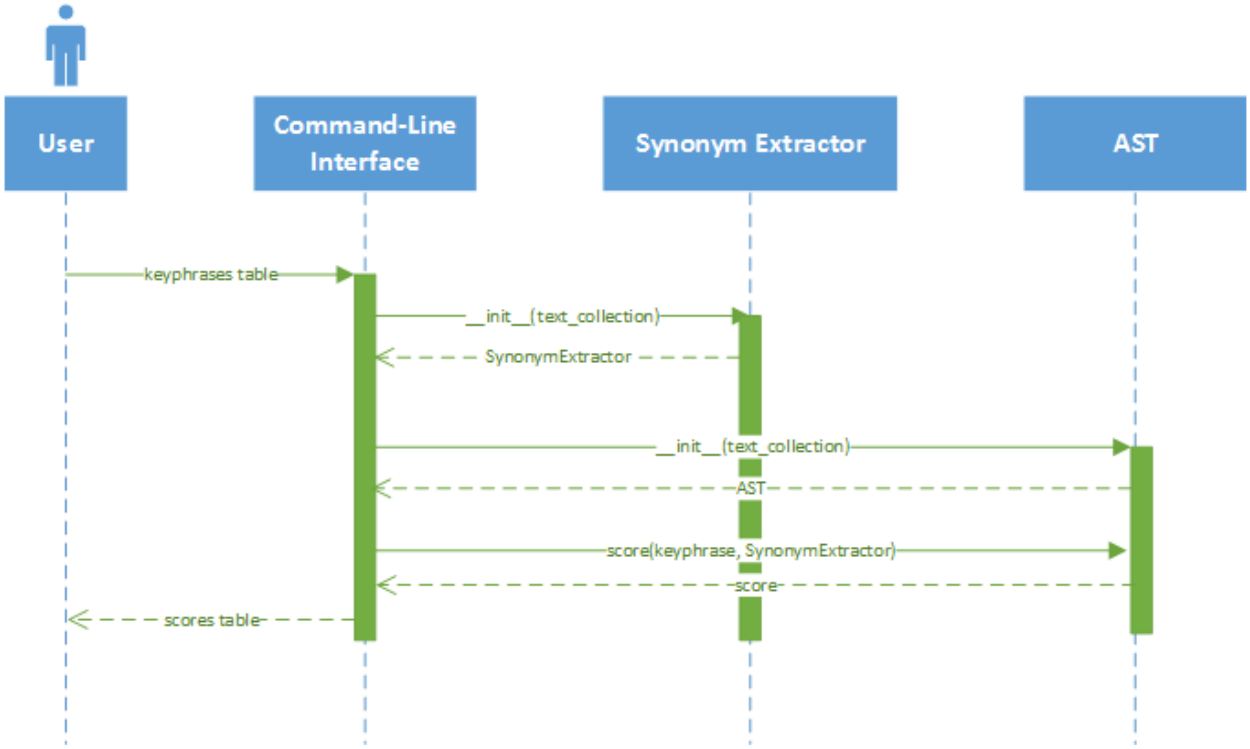

Рисунок 3. UML-диаграмма последовательности для построения таблицы оценок ключевых словосочетаний

3.3. Работа с программным средством

Работа с пакетом *EAST* может осуществляться как в режиме командной строки, так и из кода Python-приложений.

Интерфейс командной строки на данный момент реализует две функции: построение таблицы оценок вхождения ключевых словосочетаний в исходные тексты, а также построение их концептуального графа. Синтаксис команды для выполнение первой из них имеет следующий вид:

$ east [-s] [-d] [-a <ast_algorithm>] keyphrases table <файл_со_словосочетаниями.txt> <директория_с_текстами>

Детали выполнения расчетов регулируются опциями:

- *-s*: если эта опция задана, то оценки считаются с учетом выделенных из текстов синонимов;

- *-d*: если эта опция задана, то оценки считаются в денормализованном виде [25];
- *-a*: задает реализацию метода АСД, которую следует использовать (принимает в качестве аргумента одно из значений *“easa”* – для суффиксных массивов, *“ast_linear”* и *“ast_naive”* – для суффиксных деревьев).

Вывод таблицы программа осуществляет в формате XML (см. приложение Б). Вывод может быть записан вместо консоли в файл стандартными средствами перенаправления вывода.

