УДК 004.912 **DOI** 10.15622/ia.23.2.6

Д.Ю. Кравченко, Ю.А. Кравченко, А. Мансур, Ж. Мохаммад, Н.С. Павлов

АЛГОРИТМ ОПТИМИЗАЦИИ ИЗВЛЕЧЕНИЯ КЛЮЧЕВЫХ СЛОВ НА ОСНОВЕ ПРИМЕНЕНИЯ ЛИНГВИСТИЧЕСКОГО ПАРСЕРА

Кравченко Д.Ю., Кравченко Ю.А., Мансур А., Мохаммад Ж., Павлов Н.С. **Алгоритм оптимизации извлечения ключевых слов на основе применения лингвистического парсера.**

Аннотация. В данной статье представлено аналитическое исследование особенностей двух типов парсинга, а именно синтаксический анализ составляющих (constituency parsing) и синтаксический анализ зависимостей (dependency parsing). Также в рамках проведенного исследования разработан алгоритм оптимизации извлечения ключевых слов, отличающийся применением функции извлечения именных фраз, предоставляемой парсером, для фильтрации неподходящих фраз. Алгоритм реализован с помощью трех разных парсеров: SpaCy, AllenNLP и Stazna. Эффективность предложенного алгоритма сравнивалась с двумя популярными методами (Yake, Rake) на наборе данных с английскими текстами. Результаты экспериментов показали, что предложенный алгоритм с парсером SpaCy превосходит другие алгоритмы извлечения ключевых слов с точки зрения точности и скорости. Для парсера AllenNLP и Stanza алгоритм так же отличается точностью, но требует гораздо большего времени выполнения. Полученные результаты позволяют более детально оценить преимущества и недостатки изучаемых в работе парсеров, а также определить направления дальнейших исследований. Время работы парсера SpaCy значительно меньше, чем у двух других парсеров, потому что парсеры, которые используют переходы, применяют детерминированный или машинно-обучаемый набор действий для пошагового построения дерева зависимостей. Они обычно работают быстрее и требуют меньше памяти по сравнению с парсерами, основанными на графах, что делает их более эффективными для анализа больших объемов текста. С другой стороны, AllenNLP и Stanza используют модели парсинга на основе графов, которые опираются на миллионы признаков, что ограничивает их способность к обобщению и замедляет скорость анализа по сравнению с парсерами на основе переходов. Задача достижения баланса между точностью и скоростью лингвистического парсера является открытой темой, требующей дальнейших исследований в связи с важностью данной проблемы для повышения эффективности текстового анализа, особенно в приложениях, требующих точности при работе в реальном масштабе времени. С этой целью авторы планируют проведение дальнейших исследований возможных решений для достижения такого баланса.

Ключевые слова: синтаксический анализ составляющих, синтаксический анализ зависимостей, извлечение ключевых слов, обработка естественного языка, NLP, SpaCy, Stanza, AllenNLP.

1. Введение. В динамичном мире обработки естественного языка (Natural Language Processing, NLP) синтаксический анализ (парсинг) играет ключевую роль в раскрытии сложностей естественного языка. Как основа к пониманию структуры и смысла предложений, парсеры служат незаменимыми инструментами

в различных задачах NLP, позволяя машинам воспринимать и обрабатывать естественный язык с более высокой точностью и эффективностью. От анализа настроений до машинного перевода, а также для систем вопросов и ответов, парсеры играют ключевую роль в преобразовании предложений в синтаксические структуры, что в свою очередь облегчает более точную и контекстно значимую обработку языка. Разбивая предложения на понятные единицы, парсеры создают фундамент для машинного понимания семантики и взаимосвязей между словами, делая возможным достижение более сложных и тонких результатов в различных приложениях [1 – 3], поэтому *создание эффективных текстовых парсеров является весьма актуальной научной проблемой в настоящее время*.

Основные подходы к синтаксическому парсингу включают синтаксический анализ составляющих (constituency parsing) и синтаксический анализ зависимостей (dependency parsing).

Анализ составляющих и зависимостей – это взаимодополняющие подходы, которые направлены на анализ синтаксической структуры предложений. Эти методы анализа предоставляют ценные сведения о грамматической структуре и семантических отношениях в предложении [3].

Анализ составляющих сосредоточен на определении конституентов, которые являются группами слов, выполняющими единую функцию в предложении. Эти конституенты могут быть фразами, такими как именные фразы (NP) или глагольные фразы (VP), или даже более крупными единицами, такими как предложения. Анализ составляющих представляет иерархическую схему предложения с использованием древовидной структуры, называемой деревом разбора или синтаксическим деревом. С другой стороны, анализ зависимостей сосредоточен на отношениях между отдельными словами в предложении. Он представляет эти отношения в виде направленных связей или зависимостей, где каждое слово связано со своим синтаксическим корневым или управляющим словом. Анализ зависимостей обеспечивает более линейное представление структуры предложения, акцентируя внимание на зависимостях между словами, а не на иерархической организации конституентов [4].

Одним из известных алгоритмов в конституентном анализе являются алгоритм CYK (Cocke-Younger-Kasami). Этот классический алгоритм разбора, основанный на динамическом программировании, эффективно строит дерево разбора, разбивая предложения на более мелкие конституенты с использованием контекстно-свободной грамматики. Так же известен алгоритм Эрли, который способен

обрабатывать неоднозначные грамматики и разбирать предложения с использованием предсказывающего сверху-вниз и снизу-вверх подхода, что приводит к более надежному процессу разбора [3].

С другой стороны, популярным алгоритмом анализа зависимостей является алгоритм Arc-Eager. Это алгоритм разбора на основе переходов (Transition-based), который предсказывает последовательность действий для построения дерева зависимостей, эффективно отображая отношения между корневыми и зависимыми словами. Другим подходом на основе переходов является алгоритм Arc-Standard, который строит деревья зависимостей, сводя предложение к однокоренному дереву с помощью серии действий [5].

Последние исследования в области обработки естественного языка открыли потенциал глубокого обучения для повышения эффективности синтаксического анализа зависимостей [5, 6]. Используя архитектуры нейронных сетей и обширные объемы размеченных данных, парсеры на основе глубокого обучения достигли значительного улучшения точности и эффективности. Существуют техники, включая парсинг на основе переходов с использованием нейронных сетей [5], которые улучшают традиционные парсеры путем интеграции нейронных сетей для более точного улавливания контекстуальных особенностей и зависимостей.

В [6] используют графовые нейронные сети (graph-based) для выполнения синтаксического анализа зависимостей, что позволяет более эффективно обрабатывать неявные зависимости и синтаксические структуры.

Было установлено, что глубоко контекстуализированные представления слов оказывают еще больший положительный эффект на transition-based парсинг, чем graph-based парсинг [7]. Информация о синтаксической структуре, содержащаяся в глубоко контекстуализированных представлениях слов, помогает смягчить главный недостаток transition-based алгоритмов в виде ошибок при обработке длинных предложений. Модели глубоко контекстуализированных представлений слов ELMo [2] и BERT [1] позволили значительно улучшить результаты для обоих алгоритмов парсинга, причем для transition-based парсинга улучшение оказалось более значимым.

Transition-based и graph-based подходы обладают взаимодополняющими преимуществами и недостатками. Несмотря на то, что transition-based и graph-based парсеры показывают примерно равную точность, они совершают ошибки разного рода. Transition-based парсеры чаще ошибаются в длинных предложениях,

зависимостях около корня дерева, зависимостях с глаголом и союзами, а также в определении корневого слова. Это связано с жадным алгоритмом, где ошибка в одной зависимости может привести к каскадным ошибкам в других зависимостях. С другой стороны, graph-based парсеры чаще допускают ошибки в коротких предложениях, зависимостях с существительными и местоимениями, а также в зависимостях вблизи листьев дерева. Это связано с ограниченным набором признаков. Таким образом, оба подхода имеют свои преимущества и недостатки, которые дополняют друг друга.

