Transaction of Azerbaijan National Academy of Sciences, Series of Physical-Technical and Mathematical Sciences: Informatics and Control Problems, Vol. XXXII, No.3, 2012
www.science.az/cyber/journal/2012/3-04.pdf

УДК 004.272.26:004.912

Т.П. ГУРБАНОВ, Э.С. КЛЫШИНСКИЙ

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ СОСТАВЛЕНИЯ СЛОВАРЯ ГЛАГОЛЬНОГО УПРАВЛЕНИЯ ДЛЯ НОВОСТНЫХ ТЕКСТОВ НА АНГЛИЙСКОМ ЯЗЫКЕ

Описывается решение задачи выделения групп слов из синтаксически связанных конструкций, которые в свою очередь выделяются из содержимого текстового файла с исходными данными. Решение задачи производится при помощи языка программирования Python с использованием библиотек NLTK и Pymorphy.
Ключевые слова: разбор, анализ, текст, конструкции, грамматика, семантика, обработка, NLP

1. Введение. При автоматической обработке текстов на естественном языке сочетание слов несет очень важную информацию. В зависимости от контекста рассматриваемое слово может нести различную семантическую информацию и, как следствие, обрабатываться по-разному. В связи с этим на данный момент был разработан целый ряд методов, позволяющих автоматически выделить информацию о сочетаемости слов. Вид сочетаемости зависит от задач, для которых проводится выделение сочетаний. Так, например, при снятии с текста лексической неоднозначности (например, определении части речи слов) используется информация о частоте встречаемости n-грамм – групп из n последовательно идущих слов [1-3]. В некоторых случаях используется информация о разрывных группах [4]. Для сбора частотной информации используется множество текстов, в которых каждому слову вручную приписывается единственно правильный набор лексической информации – размеченный корпус текстов. Создание подобных корпусов является проектом государственного масштаба и такой же важности. Примерами подобных корпусов могут служить Национальный корпус текстов русского языка [5], British National Corpus [6], American National Corpus [7]. Однако для решения большого количества задач подобные корпуса не являются достаточными, а увеличение их размеров на порядок или на порядки требует существенных затрат. В связи с этим существует ряд решений, направленных на извлечение информации из неразмеченных корпусов текстов, когда лексическая информация приписывается словам «на лету» при помощи специальных программных модулей. Существенным недостатком подобного подхода является наличие омонимии – неоднозначности определения части речи слова, его нормальной формы или набора лексических параметров.

Выходом здесь является использование статистических методов. Так, например, может использоваться снятие омонимии с использованием уже упоминавшегося метода n-грамм, когда по соседним словам текущего слова определяется его наиболее вероятный набор параметров. Далее подобный текст может использоваться, например, для выделения коллокаций – неслучайных последовательностей слов [8-10]. Коллокации могут быть разбиты на несколько смысловых групп. К первой можно отнести «языковые маркеры», то есть составные предлоги, союзы и другие сервисные многословные конструкции, например, «таким образом», «и так далее», «так сказать». Вторая группа – это принятые в языке многословные термины: «исследование операций», «гуманитарная катастрофа», «лечащий врач». И, наконец, можно выделить «ситуационную лексику», то есть многословные конструкции, привязанные к конкретной ситуации или событию: «Красная стрела», «Белая лошадь», «олимпиада в Сочи». Через какое-то время после прохождения события подобные сочетания вымываются из активной лексики, но в ходе его обсуждения могут служить важными маркерами или смысловыми единицами.

Фокусом нашего исследования являются синтаксически связанные группы слов. Подобные конструкции используется в широком спектре задач. При переводе на данный язык словарь синтаксической сочетаемости слов необходим для того, чтобы переводчик мог проверить себя – правильно ли он связал слова или подобрал переводные

эквиваленты. В процессе обучения иностранному языку студент может проверить правильность составленных конструкций. Также информация о сочетаемости слов может использоваться и при автоматической обработке текстов. Так, например, она используется при проверке корректности получаемых в ходе синтаксического анализа деревьев синтаксических зависимостей. Кроме того, на основании информации о сочетаемости слов можно проводить собственно поверхностный синтаксический анализ. Возможны также и другие применения.

В предыдущих работах мы извлекали информацию из текстов на русском языке [11]. Для этого использовался поиск фрагментов текстов, отвечающих заданным шаблонам. Сами шаблоны с высокой долей вероятности гарантировали синтаксическую связность слов, а также определяли тип связи. Шаблоны применяли только к лексически однозначным группам слов, что гарантировало корректность применения шаблонов к тексту. Использование подобного подхода обеспечивает гораздо более высокую точность по сравнению с другими методами, например, BagofWords [12]. С другой стороны, накладываемые ограничения существенно сужают объем анализируемых текстов. Однако работа с неразмеченными текстами позволила набрать необходимый объем статистики.

В данной статье будет рассмотрен вопрос применения тех же методов к текстам на английском языке. Кроме того, мы исследовали влияние процесса снятия омонимии на качество получаемых результатов.

2. Постановка задачи. Как и в предыдущих работах, мы рассматривали неразмеченные тексты. Из текстов на английском языке извлекались конструкции следующего вида:

1. *<конструкция №1> ::= (начало предложения) <артикль>? <прилагательное>* <существительное><глагол>+*
2. *<конструкция №2> ::= <глагол>+ <предлог>? <артикль>? <прилагательное>* <существительное>*
3. *<конструкция №3> ::= <предлог|артикль><прилагательное>+ <существительное>*

Здесь знак вопроса обозначает необязательность нахождения данного слова, знак «*» означает ноль и более вхождений слова в текст, а знак «+» - одно и более вхождений.