Аналогичный синтаксис имеет и команда для построения графов:

$ east [-s] [-d] [-a <ast_algorithm>] [-l <significance_level>] [-t <score_threshold>] keyphrases graph <файл_со_словосочетаниями.txt> <директория_с_текстами>

В отличие от предыдущей команды, она имеет две дополнительные опции, *-l* и *-t*, регулирующие уровень значимости связей, для которого строится граф, а также порог оценки, начиная с которого словосочетания начинают считаться входящими в текст. Вывод графа осуществляется в одном из наиболее распространенных форматов – GML [6], что позволяет легко затем использовать сторонние средства для визуализации и анализа получившихся графов.

При использовании в качестве Python-библиотеки программа также предоставляет весь перечисленный выше функционал. Подробно интерфейсы этой библиотеки описаны в приложении В.

4. Анализ полученных результатов

4.1. Ресурсная эффективность

С помощью разработанного программного средства был проведен ряд экспериментов с целью практического подтверждения повышенной ресурсной эффективности алгоритмов, использованных в новой реализации метода АСД. Содержанием экспериментов являлось построение аннотированных суффиксных деревьев для ряда случайно сгенерированных коллекций из 100 строк (моделирующих коллекции текстов), в каждой из которых строки имели фиксированную длину. Для каждой из таких тестовых строковых коллекций АСД строилось одним из трех алгоритмов:

1. «Наивный» алгоритм построения АСД, описанный в [25] и имеющий временную сложность $\Theta(n_1^2 + n_2^2 + \cdots + n_m^2)$ или $O(mn_{max}^2)$ в худшем случае (n_i – длина i-й строки в коллекции текстов, m – общее количество текстов);
2. Линейный алгоритм построения АСД, предложенный в [24] и имеющий временную сложность $\Theta(n_1 + n_2 + \cdots + n_m)$ или, если использовать несколько более грубую оценку, $O(mn_{max})$ в худшем случае;
3. Алгоритм построения АСД, основанный на использовании суффиксных массивов и описанный в данной работе. Временная сложность этого алгоритма также составляет $O(mn_{max})$ в худшем случае.

Для всех трех алгоритмов в ходе построения АСД осуществлялся замер объема используемой программой памяти, а также времени, которое потребовалось программе для построения АСД. Это позволило проанализировать зависимость количества потребляемых программой ресурсов от длины строк в коллекции. Отдельно отметим, что коллекции текстов генерировались таким образом, что строки в них отличались друг от друга только двумя последними символами. Эти коллекции, таким образом, обеспечивали максимизацию количества проходов по строившимся для них АСД в случае использования наивного алгоритма и представляли собой входные данные, близкие для него к «худшему случаю».

На рис. 4 представлены результаты замеров объемов используемой памяти. Результаты эксперимента наглядно демонстрируют, насколько значительным является выигрыш в объеме используемой памяти при построении АСД с использованием суффиксных массивов. Так, при построении АСД для 100 строк длиной в 500 символов каждая обычным алгоритмом требуется около 250 МБ памяти, в то время как при использовании суффиксных массивов (с рядом вспомогательных массивов, как описано в

данной работе) требуется всего порядка 20-25 МБ памяти, чем достигается ее десятикратная экономия.