Чтобы подтвердить наблюдения, упомянутые выше, и с целью выяснения различий между этими двумя типами лингвистических анализаторов и изучения их влияния на производительность и скорость задач анализа текста, в данном исследовании представлено аналитическое сравнение нескольких известных лингвистических анализаторов, каждый из которых принадлежит к различным подходам: SpaCy [8], основанный на переходах (анализ зависимостей), и Stanza [9] (анализ зависимостей) и AllenNLP [10] (анализ составляющих), основанные на графах. Оценена скорость каждого парсера при анализе длинных текстов и коротких предложений. Также изучается влияние использования каждого из этих парсеров на задачу извлечения ключевых фраз, где разрабатывается алгоритм для извлечения ключевых фраз, использующий возможности этих технологий для определения именных фраз в тексте.

2. Синтаксический парсинг составляющих. Синтаксический анализ составляющих заключается в разбиении предложения на составные части (отдельные слова). Наиболее распространенной моделью, описывающей составную структуру предложения, является контекстно-свободная грамматика (context-free grammar).

Контекстно-свободная грамматика представляет собой набор правил, который определяет способы группировки и упорядочивания слов. Она получила свое название по причине того, что все порождающие правила в грамматике могут применяться независимо от контекста – они не зависят от каких-либо других языковых единиц, которые могут или не могут быть вокруг данной языковой единицы, к которой применяется правило.

Например, правила контекстно-свободной грамматики могут быть представлены следующим образом:

$$\text{Nominal} \rightarrow \text{Noun} \mid \text{Nominal Noun}, \quad (1)$$

$$\text{NP} \rightarrow Det\ \text{Nominal}, \quad (2)$$

$$\text{NP} \rightarrow \text{Proper Noun}. \quad (3)$$

Именная группа (Noun Phrase или NP) может быть составлена либо из определяющего слова (determiner или det) и следующего за ним одного или нескольких существительных (Nominal), что отражено в выражении 2, либо из имени собственного (Proper Noun) – в выражении 3.

Контекстно-свободная грамматика обладает иерархической структурой, то есть правила могут быть вложены друг в друга. Например, в правило 2 может быть вложено следующее правило:

$$Det \rightarrow a, \quad (4)$$

$$Det \rightarrow an, \quad (5)$$

$$Det \rightarrow the, \quad (6)$$

$$Noun \rightarrow cat. \quad (7)$$

В контекстно-свободной грамматике используются символы двух классов. Терминальные символы (terminal symbols) соответствуют словам (“a”, “good”, “dog” и т.д.) и не могут быть разделены на меньшие элементы. Вместе они составляют лексикон или словарь языка. Нетерминальные символы (non-terminal symbols) представляют собой группы или категории терминальных символов, таких как предложные группы (prepositional phrases), именные группы (noun phrases), глагольные группы (verb phrases) и т.д. Такие символы могут быть разложены на меньшие элементы, включая терминальные символы и другие нетерминальные символы. Таким образом, контекстно-свободная грамматика представляет собой генератор, преобразующий нетерминальные символы в строку символов.

Синтаксический анализ составляющих может быть представлен в виде дерева (рисунок 1).

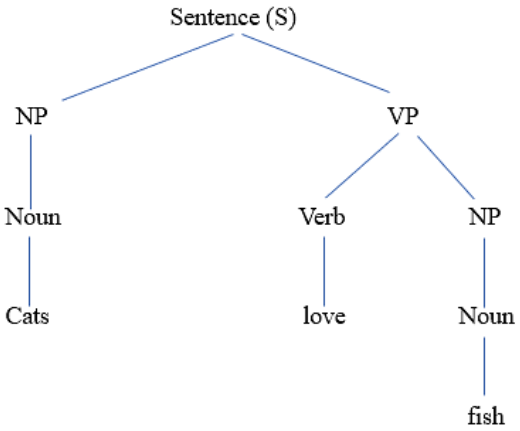

Рис. 1. Синтаксический анализ составляющих в виде дерева

Современные подходы конвертируют дерево в линейную форму, чтобы сделать возможным применение sequence-to-sequence моделей [4]. В линейном виде дерево выглядит следующим образом:

$$\big(\text{S (NP N)}\big(\text{VP V(NP N)}\big)\big). \quad (8)$$

Для применения алгоритмов парсинга составляющая контекстно-свободной грамматики должна быть приведена к нормальной форме Хомского. В нормальной форме Хомского правила грамматики могут принимать две следующие формы:

1. $A \rightarrow B\ C$, где A, B и C – нетерминальные символы;
2. $A \rightarrow a$, где A – нетерминальный символ, a – терминальный символ.

Ввиду данных правил, деревья, построенные из правил нормальной формы Хомского, будут бинарными. Потери информации при переходе к нормальной форме Хомского не происходит.

Любая контекстно-свободная грамматика может быть приведена к нормальной форме Хомского. Если правило контекстно-свободной грамматики содержит с правой стороны один нетерминальный символ и один терминальный, например, правило INF-VP → to VP (инфинитивная группа → to глагольная группа) может быть приведено к нормальной форме Хомского путем добавления нового нетерминального символа. Тогда правило INF-VP → to VP будет заменено на 2 правила: INF-VP → TO VP и TO → to.

Если правило контекстно-свободной грамматики содержит в правой части правила один нетерминальный символ, то оно преобразуется в правило с одним терминальным символом в правой части правила. Например, набор правил A → B, B → C, C → to преобразуется в A → to, устраняя тем самым избыточность информации.

Если в правой части правила более 2 символов, то оно преобразуется путем введения нового нетерминального символа. Например, правило A → B C a, где A, B, C – терминальные символы, a – нетерминальный, заменяется на 2 правила: A → X1 и X1 → B C.

3. Синтаксический парсинг зависимостей. Если парсинг составляющих основывается на контекстно-свободной грамматике, то парсинг зависимостей – на грамматике зависимостей (dependency grammar). Грамматика зависимостей представляет синтаксическую структуру предложения как совокупность направленных грамматических зависимостей между словами. В предложении выделяются главные и зависимые слова. Одно и то же слово в предложении может быть главным по отношению к одному слову и зависимым по отношению к другому. Корневым словом в предложении, которое не является зависимым по отношению ни к одному другому слову, является предикат, в роли которого, как правило, выступает глагол. На рисунке 2 отражена синтаксическая структура, предложения в соответствии с грамматикой зависимостей, где стрелки направлены от главных слов к зависимым.

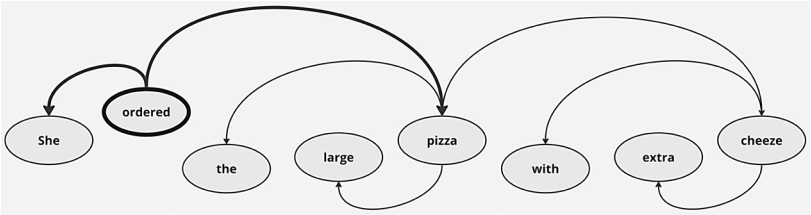

Рис. 2. Синтаксическая структура предложения в соответствии с грамматикой зависимостей

В форме синтаксического дерева данное предложение может быть представлено следующим образом (рисунок 3).

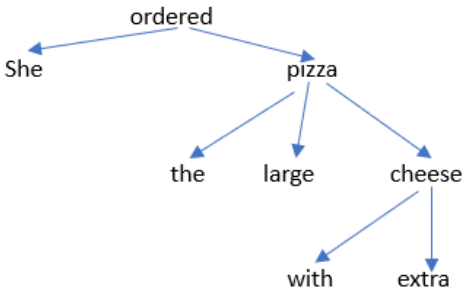

Рис. 3. Предложение в форме синтаксического дерева

Дерево зависимостей представляет собой ориентированный граф, удовлетворяющий следующим ограничениям:

1. Существует единственный определенный корневой узел, у которого нет входящих дуг.
2. За исключением корневого узла, каждая вершина имеет ровно одну входящую дугу.
3. Существует уникальный путь от корневого узла до каждой вершины в дереве зависимостей.

