УДК 004.89

А.К. Розанов, А.В. Пруцков

СПОСОБЫ ПОВЫШЕНИЯ СКОРОСТИ РАБОТЫ АЛГОРИТМА МОРФОЛОГИЧЕСКОГО АНАЛИЗА ФОРМ СЛОВ ЕСТЕСТВЕННЫХ ЯЗЫКОВ

Приводится оценка скорости существующего алгоритма анализа форм слов естественных языков, предлагаются два способа к ускорению процесса анализа и описываются их преимущества и недостатки.

Первый способ включает использование особенностей базиса элементарных операций, применяемого для описания правил формообразования. Второй подразумевает хранение всех известных словоформ в словаре.

Показан выигрыш в скорости (пятикратный для первого способа и более двух порядков – для второго), приводятся причины, обусловливающие наблюдаемый прирост скорости, приводятся некоторые статистические данные о набранном в ходе экспериментов словаре формообразования русского языка.

Ключевые слова: *преобразования строк, анализ форм слов, автоматическая обработка текста, формообразование.*

Введение. Задача определения форм слов естественного языка (получения грамматической информации для каждого из слов текста) и связанная с ней естественным образом обратная задача (генерация слов, обладающих требуемой грамматической информацией) – это задачи, возникающие в целом ряде практических вопросов языкознания [1]. Примерами областей применения генераторов и анализаторов форм слов могут служить такие области знаний, как извлечение знаний (англ. data mining), статистическая обработка текста, построение экспертных систем, поддерживающих общение на естественном языке, поисковых систем, интеллектуальных электронных справочников и словарей [2], машинный перевод [3] и другие задачи.

Целью работы является ускорение существующего алгоритма анализа форм слов естественных языков [4, 5], получение численных оценок скорости анализа слов и установление отсутствия деградации скорости с увеличением числа слов в анализируемых текстах.

В работе [7] предложены два способа ускорения анализа форм слов: с организацией правил преобразования слов в особую структуру данных («встречные префиксные деревья») и с использованием полного банка всех известных словоформ.

Условия проведения экспериментов. Все измерения скорости проводились с использованием одного и того же **тестового компьютера** следующей конфигурации: процессор Intel Core i7-4800MQ (2,7 ГГц), Windows 7 x64, RAM 16 Гбайт; все данные перед проведением замеров предварительно целиком размещались в оперативной памяти во избежание плохо предсказуемых задержек, обусловленных обращениями к жёстким дискам. Все алгоритмы были реализованы на языке C# с использованием одной и той же модели представления знаний о формообразовании [6].

Все тесты скорости проводились на случайных выборках слов из русскоязычных текстов (роман А. Азимова «Сами боги», учебник «Введение в философию»). Количество слов, подаваемых на вход процедуры анализа, последовательно увеличивалось, эксперимент повторялся несколько раз и значения усреднялись, чтобы снизить влияние других задач, выполняемых на тестовом компьютере. Линии на графиках отмечают усреднённые значения скорости по всем экспериментам.

Время, потраченное на разбиение текста на слова, не учитывалось в тестах. Все тестовые наборы представляли собой последовательности (списки) слов русского языка.

Тестовый словарь формообразования содержал более 6000 начальных форм слов русского языка и занимал в оперативной памяти около 16 Мб. Распределение частей речи использован-

ного словаря среди этих начальных форм приведено на рисунке 1.

Рисунок 1 – Части речи в тестовом словаре

Разнообразие правил формообразования русского языка иллюстрирует рисунок 2 (число семейств по частям речи). Слова относятся к разным семействам, если их парадигмы порождаются разными наборами правил формообразования.

Рисунок 2 – Распределение семейств в тестовом словаре

Представление правил преобразования.

В [4, 5] преобразования, необходимые для получения некоторой формы слова из её начальной формы, описываются в виде **цепочки преобразований строк** – конечной последовательности элементарных операций.

В [7] понятия цепочки, операции и базиса определены строго:

Элементарным обратимым преобразованием строки (элементарной операцией) любое правило преобразования, отвечающее условиям **выполнимости**, **однозначности** и **существования обратного преобразования**:

– **выполнимость и однозначность**: любая пара, состоящая из операции P и строки S, однозначно характеризуется одним из двух высказываний:

а) операция P невыполнима над строкой S (например, операция «удалить окончание *-и*» неприменима к строке *ваза*);

б) результат выполнения операции P над S есть строка S' (например, результатом применения операции «добавить окончание *-а* к строке *слон* является строка *слона*).