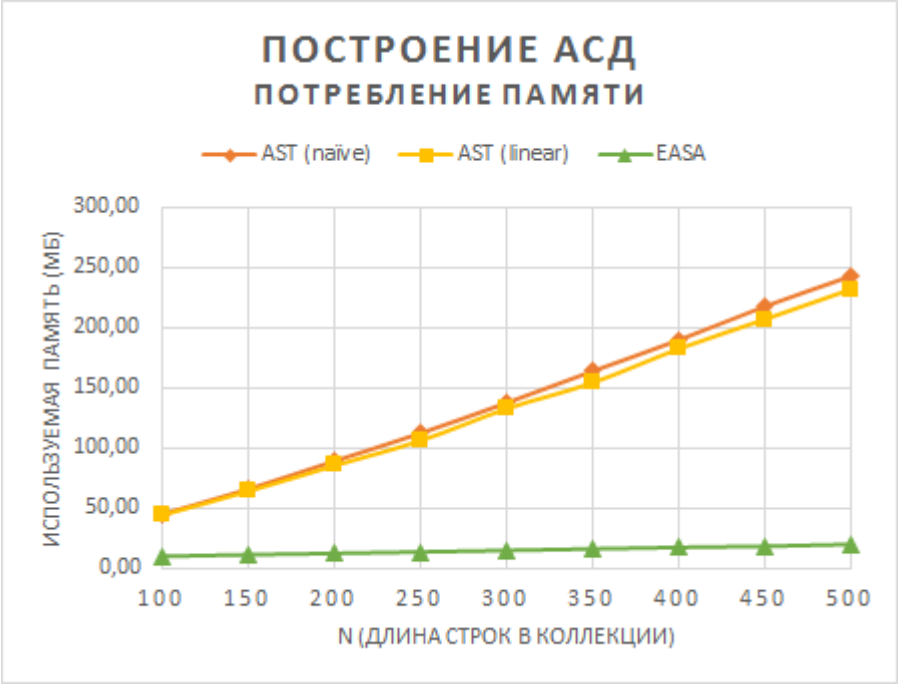

Рисунок 4. Эксперимент: объем используемой памяти

Отдельно отметим, что как линейный, так и наивный алгоритм построения АСД строят идентичную структуру данных (суффиксное дерево, представленное в виде узлов и ребер между ними), однако линейный алгоритм потребляет немного меньше памяти из-за меньшего числа выполняемых им операций.

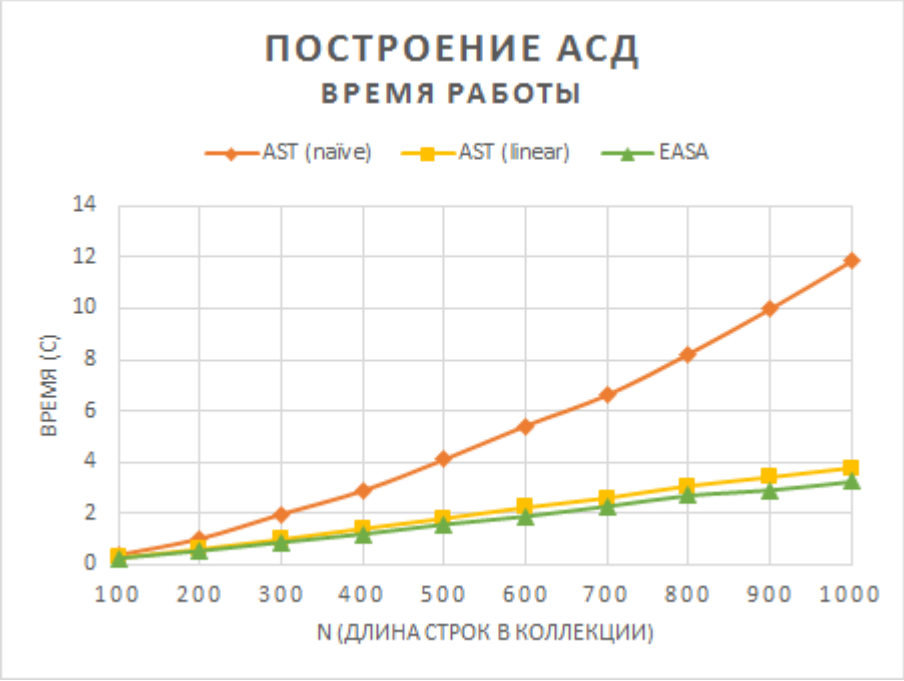

Рисунок 5. Эксперимент: время построения АСД

Замеры времени работы трех анализируемых алгоритмов представлены на рис. 5. В то время как наивный алгоритм ожидаемо имеет время работы, квадратично зависящее от длины строк в коллекции, линейный алгоритм построения АСД и алгоритм на суффиксных массивов демонстрируют асимптотически линейное (и при этом практически сопоставимое) время работы. Алгоритм на суффиксных массивах, однако, имеет несколько лучшее (в среднем на 15%) время работы благодаря меньшему числу выполняемых в нем операций с памятью.

4.2. Работа с синонимами

Для анализа качества работы алгоритма выделения синонимов был использован набор из 100 статей по теме экономика, выгруженных с интернет-сайта газеты «Известия» (опубликованы в марте 2014 г. [34]). После фильтрации слишком коротких (короче 3-х символов) и редко встречающихся слов (для 100 текстов редкими словами считались те, что встречались не более двух раз), оставшиеся слова были проанализированы программной реализацией алгоритма, описанного в [11], на предмет сходства контекстов, в которых они встречаются. Результатом работы алгоритма являлись оценки степени синонимичности (от 0 до 1) для каждой из пар слов. Синонимами считались пары слов, степень синонимичности которых составляла не менее 0.3 (значение определено экспериментально). В результате работы алгоритма было выделено один или несколько синонимов для *581* слова. В табл. 2 представлен небольшой отрывок из списка выделенных программой синонимов.