Данные конструкции рассматривались с целью извлечения следующих синтаксических конструкций:

1. *<глагол>* $\rightarrow$ *<предлог>* $\rightarrow$ *<существительное>* ,
2. *<существительное>* $\rightarrow$ *<прилагательное>*,

где первое слово является главным по отношению к последнему. Наличие предлога в английском языке обозначает тип связи.

За счет неоднозначности слов текста или наличия в нем ошибок возможно выделение некорректных конструкций. В связи с этим необходимо, чтобы каждая подобная связь встретилась статистически значимое число раз. В связи с этим конструкции, встретившиеся менее заданного количества раз, отбрасываются как шум. Тем самым снижается полнота результатов, но повышается их точность.

Решение задачи можно представить как последовательность следующих этапов (рис. 1):

1. чтение файлов, содержащих исходные тексты;
2. разбиение текстов на предложения;
3. разбиение каждого предложения на токены;
4. поиск в каждом предложении конструкций;
5. выделение групп из полученных конструкций;
6. приведение слов в выделенных группах к нормальной форме;
7. сохранение полученных «нормализованных» групп в хранилище и расчет статистики встречаемости групп.

После выполнения разметки текста можно приступать к шагу выделения групп из конструкций и приведения каждого слова в группе к нормальной форме. Разбить конструкцию на группы можно при помощи грамматики, для чего придется приводить

полученные конструкции к списку токенов, либо решить эту же задачу при помощи реализации процедуры, которая, проходя в обратном порядке по списку токенов конструкции, ищет в прямом порядке совпадение частей речи рассмотренных токенов с частями речи каждой групп. Описанный алгоритм предпочтителен, так как имеет линейную временную сложность.

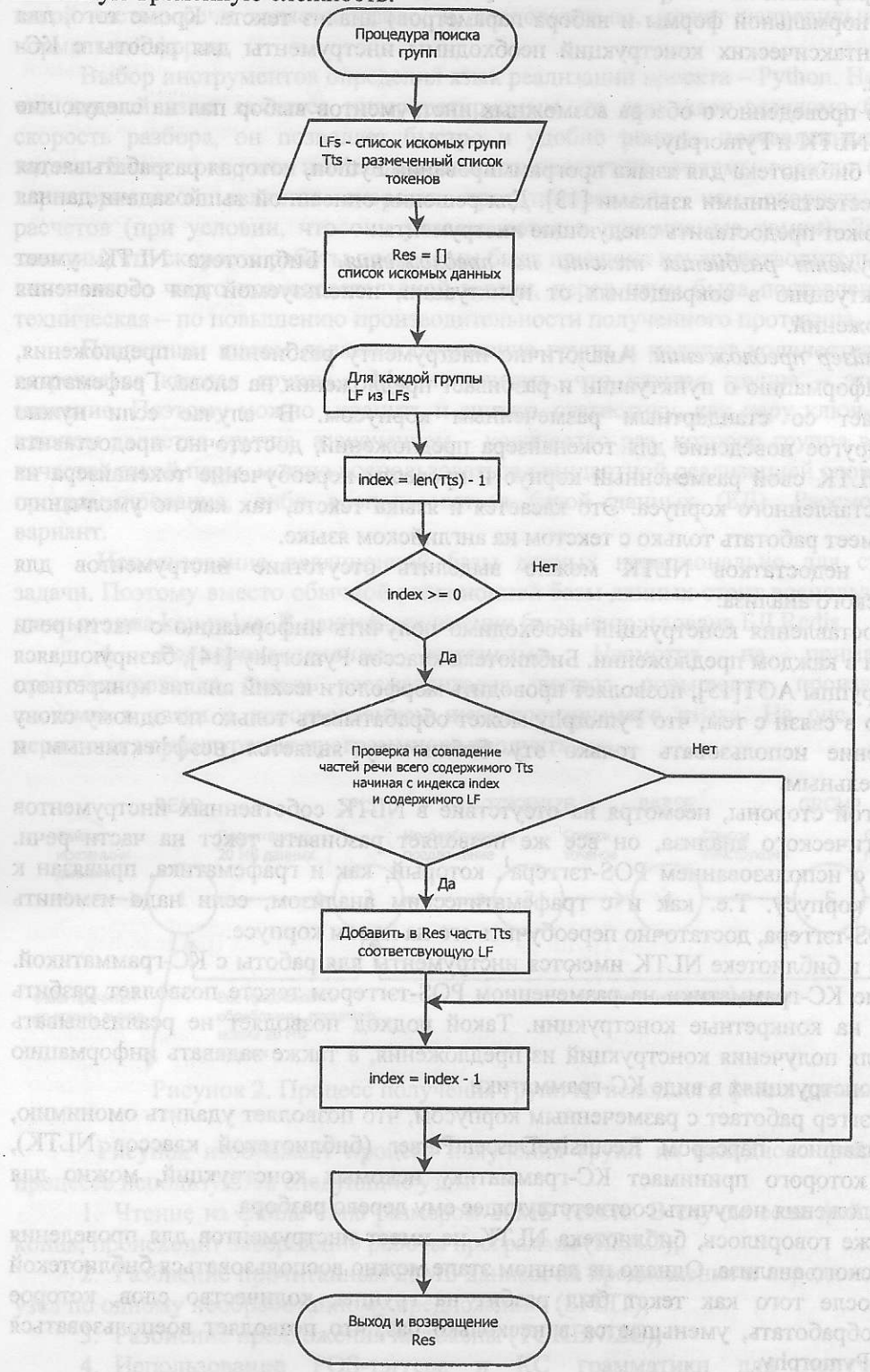

Рисунок 1. Алгоритм выделения групп из конструкций

3. Метод решения. Решение задачи начинается с прочтения текста из файла с исходными данными. Файл может иметь очень большой размер (так, например,