– **обратимость**: для любой операции P можно указать такую операцию P', что для любой строки S, над которой операция P выполнима (с результатом S'), результатом выполнения P' над S' является строка S.

Для краткости в дальнейшем будем называть элементарные обратимые преобразования строк «операциями».

Выбор **базиса элементарных операций** (набора видов операций, используемого для записи правил формообразования) диктуется конкретным языком, анализатор форм слов которого требуется построить. Для большого числа языков (в том числе русского) целесообразно использовать постфиксный базис, включающий операции отделения и присоединения подстроки справа. Особо отметим, что формообразование (получение форм одного и того же слова) следует отличать от словообразования (получение новых слов с помощью, например, приставок и суффиксов) [15]. Русские префиксы и интерфиксы представляют собой исключительно словообразовательные морфемы (§ 183 в [16]).

Цепочкой преобразований $C = (P_1; P_2; \ldots; P_n)$ будем называть (в соответствии с [7]) конечную последовательность элементарных преобразований строк (в выбранном базисе). Нулевую последовательность преобразований будем называть нулевой цепочкой.

Конкретные цепочки преобразований будем обозначать кратко, например, «*+и*» для цепочки, состоящей из одного преобразования «присоединить справа подстроку *и*», и «*–а+и*» для цепочки «заменить окончание *а* на окончание *и*».

Как показано в [7], для любой цепочки в постфиксном базисе с непустой областью применимости можно конструктивно и за полиномиальное время указать эквивалентную ей редуцированную цепочку, причём эта цепочка не будет содержать в себе более двух элементарных операций.

Для доказательства этого утверждения в работе [7] приведён алгоритм последовательного упрощения цепочек преобразований строк в постфиксном базисе («алгоритм редукции цепочек»).

Под **грамматической информацией** (в рамках данной модели) будем понимать полную совокупность грамматических признаков, определяющую (в большинстве случаев, но необязательно однозначно) форму слова (её запись с использованием алфавита языка). Так, грамматическая информация, соответствующая слову *белыми*, есть «множественное число, творительный падеж». В рамках модели, рассматриваемой в [4], эта информация не подвергается структурированию.

Понятие **начальной формы слова** будем использовать в его обычном значении [8] [например, для существительных русского языка начальной формой будем считать именительный падеж единственного числа или именительный падеж множественного числа, если слово (например, *ножницы*) не имеет единственного числа].

Словоформой (или **формой слова**) будем считать слово в любой, а не только начальной форме (слова *саней*, *санями* и *саням* являются словоформами, соответствующими начальной форме *сани*).

Под анализом формы слова будем понимать процесс получения грамматической информации в виде совокупности значений грамматических категорий [7]. В случае омонимии результатом анализа будем считать всё множество вариантов таких совокупностей.

1. **Изначальный вариант алгоритма анализа** форм слов с использованием цепочек преобразований [4, 5] подразумевал перебор всех известных правил обратного преобразования, выбор совместимых с анализируемым словом (в случае постфиксного базиса правила, не отсоединяющие постфикс, совместимы с любыми словами, а правила, отсоединяющие постфикс – только со словами, имеющими этот постфикс) и отсев тех, которые дали отсутствующие начальные формы и тех, к результатам применения которых неприменимы соответствующие прямые преобразования (последняя проверка отсекает случаи, когда обратная цепочка даёт известную основу, но прямая цепочка к ней неприменима: например, обратная цепочка «*-и*», определяющая множественное число слова «*люк*», получит из слова «сани» также известное слово «*сан*», однако прямая цепочка «*+и*» неприменима к этому слову, то есть «*сани*» – не форма слова «*сан*»).

Скорость анализа методом в его изначальном виде (включена оптимизация, группирующая цепочки по тождественным наборам операций) отражена на рисунке 3.

На рисунке 3 и следующих за ним аналогичных рисунках множества столбцов соответствуют различным наборам тестовых слов (для каждого контрольного объёма входного текста тест генерировал N случайно выбранных из большого корпуса слов), а сплошная линия показывает усреднённые значения скорости для каждого из контрольных объёмов.

Алгоритм в изначальном варианте не требует дополнительных расходов оперативной памяти (помимо уже упомянутых выше 16 мегабайт на полное структурированное представление в памяти словаря).