Такие ограничения обеспечивают связность структуры зависимостей, где у каждого слова не более одного главного слова и существует уникальный путь из корневого узла к каждому слову в предложении.

3.1. Алгоритмы синтаксического парсинга зависимостей. Основными алгоритмами парсинга зависимостей являются Transition-Based Dependency Parsing и Graph-Based Dependency Parsing.

Transition-Based Dependency Parsing основан на механизме shift-reduce. Алгоритм был предложен Ямада и Матцумото [11] и Нивре [12] на основе history-based parsing [13] и data-driven shift-reduce parsing [14]. Идея алгоритма заключается в сведении задачи парсинга к пошаговому прогнозированию наличия или отсутствия зависимости между двумя словами в предложении и направления выявленной зависимости. Парсер состоит из буфера входных токенов (слов), стека, предиктора и набора определенных зависимостей. В первоначальной конфигурации буфер входных токенов состоит из слов предложения в порядке, в котором они расположены в предложении, набор определенных зависимостей пуст, стек состоит из одного служебного элемента ROOT. Парсер обрабатывает предложение слева направо, последовательно сдвигая элементы из буфера в стек. На каждом шаге предиктор отправляет один токен из буфера в стек, анализирует два верхних элемента в стеке и принимает одно из следующих решений:

– назначить первое слово в стеке главным по отношению ко второму (левая дуга) и удалить второе слово из стека;

– если для верхнего слова в стеке уже назначены все зависимые слова, тогда назначить второе слово в стеке главным по отношению к первому слову (правая дуга) и удалить первое слово из стека;

– отложить обработку текущего слова, сдвинув его вниз по стеку.

Дополнительное условие для второго оператора (правая дуга) необходимо для того, чтобы слово не было извлечено из стека до того, как ему будут присвоены все его зависимые элементы.

Окончательное синтаксическое дерево будет составлено, когда буфер окажется пуст, а в стеке останется только служебный символ ROOT. Преимуществом алгоритма в сравнении с динамическими алгоритмами парсинга составляющих является его линейная сложность по длине предложения. Данный алгоритм представляет собой жадный алгоритм, так как предиктор делает один выбор на каждом шаге, данный выбор считается квазиоптимальным, повторно элементы не обрабатываются, и другие варианты построения зависимостей не рассматриваются. Некорректный выбор на одном шаге ведет к построению ошибочного дерева, без возможности вернуться назад и исправить ошибку. Кроме того, алгоритм возвращает только один вариант синтаксического дерева, в то время как, ввиду проблемы двусмысленности, возможно наличие более одного варианта корректных синтаксических деревьев.

Предиктор парсера может быть основан на классификаторе на основе признаков (classic feature-based algorithm) или на нейронном классификаторе. Алгоритм на основе признаков полагается на такие признаки, как форма слова, лемма, часть речи главного и зависимого слова; форма слова, лемма, часть речи для слов перед или между главным и зависимым словом; также учитывается состояние буфера входных токенов, стека и набора определенных зависимостей.

Признаки определяются вручную или с помощью обучения классификатора. Выбор признаков вручную несет в себе несколько проблем. Во-первых, слишком большое количество признаков может привести к переобучению и замедлению модели. Во-вторых, для корректного выбора признаков необходимы глубокие знания в области лингвистики [15].

Что касается классификатора, в последние годы произошел переход к нейронным классификаторам, который привел к значительному повышению точности предиктора [7]. Стандартный

алгоритм состоит в следующем: предложение проходит через энкодер, затем векторные представления двух первых слов из стека и первого слова из буфера конкатенируются и подаются в нейронную сеть прямого распространения. Кроме того, был разработан нетерпеливый (arc eager) transition-based алгоритм, который использует парсер SpaCy [5]. Главным отличием arc eager алгоритма от стандартного заключается в применении операторов к первому слову в стеке и первому слову в буфере. Это позволяет избавиться от условия предварительного назначения всех зависимых слов для зависимого слова в операторе «правая дуга», так как в данном алгоритме это зависимое слово вместо того, чтобы быть удаленным отправиться в стек и будет доступно для дальнейшей обработки. Такое изменение позволяет быстрее назначать зависимости слева направо (правая дуга) и ускоряет работу парсера. Для корректной работы парсера добавлен оператор «сокращение/reduce», необходимый для завершения процедуры парсинга в случае, если входной буфер оказался пуст.

Для повышения точности transition-based парсинга он может быть дополнен алгоритмом лучевого поиска (beam-search algorithm) [2], что смягчает проблему жадности у такого типа инструментов. Идея состоит в том, что вместо того, чтобы выбирать единственный оператор на каждом шаге, выбираются все операторы на каждом шаге, а затем все полученные частичные деревья оцениваются классификатором. Оценка каждого следующего дерева в одной последовательности рассчитывается как сумма оценки предшествующего дерева и оценки примененного к нему оператора:

$$TScore(i) = TScore(i-1) + OpScore(i-1), \quad (9)$$

где *TScore* – оценка дерева, *OpScore* – оценка оператора.

Количество деревьев ограничивается предустановленным лимитом – шириной луча. Чем шире луч, тем выше точность и ниже скорость. Когда ширина луча достигает лимита, новые деревья добавляются вместо худших, если оценка нового дерева выше оценки худшего дерева в луче. Процесс парсинга завершается, когда луч содержит только полные деревья входного предложения. Синтаксический анализатор выбирает из луча дерево с наивысшей оценкой и возвращает его в качестве окончательного вывода. Таким образом, парсеру не приходится принимать окончательные решения слишком рано, есть возможность вернуться на начальные этапы построения дерева и исправить ошибку, что значительно повышает точность парсера.

3.2. Алгоритм парсинга зависимостей на основе графа (graph-based parsing). Данный алгоритм разработан Макдональдом [16] на основе работы Эйснера [17]. В отличие от transition-based парсинга, полагающегося на жадные локальные решения, graph-based парсинг основан на оценке полного синтаксического дерева. Идея graph-based алгоритма заключается в представлении пространства возможных синтаксических деревьев в виде ориентированного графа (вершинами которого являются слова, а направленными ребрами – зависимости) и поиске в этом графе дерева с наилучшей оценкой. Общая оценка каждого дерева вычисляется как сумма весов отдельных зависимостей, из которых оно состоит. В результате, graph-based алгоритм рассматривает и оценивает все возможные зависимости в предложении, что является предпосылкой для более высокой точности по сравнению с transition-based парсингом.

Таким образом, нахождение наилучшего дерева зависимостей сводится к нахождению максимального остовного дерева, которое представляет собой подграф с максимальной суммой весов ребер, содержащий все вершины исходного графа и корневую вершину ROOT.

Так же, как и при transition-based парсинге, оценка зависимостей и полного дерева, как суммы оценок зависимостей, может производиться классификатором на основе признаков (classic feature-based algorithm) или на нейронном классификаторе. В feature-based алгоритме оценка зависимости (ребра графа) вычисляется как взвешенная сумма признаков:

$$\text{Score(S,e)} = \sum_{i=1}^{N} w_i f_i(S, e), \tag{10}$$

где S – предложение, e – зависимость (ребро), w – веса, f – признак, N – количество признаков.

Главной задачей является выявление релевантных признаков и их комбинаций. Могут использоваться следующие признаки: форма слова, лемма, часть речи главного и зависимого слова; расстояние между главным и зависимым словом, направление связи (слева направо или справа налево), векторные представления слов и т.д. По сравнению с transition-based парсингом, для graph-based парсинга возможен ограниченный набор признаков, так как алгоритм рассматривает признаки только самой пары слов (рассматриваемых

как потенциальная зависимость) и игнорирует признаки других слов в предложении, упуская тем самым глобальный контекст.