текстовый файл, содержащий патенты США занимает несколько гигабайт), в связи с чем чтение производится так называемыми «окнами» – фрагментами фиксированного размера.

Для анализа текста нам потребуются специальные программные инструменты, проводящие графематический (разбиение на предложения и слова) и морфологический (определение нормальной формы и набора параметров) анализ текста. Кроме того, для выделения синтаксических конструкций необходимы инструменты для работы с КС-грамматиками.

В ходе проведенного обзора возможных инструментов выбор пал на следующие два продукта: NLTK и Pymorphy.

NLTK – библиотека для языка программирования Python, которая разрабатывается для работы с естественными языками [13]. Для решения описанной выше задачи данная библиотека может предоставить следующие инструменты:

- *Инструмент разбиения текста на предложения.* Библиотека NLTK умеет отличать пунктуацию в сокращениях от пунктуации, используемой для обозначения границ предложений.
- *Токенайзер предложений.* Аналогично инструменту разбиения на предложения, использует информацию о пунктуации и разбивает предложения на слова. Графематика NLTK работает со стандартным размеченным корпусом. В случае если нужно реализовать другое поведение для токенайзера предложений, достаточно предоставить библиотеке NLTK свой размеченный корпус и провести переобучение токенайзера на основе предоставленного корпуса. Это касается и языка текста, так как по умолчанию инструмент умеет работать только с текстом на английском языке.

Среди недостатков NLTK можно выделить отсутствие инструментов для морфологического анализа.

Для составления конструкций необходимо получить информацию о части речи каждого слова в каждом предложении. Библиотека классов Pymorphy [14], базирующаяся на словарях группы AOT[15], позволяет проводить морфологический анализ конкретного слова. Однако в связи с тем, что Pymorphy может обрабатывать только по одному слову за раз, решение использовать только эту библиотеку является неэффективным и непроизводительным.

С другой стороны, несмотря на отсутствие в NLTK собственных инструментов для морфологического анализа, он все же позволяет разбивать текст на части речи. Делается это с использованием POS-тэггера[1], который, как и графематика, привязан к конкретному корпусу. Т.е. как и с графематическим анализом, если надо изменить поведение POS-тэггера, достаточно переобучить его на новом корпусе.

Также в библиотеке NLTK имеются инструменты для работы с КС-грамматикой. Использование КС-грамматики на размеченном POS-тэггером тексте позволяет разбить предложения на конкретные конструкции. Такой подход позволяет не реализовывать свой метод для получения конструкций из предложения, а также задавать информацию об искомых конструкциях в виде КС-грамматики.

POS-тэггер работает с размеченным корпусом, что позволяет удалить омонимию, а воспользовавшись парсером RecursiveDescentParser (библиотекой классов NLTK), конструктор которого принимает КС-грамматику искомых конструкций, можно для каждого предложения получить соответствующее ему дерево разбора.

Как уже говорилось, библиотека NLTK не имеет инструментов для проведения морфологического анализа. Однако на данном этапе можно воспользоваться библиотекой Pymorphy. После того как текст был разбит на группы, количество слов, которое необходимо обработать, уменьшается в несколько раз, что позволяет воспользоваться библиотекой Pymorphy.

---

[1]Part-Of-Speech-tagger — инструмент для разметки слов текста соответствующими им частями речи. Для разметки инструмент анализирует контекст каждого слова и ищет совпадения результатов анализа в уже размеченном наборе вспомогательных данных.

Минусом использования библиотеки Pymorphy может стать отсутствие возможности работы с контекстом слова. Это означает, что порой результаты могут быть не совсем корректными. К примеру, слово в сравнительной форме прилагательного и в форме наречия может выглядеть одинаково, но нормальные формы этого же слова для прилагательного и наречия могут различаться. Библиотека Pymorphy не будет знать, какой частью речи является слово, и, следовательно, может неправильно привести к нормальной форме.

Выбор инструментов определил язык реализации проекта – Python. Несмотря на то, что данный язык является интерпретируемым, то есть дает заведомо более низкую скорость разбора, он позволяет быстро и удобно решить поставленные перед нами задачи. Более того, для исследовательского прототипа системы гораздо более важной характеристикой является скорость прототипирования, чем скорость проводимых расчетов (при условии, что они укладываются в приемлемые рамки). Забегая вперед заметим, что скорость работы программы была признана неудовлетворительной, в связи с чем помимо чисто исследовательской задачи, перед нами была поставлена еще одна – техническая – по повышению производительности полученного прототипа.