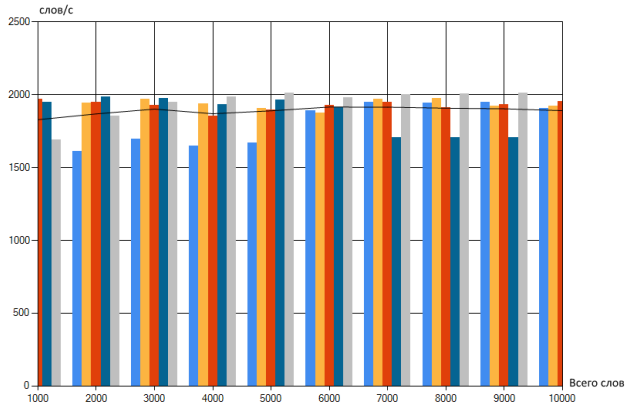

Рисунок 3 – Скорость анализа исходным методом

Помимо этого, следует отметить, что изначальный алгоритм не использует особенностей базиса преобразований строк, и будет без модификаций работать и для словарей, описывающих языки со смешанным формообразованием (в которых присутствуют и префиксные, и постфиксные правила), а также для словарей формообразования языков, в которых при формообразовании встречается редупликация (так, в малайском языке *orang* – человек, *orangorang* – люди).

Использование особенностей формообразования конкретного языка позволяет строить более быстрые, но и более узко специализированные алгоритмы анализа форм слов.

2. **Введение «встречных» префиксных деревьев** для хранения правил преобразования (алгоритм предложен А.К. Розановым и подробно описан в [7, 9]) позволяет улучшить скорость анализа форм слов за счёт исключения из перебора большей части цепочек, несовместимых с анализируемым словом.

Смысл подхода, предложенного автором, заключается в использовании особой структуры данных, представляющей собой композицию префиксных деревьев (сами префиксные деревья (англ. *trie*) описаны Эдвардом Фредкиным в [10]).

Этот подход к ускорению анализа хорошо иллюстрирует рисунок 4.

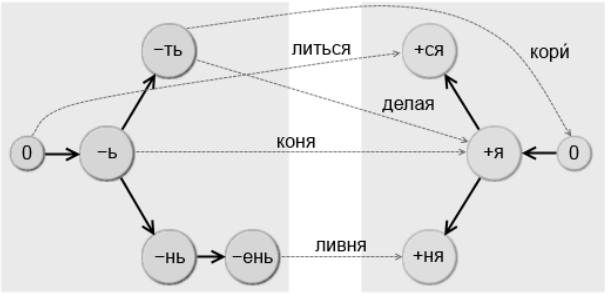

Рисунок 4 – «Встречные» префиксные деревья

«Левый» и «правый» графы рисунка 4 описывают все возможные отсоединяемые и

присоединяемые постфиксы (окончания), и каждое правило преобразования представляется в виде дуги, соединяющей левый граф с правым.

В этом случае алгоритм анализа подразумевает поиск по значительно меньшему числу цепочек, поскольку поиск при использовании этой структуры данных осуществляется только среди цепочек, представленных дугами, входящими в одну из вершин цепи правого дерева, соответствующей последовательности букв, на которую заканчивается анализируемое слово [7].

Скорость анализа при использовании «встречных» префиксных деревьев оказывается существенно выше скорости работы исходного метода даже при тестировании на наборах слов без повторений (рисунок 5).

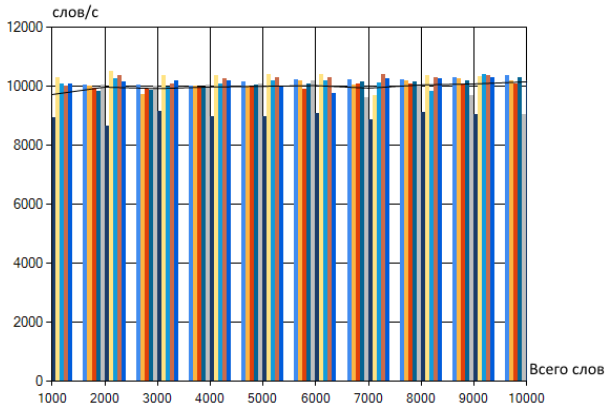

Рисунок 5 – Скорость анализа методом встречных префиксных деревьев без повторяющихся слов

На рисунке 5 приведена скорость анализа для наборов, не содержащих двух одинаковых слов (хотя не исключается содержание нескольких словоформ, относящихся к одной и той же начальной форме). График демонстрирует, что в этом случае скорость анализа не зависит от числа слов в анализируемом тексте и составляет около 10 тысяч слов в секунду.

В реальных текстах слова часто повторяются, поэтому реализации алгоритма анализа, использующие хэширование уже рассмотренных вариантов, обеспечивают прирост скорости по мере увеличения числа слов в анализируемом тексте (рисунок 6).