Нейронные классификаторы показывают более высокую точность. Предложение подается в энкодер, где для каждого токена строится глубокое контекстуализированное векторное представление. Ряд исследователей установили, что такие представления содержат информацию о синтаксической структуре предложения [18, 19]. Затем полученные представления передается в нейронную сеть, которая присваивает оценки каждой зависимости.

Дозат и Мэннинг предложили архитектуру нейронной сети, в которой использовали биафинное внимание (biaffine attention) вместо стандартного билинейного внимания [20]. В такой сети на вход подаются последовательность токенов, конкатенированных с тегами их частей речи:

$$x_i = v_i(word) \oplus v_i(tag), \tag{11}$$

где x_i – конкатенированный вектор, $v_i^{(word)}$ – представление токена, $v_i^{(tag)}$ – представление тега части речи токена.

Затем они обрабатываются энкодером в виде многослойной двунаправленной сети долгой краткосрочной памяти (LSTM):

$$r_i = \text{BiLSTM}(r_0, (x_1, \ldots, x_n))_i, \tag{12}$$

где r_i – конечное состояние, r_0 – первоначальное состояние, $(x_1,...,x_n)$ – конкатенированные вектора. Такой энкодер отражает глобальный контекст в локальных представлениях слов, что расширяет набор признаков за пределы непосредственно главного и зависимого слова, смягчая тем самым главный недостаток graph-based парсинга.

4. Постановка задачи извлечения ключевых слов с помощью парсера. Извлечение ключевых слов является фундаментальной задачей в обработке естественного языка (Natural Language Processing, NLP) и включает в себя выявление и извлечение наиболее релевантных и значимых слов или фраз из заданного текста. Парсеры играют ключевую роль в этом процессе, анализируя синтаксическую структуру текста и помогая выявлять ключевые компоненты, представляющие важные понятия или темы.

1. *Синтаксический анализ структуры для извлечения ключевых слов*. Для выполнения извлечения ключевых слов с использованием синтаксического анализа структуры, можно

сосредоточиться на конкретных синтаксических единицах, таких как именные фразы (NPs) [21] и глагольные фразы (VPs). Эти фразы часто являются хорошими кандидатами на роль ключевых слов, так как обычно содержат важную информацию о субъекте, объекте или действии в предложении. Например, рассмотрим предложение: "The swift fox jumps over the lazy dog." С помощью синтаксического анализа структуры будут выделены следующие фразы:

– Именные фразы (NPs): "The swift fox", "the lazy dog";
– Глагольная фраза (VP): "jumps over".

Из извлеченных фраз можно выделить следующие ключевые слова: "swift fox" (быстрая лиса); "lazy dog" (ленивая собака); "jumps over" (перепрыгивает через), как существенные компоненты данного предложения.

2. *Синтаксический анализ зависимостей для извлечения ключевых слов*. Синтаксический анализ зависимостей помогает определить отношения между субъектом, глаголом и объектом, а также другие существенные синтаксические зависимости, которые вносят вклад в общий смысл предложения. Ключевые слова могут быть извлечены из этих зависимостей, причем предпочтение отдается словам, несущим значительный семантический вес и играющим важные роли в структуре предложения [1].

В контексте извлечения ключевых слов текст преобразуется в граф, где вершины представляют собой возможные ключевые слова, а ребра – их отношения. Взаимосвязь между ключевыми фразами-кандидатами может быть определена по тому, как часто они встречаются вместе или насколько семантически близки.

Предположим, что строится ориентированный граф $G = (V, E)$, где – V множество вершин, а E – множество ребер. Оценка или важность вершины определяется как:

$$S(V_i) = (1 - d) + d * \sum_{j \in In(V_i)} \frac{1}{\left|Out(V_j)\right|} S(V_j), \quad (13)$$

где $In(V_i)$ – это набор вершин, которые указывают на V_i, а $Out(V_i)$ – это набор вершин, на которые указывает V_i. При этом d – это коэффициент затухания, который устанавливается в диапазоне от 0 до 1.

5. Разработка алгоритма оптимизации извлечения ключевых слов на основе парсера. Исходя из предположения, что ключевое слово (ключевые фразы) обычно является существительным

или именной фразой, предлагается алгоритм оптимизации для извлечения ключевых слов, характеризующийся использованием парсера для фильтрации неподходящих фраз, которые не являются именными фразами. Цель состоит в том, чтобы проверить влияние парсера на производительность алгоритма извлечения ключевых слов.

В сформулированном листинге 1 шаги алгоритма описаны в виде псевдокода. Входными данными алгоритма является текст, а выходными данными является окончательный список ключевых слов.

Сначала функция *preprocessing* обрабатывает текст для удаления ненужных символов, знаков препинания, стоп-слов и переводов текста в нижний регистр. После этого слова-кандидаты, *n*-граммы (длиной от 2 до 4), извлекаются с помощью функции *extract_ngrams*, которая применяет весовую функцию *TF*, определенную по следующей формуле:

$$TF = \frac{n_t^i}{\sum_k n_k^i}, \tag{14}$$

где n_k – общее количество *n*-грамм (с i элементов) в документе, а n_t – количество вхождений токена *t* в документ.

После этого текст необходимо разделить на предложения, а затем каждое предложение анализируется на компоненты для извлечения именных фраз *NP*. Для этого весь текст анализируется парсером, который, в свою очередь, возвращает предложения, составляющие текст, а также компоненты и атрибуты каждого предложения включая именных фраз.

Именные фразы в их исходной форме не подходят в качестве ключевых слов, так как они могут содержать артикли или быть слишком длинными, поэтому они обрабатываются, опуская артикли, длинные словосочетания и фразы, которые не начинаются с существительного или прилагательного (листинг 1, строки 10-20).

В итоге получается список именных фраз на уровне текста, которые используются в качестве фильтра (строки 21–32 в листинге). Если ключевая фраза-кандидат соответствует одной из именных фраз, она сохраняется. Если именная фраза является частью фразы-кандидата, в окончательном списке сохраняется именная фраза, поскольку именная фраза имеет более полную структуру.

```
1   Ввод текст
2   Вывод список ключевых фраз KWS
3:  T* = preprocessing(T) //удаление ненужных символов, знаков
      препинания, стоп-слов и перевод текста в нижний регистр
4:  Candidates = extract_ngrams(T*, (2,4)) // формула 14
5:  S = splitter(T*) // Разделить текст на список предложения S
6:  NP = [] // Пустой список для хранения именных фраз NP
7:  Foreach sentence in S do:
8:      NP ← get_noun_phrase(sentence) //Разбор предложения с
          помощью парсера и сохранение именной фразы в списке NP
9:  End
10: // преобразования именных фраз в ключевые слова
11: FNP = [] //Список для хранения отфильтрованных именных фраз.
12: Foreach np in NP do
13:     if len(np) == 1:
14:         if np[0].pos_ in ['NOUN', 'PROPN', 'ADJ']: // первое слово np
            — существительное, имя собственное или прилагательное
15:             FNP.append(np.text.strip())
16:     elif 1 <len(np)< 6: // если длина np не превышает 6 слов
17:         if np[0].pos_ in ['DET']: //первый термин - это артикль
18:             FNP.append(np[1:].text) // удалить артикль
19:         Else
20:             FNP.append(np.text.strip())
21:  // Проверка совпадения именной фразы с кандидатами на n-граммы
22: Foreach candidate in candidates do:
23:     If candidate in FNP do
24:         KWS []← candidate // добавить кандидата в список
25:     Else // проверка наличия общих термов между кандидатом и
          именной фразой
26:         Foreach np in FNP do
27:                 If np ∩ candidate ≠ ∅ do:
28:                     KWS []← np // добавить np в список
29:                 End
30:         End
31:     End
32: End
```

Листинг 1. Псевдокод алгоритма оптимизации извлечения и фильтрации ключевых слов с помощью парсеров

Этот процесс способствует повышению точности алгоритма извлечения ключевых слов, поскольку он гарантирует, что слова-кандидаты представляют собой не просто последовательность слов, а составные слова в однородном контексте. Далее, в экспериментальных исследованиях проводятся тесты для подтверждения этого утверждения.