Последним шагом является сохранение групп и подсчет количества раз, которое встречается каждая группа. Можно считать, что каждая группа – это уникальное значение. Поэтому можно хранить и считать статистику, как пару ключ-значение, где ключом является группа, а значением – количество раз, которое группа встречается. В качестве такой пары, можно воспользоваться стандартной реализацией словарей в языках программирования, либо воспользоваться базой данных (БД). Рассмотрим второй вариант.

Использование реляционной базы данных нерационально для столь простой задачи. Поэтому вместо обычной реляционной базы данных стоит воспользоваться базой данных типа key-value. В данной реализации была использована БД Redis.

4. Распараллеливание механизма. Несмотря на примат удобства прототипирования, нами рассматривался вопрос повышения производительности системы в связи с использованием интерпретируемого языка. На рис. 2 изображена первичная конфигурация программного продукта.

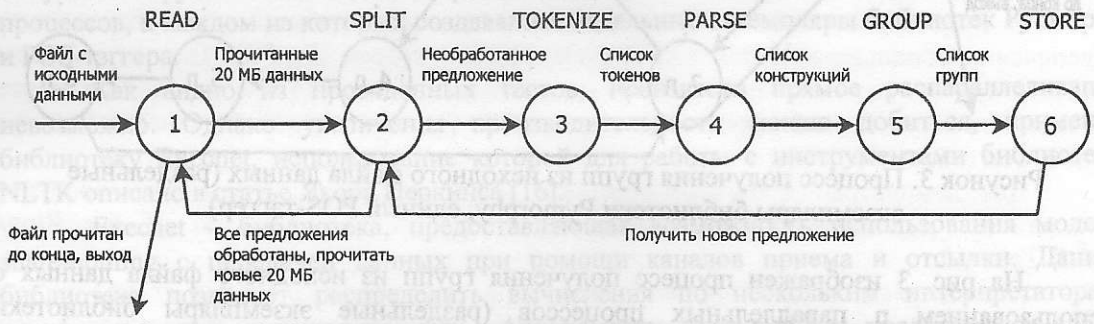

Рисунок 2. Процесс получения групп из исходного файла данных

Рисунок изображает процесс получения групп из исходного файла данных. В процессе используются следующие узлы:

1. Чтение из файла окна размером 20МБ текста. В случае если файл прочитан до конца, происходит завершение работы программы (READ);
2. Разбиение прочитанных 20МБ данных на предложения и передача в следующий узел по одному необработанному предложению (SPLIT);
3. Разбиение предложения на токены (TOKENIZE);
4. Использование POS-тэггера и КС грамматики для получения списка конструкций (PARSE);
5. Получение списка групп из списка полученных конструкций (GROUP);
6. Сохранение полученных конструкций в БД, подсчет статистики (STORE).

При хронометраже выполнения задачи на тексте объемом в 3МБ время работы программы составило 11мин. Следовательно, время обработки 900МБ тестового текста составит приблизительно 55 часов. Это не является оптимальным временем работы, поэтому была поставлена задачараспараллелить работу некоторых блоков программного продукта.

Так как для реализации используется язык программирования Python, то по причине наличия GIL[2] невозможно использовать параллельные потоки.

При мощности вычислительного ресурса в n процессоров для оптимальной работы можно запустить до n отдельных процессов. В соответствии с этим модифицируем часть кода с учетом того, что каждый процесс использует свой экземпляр библиотеки Pymorphy, но все процессы используют вызов одного и того же метода POS-тэггинга. Теперь текст объемом в 3МБ программа обрабатывает за 20 минут. Причиной является блокировка одним процессом, получившим доступ к модулю, других процессов, простаивающих всё это время.