Таблица 2

Выделенные программой синонимы для некоторых слов из новостных публикаций

Слово	**Синонимы (с оценкой схожести)**
ГЛАВА	ГЕНДИРЕКТОР (0.35), ПРЕДСЕДАТЕЛЬ (0.35), НАЧАЛЬНИК (0.44), РУКОВОДИТЕЛЬ (0.34)
НАЧАЛСЯ	ИДЁТ (0.40)
РЕНТАБЕЛЬНОСТЬ	ПРИБЫЛЬ (0.35), СЕБЕСТОИМОСТЬ (0.53), КОНКУРЕНЦИЯ (0.37)
ПИЩЕВОЙ	МЯСНОЙ (0.55)
ДЕСЯТИЛЕТНИЙ	ОПТИМИСТИЧНЫЙ (0.45), ДОЛГИЙ (0.45), ПЯТИЛЕТНИЙ (0.47), ЛЕТНИЙ (0.71), КОРОТКИЙ (0.62), ДВУХЧАСОВОЙ (0.33)
ВЫДЕЛЯТЬ	НАЗВАТЬ (1.00)
ДОЛГ	ЗАДОЛЖЕННОСТЬ (0.44), МЯСО (0.35)
НИЗКИЙ	ВЫСОКИЙ (0.32)
НАЗВАЛ	СООБЩИЛ (0.31), ПОДЧЕРКНУЛ (1.00), СЧИТАЕТ (1.00), НАПИСАЛ (1.00), ОТМЕЧАЕТ (1.00), ПОЯСНИЛ (1.00)

Данные результаты, на наш взгляд, достаточно полно отражают как достоинства, так и недостатки реализованного нами контекстно-ориентированного подхода к выделению синонимов из текстов. С одной стороны, поиск синонимов на основе анализа ограниченной (прежде всего тематически) совокупности текстов позволяет выделять ряд специфических для той или иной области пар синонимов (в нашем примере это *«глава»* и *«гендиректор»* – явные синонимы в контексте обсуждения деятельности компаний, но не всегда в других случаях). С другой стороны, анализ только лишь контекстов встречаемости отдельных слов время от времени влечет за собой выделение вместо синонимов гипонимов/гиперонимов (*«пищевой»* и *«мясной»*) или даже антонимов (*«низкий»* и *«высокий»*).

Очевидно, что значительная часть выделенных пар синонимов в данном примере оказалась нерелевантной; это, однако, не критично с точки зрения конечного результата работы метода АСД. Напомним, что при наложении ключевых словосочетаний на АСД целью является получить оценку, учитывающую возможное присутствие в тексте синонимов. Для достижения этой цели на основе списка синонимов строятся синонимичные ключевые словосочетания – например, в данном примере для словосочетания *«Долг по кредиту»* это будут *«Задолженность по кредиту»* и *«Мясо по кредиту»* – а затем каждое из них накладывается на АСД. В качестве результата принимается максимальная из полученных оценок. В нашем примере ключевое словосочетание *«Мясо по кредиту»* представляет собой один из неудачных примеров выделения синонимов, но оно с малой долей вероятности повлияет на итоговый результат, т.к. оценка вхождения в АСД для него будет ниже, чем у других синонимичных словосочетаний. При построении же графов, напомним, ребра образуются между узлами, отвечающими за словосочетания с высокой совместной встречаемостью в анализируемых текстах. Повышение значений оценок вхождения в тексты словосочетаний за счет подстановки вместо них синонимичных словосочетаний, таким образом, потенциально повышает количество ребер в концептуальном графе и позволяет более полно учитывать в нем связи между словосочетаниями.

4.3. Применение ПО к анализу реальных текстов

С целью интеграционного тестирования всех компонентов разработанного нами программного средства оно было опробовано нами на сравнительно небольшой, но вызывающей интерес коллекции текстов – положениях *«Правил внутреннего распорядка НИУ ВШЭ»* (статьи главы 3 – *«Правила учебного распорядка»* [35]). Каждая из статей данного документа была включена нами во входную коллекцию как отдельный текст

(данная коллекция текстов также включена в комплект поставки программы в директории *doc/samples/texts/HSE Rules*). Набор ключевых слов был составлен нами вручную: он содержит такие базовые для студенческой жизни понятия как *«стипендия»*, *«рейтинг»* или *«отчисление из университета»* (данный набор доступен в файле *doc/samples/keyphrases/HSE.txt*).