6. Экспериментальные исследования. Разработанный алгоритм извлечения ключевых слов реализован с использованием трех популярных парсеров, а именно:

SpaCy – это бесплатная библиотека с открытым исходным кодом для расширенной обработки естественного языка (NLP) в Python.

Парсер SpaCy – это компонент библиотеки SpaCy, который отвечает за анализ грамматической структуры предложений. Реализация парсера SpaCy выполняет синтаксический анализ зависимостей и включает в себя методы машинного обучения для прогнозирования в процессе разбора. Он использует статистическую модель, обученную на размеченных данных, для предсказания наиболее вероятного перехода на каждом шаге. Парсер использует вариант немонотонной дуговой системы с переходами [22], с добавлением перехода "break" для выполнения сегментации предложений. Для возможности предсказания неверных разборов парсер использует псевдопроективное преобразование зависимостей, предложенное в [23].

AllenNLP [10] – это платформа для исследований в области глубокого обучения методов обработки естественного языка. AllenNLP не имеет встроенного парсера, подобного SpaCy, однако предоставляет предварительно обученные модели для синтаксического парсинга составляющих и парсинга зависимостей. Парсер зависимостей следует модели глубокого биаффинного внимания для нейронного синтаксического анализа зависимостей, который использует нейронное внимание в простом графовом парсере зависимостей [20]. Парсер составляющих построен на основе минимальной нейронной модели, основанной на независимой оценке меток. Модель использует встроенную процедуру ELMo и кодирует последовательность текста с помощью стекового Seq2SeqEncoder.

Stanza [9] – это пакет для анализа естественного языка, написанный на Python, созданный Стэнфордской группой NLP. Он предоставляет несколько инструментов для анализа естественного текста, одним из которых является предоставление синтаксической структуры в виде дерева зависимостей. Как и в AllenNLP парсер

зависимостей в Stanza реализует Bi-LSTM-сеть, основанную на глубокой биаффинной нейронной модели, которая относится к категории парсеров зависимостей на основе графов [20].

Предложенный авторами алгоритм сравнивается с двумя известными методами, Yake и Rake.

Метод Yake (Yet Another Keyword Extractor) [24] использует комбинацию статистических и лингвистических признаков для определения важности слов или фраз в тексте. Он учитывает как частоту и распределение терминов внутри документа, так и их семантическую связь. Метод использует подход скользящего окна для выявления кандидатов на ключевые фразы, а затем применяет функцию оценки для их ранжирования по степени релевантности.

Метод Rake (Rapid Automatic Keyword Extraction) [25] использует простой алгоритм, который опирается на распределение слов в тексте для определения фраз кандидатов. Сначала текст разбивается на отдельные слова, а затем генерируются фразы-кандидаты, идентифицируя последовательности слов, разделенные стоп-словами или знаками пунктуации. Алгоритм присваивает оценки этим фразам на основе их частоты и степени совместного встречаемости слов.

Для Yake из каждого текста извлекаются первые 20 *n*-грамм (от 1 до 3). То же самое и с Rake, но Rake не позволяет указать количество извлекаемых слов. Не выполнялась никакая предварительная обработка текстов, чтобы не повлиять на структуру, но после получения именных фраз выполняется некоторая обработка текста, например, такая как удаление стоп-слов.

Набор данных Inspec [26] состоит из 2000 рефератов статей из научных журналов по компьютерным наукам. Каждому документу присвоены два набора ключевых слов: контролируемые ключевые слова, которые являются назначенными вручную ключевыми словами и появляются в тезаурусе Inspec, но могут не появляться в документе. Неконтролируемые ключевые слова свободно назначаются редакторами.

Для оценки результатов отслеживалось точное совпадение, при котором автоматически извлеченная ключевая фраза из документа должна точно соответствовать ключевой фразе эталонного паттерна для документа. Традиционно извлечение ключевых слов по своей природе является проблемой ранжирования. Исходя из этого, для определения эффективности предлагаемого алгоритма используется одна из наиболее часто используемых мер качества для ранжирования Средняя точность на *K* элементах «*Mean average precision at K,*

MAP@K». Таким образом, совпадения проверяются среди первых *K* ключевых слов, возвращаемых алгоритмом. Для оценки *MAP@K* сначала подсчитывается точность на *K* элементах p@k. Базовая метрика качества ранжирования для одного объекта определяется следующим выражением:

$$pr_k = \frac{\sum_{k=1}^{K} r_{true}(e_i)}{k}, \tag{15}$$

где e_i это элемент (ключевое слово) $e \in E$, который в результате перестановки оказался на *i*-ой позиции, $r_{true}(e_i)$ – функция равная 1, если *e* релевантен, 0 – в противном случае. Недостаток этой метрики заключается в том, что не учитываются позиции правильных элементов (порядок элементов). Далее на основе pr_K рассчитывается average precision at K (apr_K):

$$apr_k = \frac{1}{k}\sum_{k=1}^{K} r_{true}\pi^{-1}(k) \cdot pr_k. \tag{16}$$

Такая мера учитывает позиции элементов, но качество ранжирования оценивается для отдельно взятого объекта. Для того чтобы посчитать *Mean average precision at K* для *N* различных объектов вычисляется среднее по apr@K для каждого:

$$map_k = \frac{1}{N}\sum_{i=1}^{N} apr_k^i, \tag{17}$$

где apr_K^i – это apr_K для *i*-го объекта. Под MAP подразумевается Mean average precision по всем объектами и всем элементам.

7. Результаты экспериментальных исследований. Результаты сравнительного анализа алгоритмов извлечения ключевых фраз представлены в таблицах 1 и 2. Сравнивается производительность предложенного алгоритма при использовании трех разных парсеров с производительностью двух исследованных алгоритмов Yake и Rake.

Разработанный алгоритм (с использованием любого парсера) превосходит методы Yake и Rake, подтверждая утверждение о том, что большинство ключевых слов являются именными фразами. Наилучшие результаты достигаются с парсером SpaCy, как с точки зрения точности, так и с точки зрения скорости.

Таблица 1. Точность алгоритмов, измеренная с помощью mAP@K

Метод	map@k			
K	@1	@5	@10	@20
TF-SpaCy	**0.36**	**0.15**	**0.095**	0.077
TF-Stanza	0.303	0.131	0.083	0.06
TF-AllenNLP	0,299	0,128	0,081	0,064
Yake	0.267	0.127	0.084	**0.0775**
Rake	0.169	0.103	0.081	0.074

Результаты Yake и Rake приближается к показателям TF-SpaCy при определении правильных ключевых слов в топ-20 ($K = 20$).

Результаты проведенного исследования проиллюстрированы в таблице 2. В строке “Gold keywords” выделены полужирным шрифтом ключевые слова, определенные из текста экспертами. Что касается слов вне текста, то в данной работе они не учитываются.

Из таблицы 2 видно, что разработанный алгоритм оптимизации определения ключевых слов находит 4 правильных слова из 5 с помощью SpaCy, в то время как AllenNLP определяет только 2. Stanza правильно определяет “Bar code labels”, в то время как SpaCy определяет часть этого как “code labels”. AllenNLP не распознает “food processing” и “Bar code labels”, потому что он рассматривает "Fresh tracks [food processing] Bar code labels" как одну фразу на основе существительного.