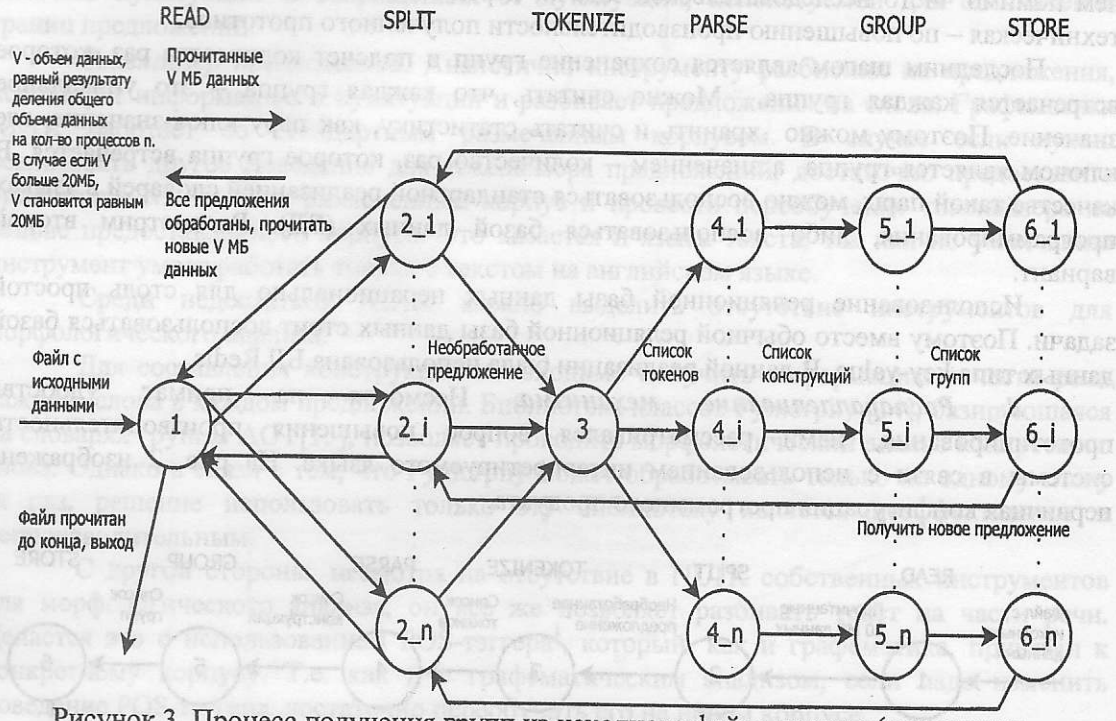

Рисунок 3. Процесс получения групп из исходного файла данных (раздельные экземпляры библиотеки Pymorphy, единый POS-тэггер)

На рис. 3 изображен процесс получения групп из исходного файла данных с использованием n параллельных процессов (раздельные экземпляры библиотеки Pymorphy, единый POS-тэггер).

Воспользовавшись профилировщиком и документацией по NLTK[13] можно прийти к выводу, что POS-тэггинг однопоточен и работает с диском, а работу с единственным экземпляром нельзя разбить на параллельные процессы. Основываясь на [13] и результатах работы профилировщика, перепишем код так, что каждый процесс использует свой Pymorphy и метод POS-тэггинга, который работает с дампом корпуса, хранящимся в памяти. У каждого процесса свой дамп. Программа обработала текста объемом в 3МБ за 19 минут. Это всё еще хуже, чем решение без использования распараллеливания.

[2]GlobalInterpreterLock – используемая в языке программирования Python глобальная блокировка интерпретатора, не позволяющая выполнять больше одного потока в единицу времени.

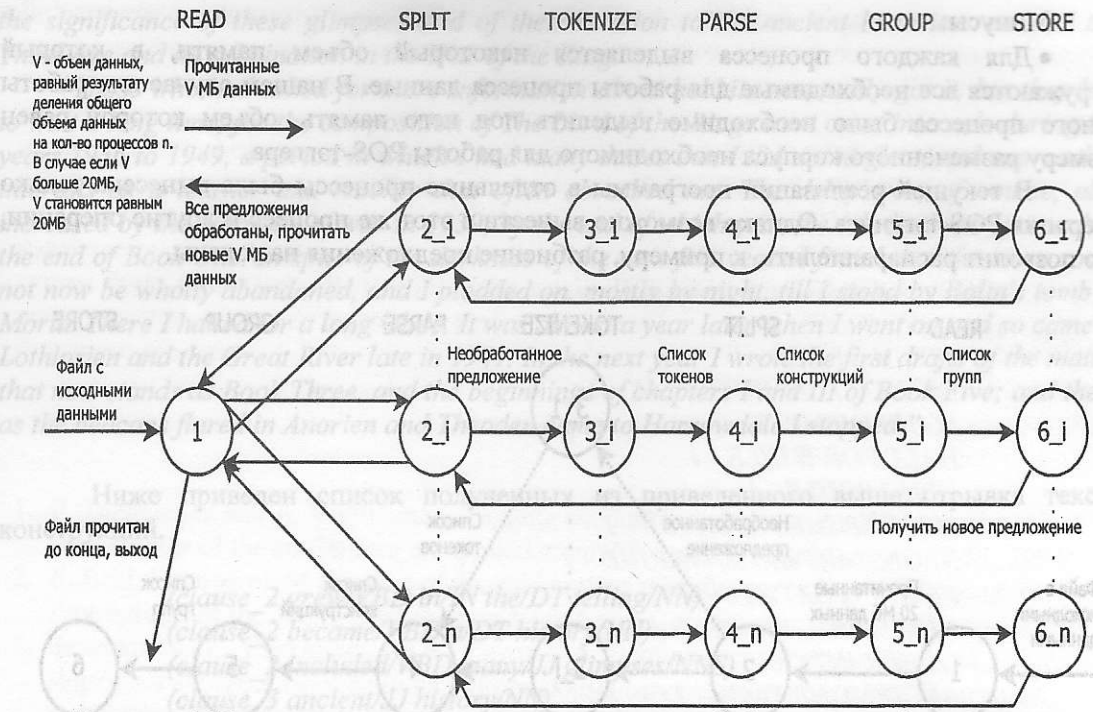

Рисунок 4. Процесс получения групп из исходного файла данных (раздельные экземпляры библиотеки Pymorphy, разделенный вызов POS-тэггера)

В связи с этим было принято решение использовать технологию MIMD – Multiple Instructions Multiple Data. Дело в том, что входной файл может рассматриваться как независимый набор окон, каждое из которых может обрабатываться параллельно. Для этого необходимо обрабатывать каждое текстовое окно полностью независимо всем алгоритмом, представленным на рис. 2. Полученные в итоге данные складываются в базу данных, являющуюся таким образом аггрегатором. На рис. 4 изображен процесс получения групп из исходного файла данных с использованием n параллельных процессов, в каждом из которых создавались отдельные экземпляры библиотек Pymorphy и POS-тэггера.

Как видно из проведенных тестов, произвести прямое распараллеливание невозможно. Однако увеличения производительности можно добиться, применив библиотеку Execnet, использование которой для работы с инструментами библиотеки NLTK описано в статье Якоба Перкинса [16].