Программное средство *EAST* было запущено нами со следующими параметрами:

$ east keyphrases graph "doc/samples/keyphrases/HSE.txt" "doc/samples/texts/HSE Rules" > hse_graph.gml

Напомним последовательность действий, выполняемых программой при подобном ее вызове через интерфейс командной строки:

1. Входная коллекция текстов индексируется путем построения для каждого из ее текстов расширенного аннотированного суффиксного массива;
2. Для каждого из входных ключевых словосочетаний считается оценка степени его встречаемости в каждом из текстов коллекции;
3. Полученные оценки анализируются согласно алгоритму, описанному в главе 2, с целью построения графа связей между ключевыми словосочетаниями;
4. Полученный граф печатается программой в стандартный поток вывода в формате GML [6].

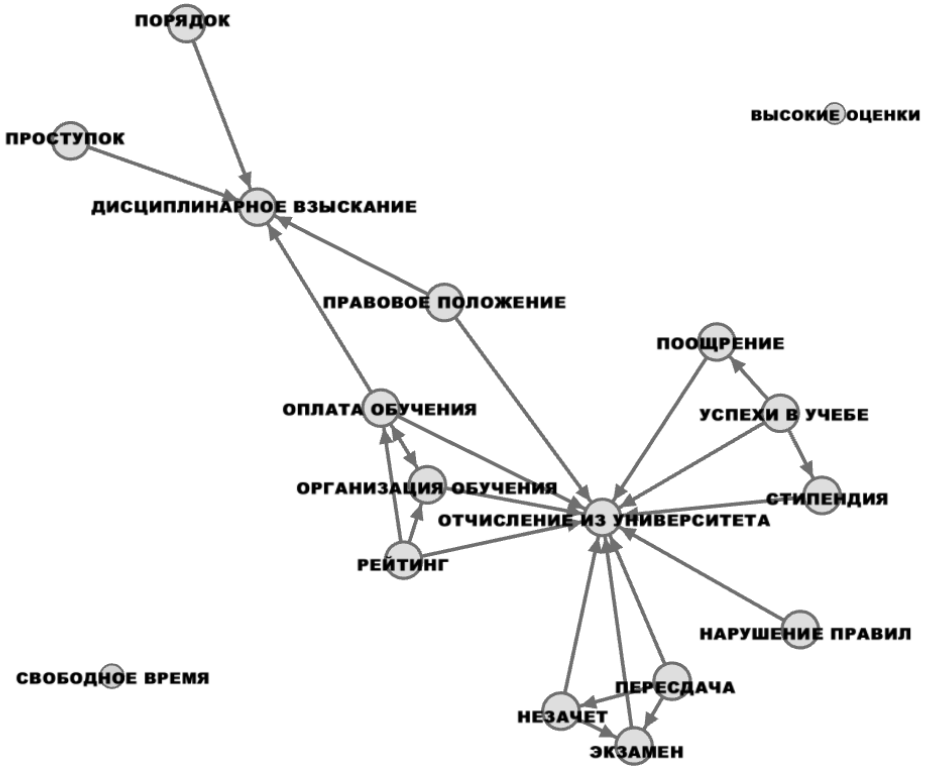

Рисунок 6. Концептуальный граф словосочетаний на основе анализа "Правил внутреннего распорядка НИУ ВШЭ"

Визуализация полученного нами графа связей между ключевыми словосочетаниями представлена на рис. 6. Многие автоматически выведенные на основе анализа текстов входного документа связи вполне объективно отражают окружающую реальность: например, из *«проступка»* на графе следует *«дисциплинарное взыскание»*, а из *«успехов в учебе»* – *«поощрение»* и *«стипендия»*. Присутствует, однако, и небольшое количество кажущихся неправдоподобными связей, таких как связь между *«поощрением»* и *«отчислением из университета»*. Их возникновение можно объяснить достаточно малым объемом анализируемых текстов (некоторые разделы документа состоят из одного-двух предложений) и, как следствие, некоторой их зашумленностью. Отметим также отсутствие каких-либо ребер, входящих в узлы с метками *«свободное время»* и *«высокие оценки»* или выходящих из них. Это является верным признаком того, что данные понятия не упоминаются во входных текстах, что подтверждается ручной их проверкой.