Таблица 2. Текст из набора данных в качестве примера, показывающего эталонные ключевые слова и ключевые слова, определенные каждым алгоритмом

Текст	Fresh tracks [food processing] Bar code labels and wireless terminals linked to a centralized database accurately track meat products from receiving to customers for Farmland Foods
Gold keywords	[**'food processing'**, **'bar code labels', 'wireless terminals'**, **'Farmland Foods'**, 'automatic data capture', 'Intermec Technologies', **'bar codes'**, 'data acquisition', 'food processing industry', 'mobile computing', 'production control', '']
TF-SpaCy	['Fresh tracks', **'food processing'**, **'code labels'**, **'wireless terminals'**, 'centralized database', 'meat products', **'Farmland Foods'**]
TF-AllenNLP	[**'wireless terminals'**, **'Farmland Foods',** 'centralized database', 'Fresh tracks', 'meat products']
TF-Stanza	[**'wireless terminals'**, **'Farmland Foods', 'food processing'**, 'customers for Farmland Foods', 'Fresh tracks', 'meat products', 'centralized database', **'Bar code labels'**]
Yake	[**'Bar code labels'**, 'wireless terminals linked', 'centralized database accurately', 'database accurately track', 'accurately track meat', 'track meat products', **'Bar code'**, **'Farmland Foods'**, 'customers for Farmland', **'food processing'**]
Rake	['wireless terminals linked', **'bar code labels'**, 'fresh tracks', **'food processing'**, **'farmland foods'**, 'receiving', 'customers']

Алгоритмы Yake и Rake неправильно определили глагол «linked» как часть ключевого слова «wireless terminals», тогда как трем парсерам удалось идентифицировать его правильно и исключить из фразы.

Хотя SpaCy превосходит другие парсеры в задаче определения именных фраз, нельзя однозначно утверждать, что парсеры, основанные на переходах, всегда превосходят парсеры, основанные на графах. Это зависит от нескольких факторов, включая структуру и сложность анализируемого текста, эффективность реализации парсера и его внутренних алгоритмов.

Парсеры на основе переходов постепенно строят дерево и принимают решения о разборе на основе локального контекста, что может помочь распознавать и извлекать именные фразы, имеющие явные зависимости в пределах короткого расстояния. Таким образом, поскольку текст, использованный в этих экспериментах, относительно короткий, он содержит простые предложения, в которых преобладают локальные зависимости, подходящие для данного типа синтаксического анализатора. С другой стороны, парсеры на основе графов учитывают всю структуру предложения при построении дерева зависимостей. Этот глобальный контекст может быть полезен для определения неявных зависимостей между словами, которые охватывают несколько слов или предложений, как в примере AllenNLP в таблице 2.

Анализ времени выполнения. Рисунки 4 и 5 и таблица 3 демонстрируют различие в скорости работы предложенного алгоритма по сравнению с методами Yake и Rake при обработке разного количества документов. Поскольку время выполнения алгоритмов Rake, Yake и SpaCy очень мало по сравнению с остальными алгоритмами, для наглядности их временные характеристики продемонстрированы в другом масштабе (рисунок 5).

Время выполнения предложенного авторами алгоритма представляет собой сумму времен выполнения нескольких операций (листинг 1): время извлечения *n*-грамм в качестве ключевых слов-кандидатов; время анализа текста на компоненты предложений и именных фраз; время фильтрации этих именных фраз; время фильтрации ключевых слов-кандидатов по именным фразам.

Таблица 3. Анализ времени выполнения алгоритмов

	Время выполнения в секундах						
Кол-во документов	10	20	30	40	50	100	2000
TF-SpaCy	1	1,33	2,2	3,58	3,56	8	102
TF-AllenNLP	71	80	147	222	295	481	10614
TF-Stanza	223	414	585	820	1071	1954	14625
Yake	2,39	3,08	3,61	5,6	9	14	207
Rake	0,012	0,035	0,047	0,056	0,071	0,094	2 sec

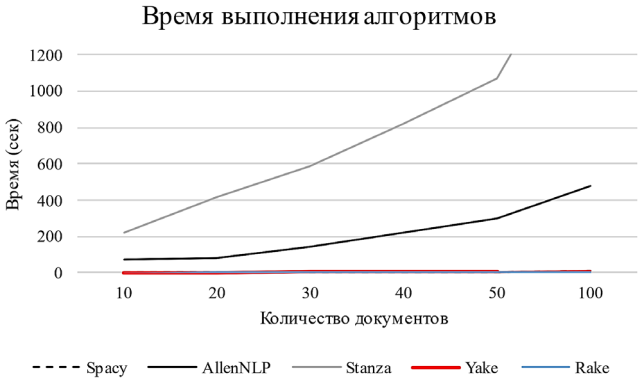

Рис. 4. Сравнение времени выполнения алгоритмов извлечения ключевых слов

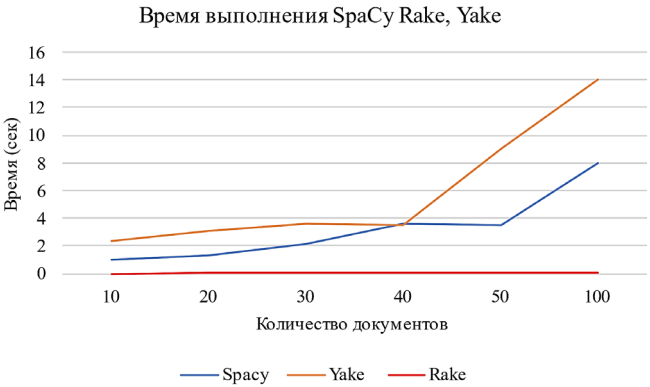

Рис. 5. Сравнение времени выполнения алгоритмов SpaCy, Yake и Rake

Хотя статистические алгоритмы работают быстрее, чем алгоритмы на основе парсера, поскольку их работа не требует анализа текста, предложенный в данной статье алгоритм с парсером Spacy (TF-SpaCy) превзошёл все рассмотренные канонические алгоритмы кроме Rake.

Время выполнения предложенного алгоритма с парсером SpaCy значительно меньше, чем у двух других парсеров, потому что парсеры, которые используют переходы, применяют детерминированный или машинно-обучаемый набор действий для пошагового построения дерева зависимостей. Они обычно работают быстрее и требуют меньше памяти по сравнению с парсерами, основанными на графах, что делает их более эффективными для анализа больших объемов

текста. С другой стороны, AllenNLP и Stanza используют модели парсинга на основе графов, которые полагаются на миллионы созданных вручную признаков, что ограничивает их способность к обобщению и замедляет скорость анализа по сравнению с парсерами на основе переходов.

Задача достижения баланса между точностью и скоростью лингвистического парсера является открытой темой, требующей дальнейших исследований в связи с важностью данной проблемы для повышения эффективности текстового анализа, особенно в приложениях, требующих точности при работе в реальном масштабе времени. С этой целью авторы планируют проведение дальнейших исследований возможных решений для достижения такого баланса.

Планируется рассмотреть дистилляцию нейронных сетей для быстрого разбора зависимостей, а также использование атомарных признаков, таких как униграммы слов и униграммы POS-тегов, вместо использования большого количества признаков, созданных вручную. Кроме того, планируется рассмотреть интеграцию подходов на основе графов и переходов с использованием преимуществ глубокого контекстного представления.

Заключение. В данной статье исследованы два основных типа лингвистического анализа: синтаксический анализ составляющих и анализ зависимостей. Представлено аналитическое сравнение нескольких известных парсеров (SpaCy, Stanza, AllenNLP), принадлежащих к различным подходам обработки текстовой информации. Также был разработан алгоритм извлечения ключевых слов на основе частоты, отличающийся применением функции извлечения именных фраз, предоставляемой парсером, для извлечения уточненного списка именной фразы. Этот список используется в качестве фильтра для отсеивания неподходящих ключевых слов-кандидатов, извлеченных на основе частоты, что позволяет повысить точность извлечения ключевых слов.