Execnet – библиотека, предоставляющая возможность использования модели share-nothing с передачей данных при помощи каналов приема и отсылки. Данная библиотека позволяет распределить вычисления по нескольким интерпретаторам, платформам или участникам сети. Архитектура share-nothing представляет собой распределенную вычислительную архитектуру, в которой каждый узел является независимым и самодостаточным, а в системе не существует единой конкурирующей точки. В частности, ни один из узлов не имеет общего пространства памяти или дискового хранилища.

Теперь время обработки 3МБ текста уменьшилось до 6 минут. Время обработки 900МБ – 24 часа. Практика показывает, что подобная скорость является достаточной даже для применения в реальных системах, например, мониторинга новостной ленты и/или сообщений пользователей.

На рис. 5 изображен процесс получения групп из исходного файла данных с использованием n параллельных каналов Execnet.

Плюсы использования данного подхода:

- Возможность запустить количество параллельных процессов, ограниченное лишь ресурсами системы.

Минусы:

• Для каждого процесса выделяется некоторый объем памяти, в который загружаются все необходимые для работы процесса данные. В нашем случае для работы одного процесса было необходимо выделить под него память, объем которой равен размеру размеченного корпуса необходимого для работы POS-тэггера.

В текущей реализации программы в отдельные процессы была вынесена только операция POS-тэггинга. Однако возможно вынести в этот же процесс и другие операции, что позволит распараллелить, к примеру, разбиение предложения на токены.

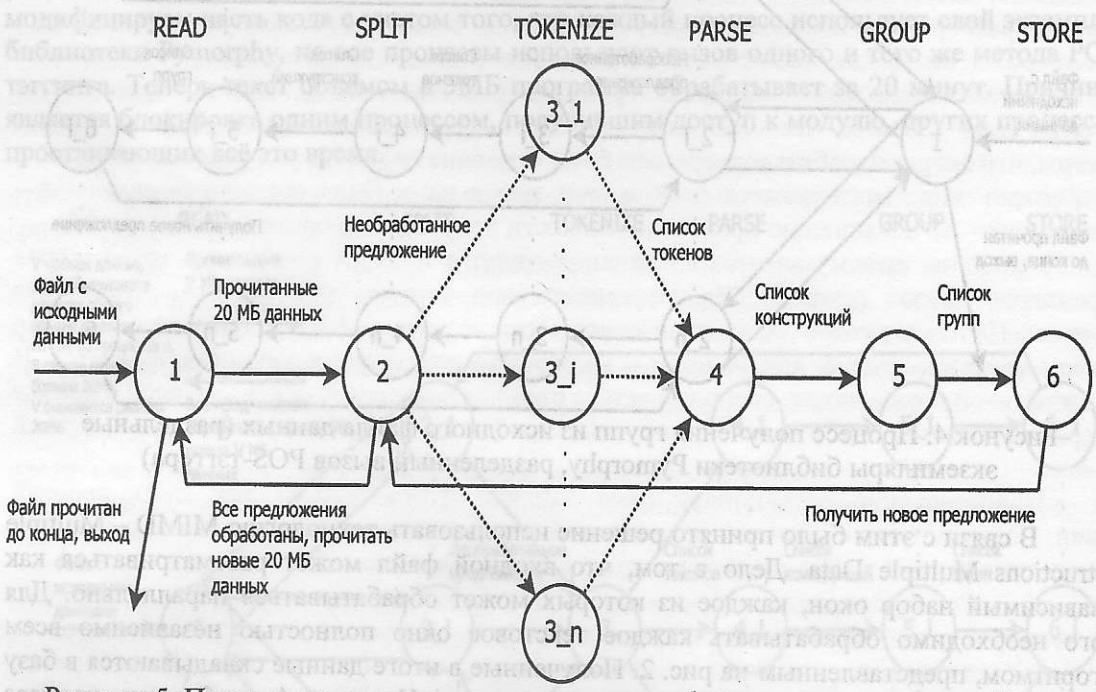

Рисунок 5. Процесс получения групп из исходного файла с использованием Execnet

5. Результаты экспериментов. Для проверки разработанного программного обеспечения использовался текст книги «Властелин колец» Дж.Р.Р. Толкиена (объем 3МБ). Получение синтаксически связанных конструкций должно проводиться на тексте гораздо большего объема, в связи с этим использовался дамп новостей компании Рейтерс за 2007 год (объем 900МБ).

Для демонстрации принципа работы разработанного программного продукта предлагается рассмотреть следующий отрывок из книги The Lord of the Rings (J.R.R.Tolkien):

"This tale grew in the telling, until it became a history of the Great War of the Ring and included many glimpses of the yet more ancient history that preceded it. It was begun soon after The Hobbit was written and before its publication in 1937; but I did not go on with this sequel, for I wished first to complete and set in order the mythology and legends of the Elder Days, which had then been taking shape for some years. I desired to do this for my own satisfaction, and I had little hope that other people would be interested in this work, especially since it was primarily linguistic in inspiration and was begun in order to provide the necessary background of 'history' for Elvish tongues. When those whose advice and opinion I sought corrected little hope to no hope, I went back to the sequel, encouraged by requests from readers for more information concerning hobbits and their adventures. But the story was drawn irresistibly towards the older world, and became an account, as it were, of its end and passing away before its beginning and middle had been told. The process had begun in the writing of The Hobbit, in which there were already some references to the older matter: Elrond, Gondolin, the High-elves, and the orcs, as well as glimpses that had arisen unbidden of things higher or deeper or darker than its surface: Durin, Moria, Gandalf, the Necromancer, the Ring. The discovery of

the significance of these glimpses and of their relation to the ancient histories revealed the Third Age and its culmination in the War of the Ring.