Следующим шагом эксперимента нами было проведено повторное построение графа, но на этот раз уже с учетом встречающихся в текстах синонимов (для этого в интерфейсе командной строки программа вызывалась с опцией *'-s'*):

$ east ***-s*** *keyphrases graph "doc/samples/keyphrases/HSE.txt" "doc/samples/texts/HSE Rules" > hse_graph_2.gml*

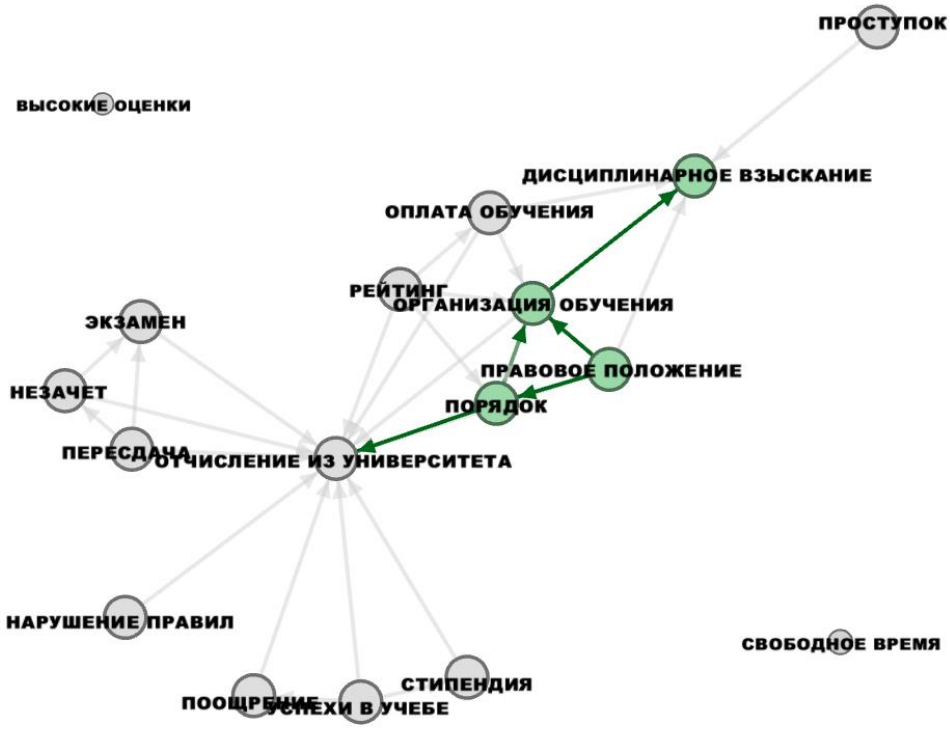

Рисунок 7. Концептуальный граф словосочетаний на основе анализа "Правил внутреннего распорядка НИУ ВШЭ" (с учетом синонимов)

Ручная проверка синонимов, выделенных программой при анализе входных текстов, обнаруживает несколько удачно распознанных пар схожих по значению слов, среди них: *«положение»* и *«акт»*, *«организация»* и *«порядок»*, *«группа»* и *«курс»*. Выделение в качестве синонимов понятий *«организации»* и *«порядка»* является причиной тех изменений, которые произошли в концептуальном графе (см. рис. 7) по сравнению с его предыдущей версией (новые узлы и ребра выделены зеленым). Появившиеся в графе связи – например, между *«правовым положением»* и *«порядком»*, а также между *«организацией обучения»* и *«дисциплинарным взысканием»* – следует признать вполне удачными. Этот пример наглядно демонстрирует, что даже при анализе все еще относительно скромной по объему коллекции текстов (правил внутреннего распорядка НИУ ВШЭ) учет синонимов уже может принести ощутимое улучшение в результатах работы метода АСД. Ожидаемые улучшения при использовании модифицированного метода аннотированных суффиксных деревьев относительно оригинального при анализе больших текстовых корпусов, соответственно, еще выше.

Эти ожидания в скором времени планируется подтвердить или опровергнуть путем анализа значительно более объемного русскоязычного текстового корпуса, который будет состоять из собираемых нами в настоящий момент в рамках деятельности Научно-учебной группы НИУ ВШЭ «Методы анализа и визуализации текстов» [27] новостных публикаций ведущих российских изданий, таких как уже упомянутые выше «Известия» [34]. К настоящему моменту созданный нами в рамках данной работы программный продукт *EAST* внедрен в ПО для анализа текстов, разрабатываемое членами НУГ. В ближайшем времени оно будет применено на практике для построения концептуальных графов на основе обработки собранных нами статей.

Заключение

Нами разработано и опубликовано программное средство *EAST* для анализа текстов методом аннотированных суффиксных деревьев, являющееся на сегодняшний день наиболее эффективным среди аналогов по объему потребляемой памяти и времени работы основных алгоритмов. Программное средство поддерживает функции построения аннотированных суффиксных деревьев для входной коллекции текстов и ответа на запросы об оценке степени вхождения в построенное АСД списка ключевых словосочетаний. Кроме того, программное средство опционально осуществляет выделение синонимов в анализируемой коллекции текстов и корректировку оценки присутствия в текстах ключевых словосочетаний с учетом этих синонимов. Тем самым достигается улучшение семантической выразительности результатов работы программы.