Множество экспериментов проведено с целью изучения влияния использования парсера на время и эффективность извлечения ключевых фраз по сравнению с двумя известными методами YAKE и RAKE. Результаты экспериментов подтвердили тот факт, что использование парсера существенно повышает точность алгоритма извлечения ключевых слов. Также отмечается, что эффективность зависит от типа парсера, контекста и длины обрабатываемого текста.

В данной работе основное внимание уделялось производительности парсера в качестве инструмента для извлечения ключевых фраз без углубления в анализ внутренних алгоритмов. Это

не дает полной картины различий в производительности анализаторов, а лишь показывает различия с точки зрения их применения. В будущем требуется провести дополнительный анализ данных алгоритмов с целью поиска решений проблемы снижения их временной сложности.

Литература

1. Brown T., Mann B., Ryder N., Subbiah M., Kaplan J.D., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A., et al. Language models are few-shot learners // Advances in neural information processing systems. 2020. vol. 33. pp. 1877–1901.
2. Zhang Y., Clark S. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing // Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing. 2008. pp. 562–571.
3. Gao L., Madaan A., Zhou S., Alon U., Liu P., Yang Y., Callan J., Neubig G. Pal: Program aided language models. 2023. pp. 10764–10799.
4. Kravchenko Yu.A., Bova V.V., Kuliev E.V., Rodzin S.I. Simulation of the semantic network of knowledge representation in intelligent assistant systems based on ontological approach // Futuristic Trends in Network and Communication Technologies: Third International Conference, FTNCT. 2021. pp. 241–252.
5. Chen D., Manning C.D. A fast and accurate dependency parser using neural networks // Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014. pp. 740–750.
6. Kiperwasser E., Goldberg Y. Simple and accurate dependency parsing using bidirectional LSTM feature representations // Transactions of the Association for Computational Linguistics. 2016. vol. 4. pp. 313–327.
7. Kulmizev A., de Lhoneux M., Gontrum J., Fano E., Nivre J. Deep Contextualized Word Embeddings in Transition-Based and Graph-Based Dependency Parsing – A Tale of Two Parsers Revisited // arXiv preprint arXiv: 07397. 2019.
8. Vasiliev Y. Natural language processing with Python and SpaCy: A practical introduction. No Starch Press, 2020. 216 p.
9. Qi P., Zhang Y., Zhang Y., Bolton J., Manning C.D. Stanza: A Python natural language processing toolkit for many human languages // arXiv preprint arXiv: 07082. 2020.
10. Gardner M., Grus J., Neumann M., Tafjord O., Dasigi P., Liu N., Peters M., Schmitz M., Zettlemoyer L. Allennlp: A deep semantic natural language processing platform // arXiv preprint arXiv: 07640. 2018.
11. Yamada H., Matsumoto Y. Statistical dependency analysis with support vector machines // Proceedings of the eighth international conference on parsing technologies. 2003. pp. 195–206.
12. Nivre J. An efficient algorithm for projective dependency parsing // Proceedings of the eighth international conference on parsing technologies. 2003. pp. 149–160.
13. Kim G., Baldi P., McAleer S. Language models can solve computer tasks. arXiv preprint arXiv:2303.17491. 2023.
14. Liu B., Jiang Y., Zhang X., Liu Q., Zhang S., Biswas J., Stone P. Llm+p: Empowering large language models with optimal planning proficiency. arXiv preprint arXiv:2304.11477. 2023.
15. Pei W., Ge T., Chang B. An effective neural network model for graph-based dependency parsing // Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing. 2015. vol. 1. pp. 313–322.

16. McDonald R., Crammer K., Pereira F. Online large-margin training of dependency parsers // Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL'05). 2005. pp. 91–98.
17. Eisner J. Three new probabilistic models for dependency parsing: An exploration // arXiv preprint cmp-lg/ 9706003. 1997.
18. Tenney I., Das D., Pavlick E. BERT rediscovers the classical NLP pipeline // arXiv preprint arXiv: 05950. 2019.
19. Hewitt J., Manning C.D. A structural probe for finding syntax in word representations // Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2019. vol. 1. pp. 4129–4138.
20. Dozat T., Manning C.D. Deep biaffine attention for neural dependency parsing // arXiv preprint arXiv: 01734. 2016.
21. Mao X., Huang S., Li R., Shen L. Automatic keywords extraction based on co-occurrence and semantic relationships between words // IEEE Access. 2020. vol. 8. pp. 117528–117538.
22. Yang S., Nachum O., Du Y., Wei J., Abbeel P., Schuurmans D. Foundation models for decision making: Problems, methods, and opportunities. arXiv preprint arXiv:2303.04129. 2023.
23. Honnibal M., Johnson M. An Improved Non-monotonic Transition System for Dependency Parsing. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing – Lisbon, Portugal: Association for Computational Linguistics. 2015. pp. 1373–1378. DOI: 10.18653/v1/D15-1162.
24. Campos R., Mangaravite V., Pasquali A., Jorge A., Nunes C., Jatowt A. YAKE! Keyword extraction from single documents using multiple local features // Information Sciences. 2020. vol. 509. pp. 257–289.
25. Rose S., Engel D., Cramer N., Cowley W. Automatic keyword extraction from individual documents // Text mining: applications theory. 2010. pp. 1–20.
26. Hulth A. Improved automatic keyword extraction given more linguistic knowledge // Proceedings of the 2003 conference on Empirical methods in natural language processing. 2003. pp. 216–223.

Кравченко Даниил Юрьевич — аспирант, кафедра систем автоматизированного проектирования, Федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет». Область научных интересов: технологии искусственного интеллекта, инженерия знаний. Число научных публикаций — 40. dkravchenko@sfedu.ru; переулок Некрасовский, 44, 347922, Таганрог, Россия; р.т.: 8(8634)371-651.

Кравченко Юрий Алексеевич — профессор, кафедра систем автоматизированного проектирования, Федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет». Область научных интересов: технологии искусственного интеллекта, инженерия знаний. Число научных публикаций — 240. krav-jura@yandex.ru; переулок Некрасовский, 44, 347922, Таганрог, Россия; р.т.: 8(8634)371-651.

Мансур Али — аспирант, кафедра систем автоматизированного проектирования, Федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет». Область научных интересов: технологии искусственного интеллекта, инженерия знаний. Число научных публикаций — 15. mansur@sfedu.ru; переулок Некрасовский, 44, 347922, Таганрог, Россия; р.т.: 8(9880)158-697.

Мохаммад Жуман — аспирант, кафедра систем автоматизированного проектирования, Федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет». Область научных интересов: технологии искусственного интеллекта, инженерия знаний. Число научных публикаций — 14. zmohammad@sfedu.ru; переулок Некрасовский, 44, 347922, Таганрог, Россия; р.т.: +7(918)543-3526.

Павлов Николай Сергеевич — аспирант, кафедра систем автоматизированного проектирования, Федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет». Область научных интересов: технологии искусственного интеллекта, инженерия знаний. npavlov@sfedu.ru; переулок Некрасовский, 44, 347922, Таганрог, Россия; р.т.: +7(8634)371-651.

Поддержка исследований. Исследование выполнено за счет гранта Российского научного фонда № 23-21-00089, https://rscf.ru/project/23-21-00089/ в Южном федеральном университете.