Those who had asked for more information about hobbits eventually got it, but they had to wait a long time; for the composition of The Lord of the Rings went on at intervals during the years 1936 to 1949, a period in which I had many duties that I did not neglect, and many other interests as a learner and teacher that often absorbed me. The delay was, of course, also increased by the outbreak of war in 1939, by the end of which year the tale had not yet reached the end of Book One. In spite of the darkness of the next five years I found that the story could not now be wholly abandoned, and I plodded on, mostly by night, till I stood by Balin's tomb in Moria. There I halted for a long while. It was almost a year later when I went on and so came to Lothlorien and the Great River late in 1941. In the next year I wrote the first drafts of the matter that now stands as Book Three, and the beginnings of chapters I and III of Book Five; and there as the beacons flared in Anorien and Theoden came to Harrowdale I stopped."

Ниже приведен список полученных из приведенного выше отрывка текста конструкций.

(clause_2 grew/VBD in/IN the/DT telling/NN)
(clause_2 became/VBD a/DT history/NN)
(clause_2 included/VBD many/JJ glimpses/NNS)
(clause_3 ancient/JJ history/NN)
(clause_2 set/VB in/IN order/NN)
(clause_2been/VBN taking/VBG shape/NN)
(clause_3 own/JJ satisfaction/NN)
(clause_2 had/VBD little/JJ hope/NN)
(clause_3 that/IN other/JJ people/NNS)
(clause_2 was/VBD begun/VBN in/IN order/NN)
(clause_2 provide/VB the/DT necessary/JJ background/NN)
(clause_2 sought/VBD corrected/VBN little/JJ hope/NN)
(clause_2 encouraged/VBD by/IN requests/NNS)
(clause_3 for/IN more/JJR information/NN)
(clause_2 concerning/VBG hobbits/NNS)
(clause_2 became/VBD an/DT account/NN)
(clause_1 The/DT process/NN had/VBD begun/VBN)
(clause_2 had/VBD arisen/VBN unbidden/VBN of/IN things/NNS)
(clause_3 the/DT ancient/JJ histories/NNS)
(clause_2 revealed/VBD the/DT Third/NNP Age/NNP)
(clause_2 had/VBD asked/VBN for/IN more/JJR information/NN)
(clause_2 wait/VB a/DT long/JJ time/NN)
(clause_2 had/VBD many/JJ duties/NNS)
(clause_3 other/JJ interests/NNS)
(clause_1 The/DT delay/NN was/VBD)
(clause_2 increased/VBD by/IN the/DT outbreak/NN)
(clause_2 reached/VBN the/DT end/NN)
(clause_2 found/VBD that/IN the/DT story/NN)
(clause_2 stood/VBD by/IN Balin/NNP)
(clause_2 halted/VBD for/IN a/DT long/JJ while/NN)
(clause_3 In/IN the/DT next/JJ year/NN)
(clause_2 wrote/VBD the/DT first/JJ drafts/NNS)
(clause_2 stands/VBZ as/IN Book/NNP Three/NNP)

Здесь *clause_i* – это обозначение, соответствующее номеру конструкции, а *VBD, IN, DT,NN*и т.д – описывают название части речи. Полный перечень обозначений частей речи можно найти на сайте документации к NLTK[13].

Результатом работы программного продукта является статистика, которая для данного отрывка и полученных конструкций будет выглядеть следующим образом (из-за использования БД порядок групп не соответствует порядку конструкций).

STAND BY BALIN - 1
MANY INFORMATION - 2
MANY DUTY - 1
OWN SATISFACTION - 1
ENCOURAGE BY REQUEST - 1
LONG TIME - 1
SET IN ORDER - 1
INCREASE BY OUTBREAK - 1
MANY GLIMPSE - 1
HALT FOR WHILE - 1
LONG WHILE - 1
ASK FOR INFORMATION - 1
NECESSARY BACKGROUND - 1
FOUND THAT STORY - 1
NEXT YEAR - 1
OTHER INTEREST - 1
ANCIENT HISTORY - 2
UNBIDDEN OF THING - 1
BEGIN IN ORDER - 1
OTHER PERSON - 1
FIRST DRAFT - 1
STAND AS BOOK - 1
LITTLE HOPE – 2

Результат работы представлен в виде словаря, где ключ – это группа, а значение – количество раз, которые данная группа встречается в конструкциях.

В ходе решения задачи был обработан текст дампа новостей компании «Рейтерс» за 2007 год, состоящий из 154 млн. токенов. Текст новостей разбивался на предложения, которые в свою очередь разбивались на токены. Из набора токенов одного предложения выделялись конструкции, из которых в свою очередь выделялись группы. В процессе выделения конструкций было задействовано 40 млн. токенов. Из полученных конструкций было выделено около 1,2 млн. уникальных групп и посчитана статистика встречаемости каждой группы в дампе новостей.