Тот факт, что разработанное программное средство продемонстрировало десятикратный выигрыш в объеме используемой памяти в сравнении с предшествующими реализациями метода АСД, открывает широкие перспективы для его использования исследователями в своей работе: нетребовательность к ресурсам означает возможность анализа текстовых корпусов больших размеров на доступных большинству исследователей вычислительных мощностях.

Публикация же программного средства и организация его кода в соответствии с лучшими практиками разработки открытого программного обеспечения обеспечивает низкий порог вхождения для людей, готовых внести свой вклад в его разработку или осуществить его внедрение в другой программный продукт.

К настоящему моменту с участием автора в рамках Научно-учебной группы НИУ ВШЭ «Методы анализа и визуализации текстов» [27] была проведена работа по внедрению разработанного ПО в проект по созданию системы для сбора и анализа текстов новостных публикаций из Web-ресурсов. В данном проекте метод АСД используется для вычисления оценок вхождения заданных ключевых словосочетаний в собранные из Web тексты и построения на основе анализа этих оценок графов связей между ключевыми словосочетаниями.

Дальнейшие направления работы над описанным нами программным средством включают оптимизацию времени работы используемого в нем алгоритма построения АСД с использованием суффиксных массивов, уточнение множества используемых для выделения синонимов грамматических отношений (с привлечением к работе экспертов-

лингвистов), а также расширение круга практических приложений метода АСД, поддерживаемых нашим ПО.

Список использованных источников

1. Abouelhoda, M.I. Replacing Suffix Trees with Enhanced Suffix Arrays / M.I. Abouelhoda, S. Kurtz, E. Ohlebusch // *Journal of Discrete Algorithms*, Amsterdam: Elsevier. – 2004. – №2. – С. 53-86.
2. Barsky, M. A Survey of Practical Algorithms for Suffix Tree Construction in External Memory / M. Barsky, U. Stege, A. Thomo // *Software – Practice & Experience*, New York: John Wiley & Sons. – 2010. – №40-11 – С. 965-988.
3. Blondel, V.D. Automatic Extraction of Synonyms in a Dictionary. / V.D. Blondel, P.P. Senellart // *Proceedings of the SIAM Workshop on Text Mining*. – 2002.
4. Gusfield, D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology / D. Gusfield. – Cambridge University Press, 1997.
5. Haralambous, Y. A Semantic Relatedness Measure Based on Combined Encyclopedic, Ontological and Collocational Knowledge. / Y. Haralambous, V. Klyuev, V. // *Proceedings of the 5th International Joint Conference on Natural Language Processing*, Asian Federation of Natural Language Processing. – 2011. – С. 1397-1402.
6. Himsolt, M. GML: A Portable Graph File Format. Technical Report. / M. Himsolt // Universität Passau Projects.
7. Jones, N.C. An Introduction to Bioinformatics Algorithms. / N.C. Jones, & P.A. Pevzner. – Massachusetts Institute of Technology Press, 2004.
8. Kärkkäinen, J. Simple Linear Work Suffix Array Construction. / J. Kärkkäinen, P. Sanders. // *Automata, Languages and Programming. Lecture Notes in Computer Science*, 2719 – С. 943.
9. Kasai, T. Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and its Applications. / T. Kasai, G. Lee, H. Arimura, S. Arikawa, K. Park. // *Proceedings of the Annual Symposium on Combinatorial Pattern Matching, in: Lecture Notes in Computer Science, 2089*, Berlin: Springer-Verlag. – С. 181-192.
10. Ko, P. Space Efficient Linear Time Construction of Suffix Arrays. / P. Ko, S. Aluru // *Proceedings of the Annual Symposium on Combinatorial Pattern Matching, in: Lecture Notes in Computer Science, 2676*, Berlin: Springer-Verlag. – 2003. – С. 200-210.
11. Lin, D. Automatic Retrieval and Clustering of Similar Words. / D. Lin // *Proceedings of the 17th International Conference on Computational Linguistics*. – 1998. – С. 768-774.
12. Lutz, M. Programming Python. / M. Lutz – O'Reilly Media, 2010.
13. Manber, U. Suffix Arrays: a New Method for On-Line String Searches. / U. Manber, E.W. Myers // *SIAM J. Comput.* – 1993. - №22(5). – С. 935-948.