DOI 10.15622/ia.23.2.6

D. KRAVCHENKO, YU. KRAVCHENKO, A. MANSOUR, J. MOHAMMAD, N. PAVLOV

ALGORITHM FOR OPTIMIZATION OF KEYWORD EXTRACTION BASED ON THE APPLICATION OF A LINGUISTIC PARSER

Kravchenko D., Kravchenko Yu., Mansour A., Mohammad J., Pavlov N. **Algorithm for Optimization of Keyword Extraction Based on the Application of a Linguistic Parser.**

Abstract. This article presents an analytical comparison between constituency parsing and dependency parsing – two types of parsing used in the field of natural language processing (NLP). The study introduces an algorithm to enhance keyword extraction, employing the noun phrase extraction feature of the parser to filter out unsuitable phrases. This algorithm is implemented using three different parsers: Spacy, AllenNLP and Stazna. The effectiveness of this algorithm was compared with two popular methods (Yake, Rake) on a dataset of English texts. Experimental results show that the proposed algorithm with the SpaCy parser is superior to other keyword extraction algorithms in terms of accuracy and speed. For the AllenNLP and Stanza parsers, our algorithm is also more accurate, but requires much longer execution time. The results obtained allow us to evaluate in more detail the advantages and disadvantages of the parsers studied in the work, as well as to determine directions for further research. The running time of the SpaCy parser is significantly less than the other two parsers because parsers that use transitions for deterministic or machine-learned set of actions to build the dependency tree step by step. They are typically faster and require less memory than graph-based parsers, making them more efficient for parsing large amounts of text. On the other hand, AllenNLP and Stanza use graph-based parsing models that rely on millions of features, which limits their ability to generalize and slows down the speed of analysis compared to transition-based parsers. The task of achieving a balance between the accuracy and speed of a linguistic parser is an open topic that requires further research due to the importance of this problem for improving the efficiency of text analysis, especially in applications that require real-time accuracy. To this end, the authors plan to conduct further research into possible solutions to achieve this balance.

Keywords: constituency parsing, dependency parsing, keyword extraction, natural language processing, SpaCy, Stanza, AllenNLP.

References

1. Brown T., Mann B., Ryder N., Subbiah M., Kaplan J.D., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A., et al. Language models are few-shot learners. Advances in neural information processing systems. 2020. vol. 33. pp. 1877–1901.
2. Zhang Y., Clark S. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing. 2008. pp. 562–571.
3. Gao L., Madaan A., Zhou S., Alon U., Liu P., Yang Y., Callan J., Neubig G. Pal: Program aided language models. 2023. pp. 10764–10799.
4. Kravchenko Yu.A., Bova V.V., Kuliev E.V., Rodzin S.I. Simulation of the semantic network of knowledge representation in intelligent assistant systems based on ontological approach. Futuristic Trends in Network and Communication Technologies: Third International Conference, FTNCT. 2021. pp. 241–252.

5. Chen D., Manning C.D. A fast and accurate dependency parser using neural networks. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014. pp. 740–750.
6. Kiperwasser E., Goldberg Y. Simple and accurate dependency parsing using bidirectional LSTM feature representations. Transactions of the Association for Computational Linguistics. 2016. vol. 4. pp. 313–327.
7. Kulmizev A., de Lhoneux M., Gontrum J., Fano E., Nivre J. Deep Contextualized Word Embeddings in Transition-Based and Graph-Based Dependency Parsing – A Tale of Two Parsers Revisited. arXiv preprint arXiv: 07397. 2019.
8. Vasiliev Y. Natural language processing with Python and SpaCy: A practical introduction. No Starch Press, 2020. 216 p.
9. Qi P., Zhang Y., Zhang Y., Bolton J., Manning C.D. Stanza: A Python natural language processing toolkit for many human languages. arXiv preprint arXiv: 07082. 2020.
10. Gardner M., Grus J., Neumann M., Tafjord O., Dasigi P., Liu N., Peters M., Schmitz M., Zettlemoyer L. Allennlp: A deep semantic natural language processing platform. arXiv preprint arXiv: 07640. 2018.
11. Yamada H., Matsumoto Y. Statistical dependency analysis with support vector machines. Proceedings of the eighth international conference on parsing technologies. 2003. pp. 195–206.
12. Nivre J. An efficient algorithm for projective dependency parsing. Proceedings of the eighth international conference on parsing technologies. 2003. pp. 149–160.
13. Kim G., Baldi P., McAleer S. Language models can solve computer tasks. arXiv preprint arXiv:2303.17491. 2023.
14. Liu B., Jiang Y., Zhang X., Liu Q., Zhang S., Biswas J., Stone P. Llm+p: Empowering large language models with optimal planning proficiency. arXiv preprint arXiv:2304.11477. 2023.
15. Pei W., Ge T., Chang B. An effective neural network model for graph-based dependency parsing. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing. 2015. vol. 1. pp. 313–322.
16. McDonald R., Crammer K., Pereira F. Online large-margin training of dependency parsers. Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL'05). 2005. pp. 91–98.
17. Eisner J. Three new probabilistic models for dependency parsing: An exploration. arXiv preprint cmp-lg/ 9706003. 1997.
18. Tenney I., Das D., Pavlick E. BERT rediscovers the classical NLP pipeline. arXiv preprint arXiv: 05950. 2019.
19. Hewitt J., Manning C.D. A structural probe for finding syntax in word representations. Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2019. vol. 1. pp. 4129–4138.
20. Dozat T., Manning C.D. Deep biaffine attention for neural dependency parsing. arXiv preprint arXiv: 01734. 2016.
21. Mao X., Huang S., Li R., Shen L. Automatic keywords extraction based on co-occurrence and semantic relationships between words. IEEE Access. 2020. vol. 8. pp. 117528–117538.
22. Yang S., Nachum O., Du Y., Wei J., Abbeel P., Schuurmans D. Foundation models for decision making: Problems, methods, and opportunities. arXiv preprint arXiv:2303.04129. 2023.
23. Honnibal M., Johnson M. An Improved Non-monotonic Transition System for Dependency Parsing. Proceedings of the 2015 Conference on Empirical Methods in

Natural Language Processing – Lisbon, Portugal: Association for Computational Linguistics. 2015. pp. 1373–1378. DOI: 10.18653/v1/D15-1162.
24. Campos R., Mangaravite V., Pasquali A., Jorge A., Nunes C., Jatowt A. YAKE! Keyword extraction from single documents using multiple local features. Information Sciences. 2020. vol. 509. pp. 257–289.
25. Rose S., Engel D., Cramer N., Cowley W. Automatic keyword extraction from individual documents. Text mining: applications theory. 2010. pp. 1–20.
26. Hulth A. Improved automatic keyword extraction given more linguistic knowledge. Proceedings of the 2003 conference on Empirical methods in natural language processing. 2003. pp. 216–223.

Kravchenko Daniil — Postgraduate, Department of computer-aided design, Southern Federal University. Research interests: artificial intelligence technologies, knowledge engineering. The number of publications — 40. dkravchenko@sfedu.ru; 44, Nekrasovsky Lane, 347922, Taganrog, Russia; office phone: 8(8634)371-651.

Kravchenko Yury — Professor, Department of computer-aided design, Southern Federal University. Research interests: artificial intelligence technologies, knowledge engineering. The number of publications — 240. krav-jura@yandex.ru; 44, Nekrasovsky Lane, 347922, Taganrog, Russia; office phone: 8(8634)371-651.

Mansour Ali Mahmoud — Postgraduate, Department of computer-aided design, Southern Federal University. Research interests: artificial intelligence technologies, knowledge engineering. The number of publications — 15. mansur@sfedu.ru; 44, Nekrasovsky Lane, 347922, Taganrog, Russia; office phone: 8(9880)158-697.

Mohammad Juman — Postgraduate, Department of computer-aided design, Southern Federal University. Research interests: artificial intelligence technologies, knowledge engineering. The number of publications — 14. zmohammad@sfedu.ru; 44, Nekrasovsky Lane, 347922, Taganrog, Russia; office phone: +7(918)543-3526.

Pavlov Nikolai — Postgraduate, Department of computer-aided design, Southern Federal University. Research interests: artificial intelligence technologies, knowledge engineering. npavlov@sfedu.ru; 44, Nekrasovsky Lane, 347922, Taganrog, Russia; office phone: +7(8634)371-651.

Acknowledgements. The study was performed with the grant from the Russian Science Foundation № 23-21-00089, https://rscf.ru/project/23-21-00089/ in the Southern Federal University.