6. Выводы. В результате выполненной работы был создан программный продукт, позволяющий извлекать из английских текстов такие синтаксические конструкции, как «глагол+предлог+существительное» и «существительное+прилагательное». В качестве входных данных программный продукт получает текстовый файл, который необходимо проанализировать, список правил КС-грамматики для разбиения текста на конструкции, список групп и соответствующие им списки частей речи, характеризующие группы. Анализ показал, что для получения удовлетворительной скорости работы продукта необходимо использовать распараллеливание потоков команд по технологии MIMD.

Результатом работы программного продукта является текстовый файл со списком выделенных конструкций и текстовый файл со списком выделенных из конструкций групп – словарь сочетаемости слов для английского языка. В ходе проведения экспериментов над дампом новостей компании «Рейтерс» за 2007 год (900МБ текста) было получено 1,2 млн. различных групп. Заметим, что данный результат сопоставим с результатами, получаемыми для русского языка на коллекции того же объема [1]. Напомним, что для русских текстов брались лишь морфологически однозначные слова. Проведенные исследования показали, что английский язык обладает омонимией совершенно иного типа: количество частеречных омонимов в нем в несколько раз

больше, чем в русском языке. В связи с этим применение разработанного метода без предварительного снятия омонимии не позволяет получить сколь-либо значимых результатов. Качество полученных результатов также сопоставимо с результатами, полученными для русского языка. В связи с этим можно сделать вывод о том, что использование методов автоматического снятия омонимии позволяет сохранить качество работы, существенно увеличивая количество выделенных конструкций.

Таким образом, можно сделать вывод, что предлагаемый метод пригоден для извлечения синтаксически связанных конструкций из текстов на английском языке и может быть в дальнейшем использован для формирования словаря глагольной и другой сочетаемости слов английского языка на основе анализа неразмеченных корпусов большего объема.

Литература

1. P. Tapanainen and A. Voutilainen. Tagging accurately – don't guess if you know. In Proceedings of the conference on applied natural language processing, pp. 47-52, 1994/
2. E. Brill. Unsupervised learning of disambiguation rules for part of speech tagging. In Proceedings of the Third Workshop on Very Large Corpora, pp. 1–13, 1995.
3. Зеленков Ю.Г., Сегалович Ю.А., Титов В.А. Вероятностная модель снятия морфологической омонимии на основе нормализующих подстановок и позиций соседних слов // Компьютерная лингвистика и интеллектуальные технологии. Труды международного семинара Диалог'2005., сс/ 188-197, 2005.
4. Протасов С.В. Вывод и оценка параметров дальнодействующей триграммной модели языка // Компьютерная лингвистика и интеллектуальные технологии. Труды международного семинара Диалог'2008., сс. 443-449, 2008.
5. Национальный корпус русского языка, http://www.ruscorpora.ru
6. Национальный корпус Великобритании, http://www.natcorp.ox.ac.uk/
7. Национальный корпус США, http://americannationalcorpus.org/
8. Хохлова М.В. Экспериментальная проверка методов выделения коллокаций // Сб. статей «Инструментарий русистики: корпусные подходы». – Хельсинки, 2008. сс. 343-357
9. Church K., Hanks, P. Word association norms, mutual information, and lexicography, *Computational Linguistics*, 1990, 16(1), 22–29.
10. Stubbs, M. Collocations and semantic profiles: On the cause of the trouble with quantitative studies, 1995. *Functions of Language*, 1, pp. 23-55.
11. Клышинский Э.С., Кочеткова Н.А., Литвинов М.И., Максимов В.Ю. Метод разрешения частеречной омонимии на основе применения корпуса синтаксической сочетаемости слов в русском языке // Научно-техническая информация. Серия 2: Информационные системы и процессы. №1 2011 г., сс. 31-35
12. Suzan Verberne, Lou Boves, Nelleke Oostdijk, Peter-Arno Coppen What is not in the bag of words for why-qa? // Computational Linguistics Volume 36 Issue 2, June 2010 pp. 229-245.
13. Документация к NLTK: http://www.nltk.org/documentation
14. Pymorphy v0.5.5 documentation: http://packages.python.org/pymorphy/usage/base.html
15. Автоматическая обработка текста: http://aot.ru/
16. Jacob Perkins, Using Execnet for Parallel and Distributed Processing with NLTK: http://www.packtpub.com/article/using-execnet-parallel-and-distributed-processing-nltk

UOT 004.272.26:004.912

T.P. Qurbanov, E.S. Klışınski. İngilis dilində xəbər mətnləri üçün feli idarəetmə lüğətinin tərtibinin paralel alqoritmi

Məqalədə sintaksis əlaqəli konstruksiyalardan sözlər qrupunun ayrılması məsələsinin həlli təsvir olunur. Konstruksiyalar öz növbəsində ilkin verilənlərdən ibarət olan edən mətn faylından

ayrılır. Məsələnin həlli NLTK və Pymorphy kitabxanalarından istifadə etməklə Python proqramlaşdırma dilinin köməyilə yerinə yetirilir.
Açar sözlər: təhlil, analiz, mətn, konstruksiyalar, qrammatika, semantika, emal, NLP

T.P. Gurbanov, E.S. Klyshinsky. Parallel verbal government dictionary building algorithm for news texts in English

This article describes solution to the problem of word group allocation from syntactically connected constructions. Constructions are extracted from text data. The solution is written in Python programming language with use of NLTK and Pymorphy libraries.
Key words: parsing, analyzing, text, word groups, grammar, semantics, processing, NLP

МИЭМ,
МГТУ им.Н.Э.Баумана

Представлено 09.03.12