14. Mansour, E. ERA: Efficient Serial and Parallel Suffix Tree Construction for Very Long Strings / E. Mansour, A. Allam, S. Skiadopoulos, P. Kalnis // *Proceedings of the VLDB Endowment*. – 2011. – №5-1. – С. 49-60.
15. Pampapathi, R. A Suffix Tree Approach to Anti-Spam Email Filtering / R. Pampapathi, B. Mirkin, M. Levene // Machine Learning – 2006. - v.65 №1 – С. 309-338.
16. Pampapathi, R.M. *Annotated Suffix Trees for Text Modelling and Classification* / R.M. Pampapathi // Doctoral dissertation, Birkbeck College, University of London. – Retrieved from CiteSeerX. – 2008.
17. Perkins, J. Python Text Processing with NLTK 2.0 Cookbook. / J. Perkins. – Birmingham, UK: Packt Publishing, 2010.
18. Sadakane K. Compressed Suffix Trees with Full Functionality / K. Sadakane // *Theory of Computing Systems*, New-York: Springer-Verlag. – 2007. – №41-4. – С. 589-607.
19. Sadakane K. Succinct Representations of Lcp Information and Improvements in the Compressed Suffix Arrays. / K. Sadakane // *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. – 2002. – С. 225–232.
20. Ukkonen, E. On-Line Construction of Suffix Trees. / E. Ukkonen // *Algorithmica*. – 1995. – №14(3). – С. 249-260.
21. Wang, T. *Extracting Synonyms from Dictionary Definitions* / T. Wang // Master dissertation, University of Toronto. – Retrieved from Focus on Research. – 2009.
22. Weiner P. Linear Pattern Matching Algorithms. / P. Weiner / Proceedings of the 14th IEEE Symposium on Switching and Automata Theory. – 1973. – С. 1-11.
23. Дубов, М.С. Text Analysis via Annotated Suffix Trees (git-репозиторий) [Электронный ресурс] / М.С. Дубов. – Режим доступа: https://github.com/msdubov/AST-text-analysis (дата обращения: 13.12.2013).
24. Дубов, М.С. Аннотированные суффиксные деревья: особенности реализации / М.С. Дубов, Е.Л. Черняк // *Доклады по компьютерным наукам и информационным технологиям* – 2013. – №2 – Доклады всероссийской научной конференции «Анализ изображений, сетей и текстов» (АИСТ 2013). М: Национальный Открытый Университет ИНТУИТ, апрель 2013 – С. 49-57.
25. Миркин, Б.Г. Метод аннотированного суффиксного дерева для оценки степени вхождения строк в текстовые документы / Б.Г. Миркин, Е.Л. Черняк, О.Н. Чугунова // Бизнес-информатика – 2012 г. - №3 (21) – стр. 31-41.
26. Миркин, Б.Г. Построение графов связи между ключевыми словосочетаниями по методу АСД / Б.Г. Миркин, Е.Н. Моренко. – 2013.

27. Научно-учебная группа «Методы анализа и визуализации текстов» [Электронный ресурс] / НИУ ВШЭ. Режим доступа: http://ami.hse.ru/vitext (дата обращения: 27.04.2014).
28. Яндекс.Томита-парсер [Электронный ресурс] / Яндекс. Режим доступа: http://api.yandex.ru/tomita/ (дата обращения: 20.04.2014).
29. EAST [Электронный ресурс] / Python Package Index. Режим доступа: https://pypi.python.org/pypi/EAST (дата обращения: 25.05.2014).
30. OpenStack Rally (git-репозиторий) [Электронный ресурс] / Режим доступа: https://github.com/stackforge/rally (дата обращения: 15.05.2014).
31. The MIT License [Электронный ресурс] / Open Source Initiative. Режим доступа: http://opensource.org/licenses/mit-license.php (дата обращения: 15.05.2014).
32. Unittest – Unit Testing Framework [Электронный ресурс] / Python Documentation. Режим доступа: https://docs.python.org/2/library/unittest.html (дата обращения: 17.05.2014).
33. UTF-8 and Unicode Standards [Электронный ресурс] / Режим доступа: http://www.utf-8/com (дата обращения: 02.04.2014).
34. Новости Экономики – Известия [Электронный ресурс] / Режим доступа: http://izvestia.ru/rubric/16 (дата обращения - 01.04.2014).
35. Правила Внутреннего Распорядка НИУ ВШЭ [Электронный ресурс] / Режим доступа: http://www.hse.ru/docs/48094015.html (дата обращения: 25.05.2014).