При этом следует отметить, что дополнительные затраты памяти на построение «встречных» деревьев оказались сравнительно малы: для использованного в тестах словаря в 6000 начальных форм слов дополнительно было израсходовано лишь чуть менее 3 мегабайт (в дополнение к 16 мегабайт памяти, хранящим весь словарь в виде сложной иерархии объектов).

Следует также заметить, что, хотя структуры данных, подобные «левому» и «правому» графам рисунка 4, называют префиксными деревьями, в контексте решаемой задачи и в случае постфиксного базиса операций эти деревья строятся по постфиксам, отделяемым и присоединяемым представляемыми правилами формообразования.

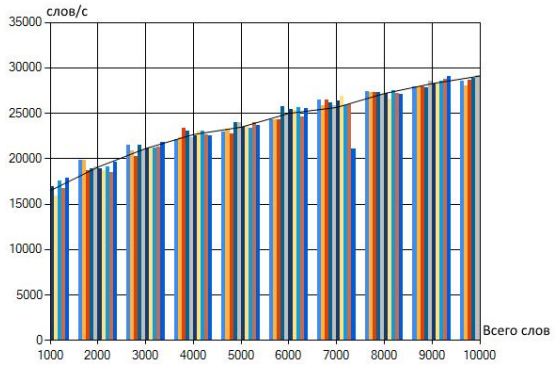

Рисунок 6 – Скорость анализа методом встречных префиксных деревьев при наличии в тексте повторяющихся слов

3. **Анализ с применением банка всех известных форм слов** (идея описана, например, в [11]) является наиболее быстрым решением ввиду исключительной простоты алгоритма: все затраты на поиск омонимов, анализ структуры словаря и построение списков вариантов грамматической информации выполняются при построении банка всех форм (рисунок 7).

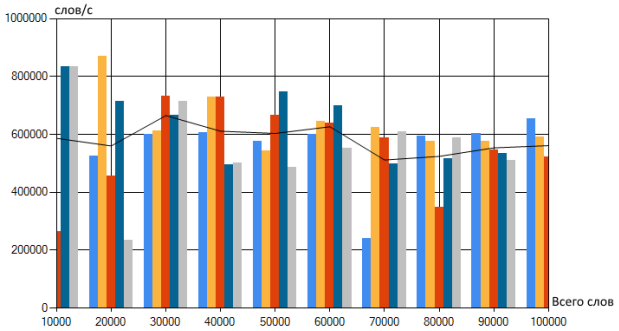

Рисунок 7 – Скорость анализа с применением банка всех известных форм слов

Рисунок 7 отражает скорость анализа в случае, когда каждое слово входного текста известно системе.

Практически все современные высокоуровневые языки программирования уже содержат реализацию ассоциативного массива с быстрой выборкой элементов по ключу [даже без хеширования выборка из упорядоченного миллиона элементов требует не более $\log_2(1\ 000\ 000)$ сравнений (то есть, менее 20), а в случае хеширования сравниваться будут значения хэш-функции], – настолько быстрой, что официаль-

ная документация предписывает считать операцию выборки имеющей сложность O(1) [12, 13, 14] в среднем (при низком числе коллизий). По этой причине наличие неизвестных системе слов не влияет на скорость анализа негативно.

Напротив, при анализе случайной выборки слов из крупных текстов (часть которых была неизвестна системе), тесты показывали прирост скорости (вплоть до миллиона слов в секунду).

Это объясняется тем, что процедура поиска слова в банке не усложняется для отсутствующих в нём слов, однако дополнительные операции, необходимые для генерации структуры объектов, описывающей все варианты грамматической информации анализируемого слова, сводятся к минимуму, когда это слово системе не известно.

Недостатком такого подхода является сравнительно большой расход оперативной памяти: так, для словаря, содержащего чуть более 6000 начальных форм, банк всех форм слов занял порядка 250 мегабайт оперативной памяти (чуть более 200 тысяч форм слов).

Это обстоятельство затрудняет применение такого подхода в устройствах с ограниченными объёмами доступной оперативной памяти (мобильные телефоны, планшеты и пр.), однако не является непреодолимым препятствием на серверных системах, где объёмы доступной оперативной памяти могут составлять десятки гигабайт.

Сравнение с существующими аналогами. В настоящее время существует большое число различных систем анализа форм слов как словарных, так и без словаря, скорость анализа в которых варьируется от тысяч до сотен тысяч слов в секунду. Ниже приведены некоторые из доступных бесплатно распространяемых программных средств.

– Библиотека pymorphy2 [17] позволяет выполнять анализ форм слов со скоростью 30-50 тыс. слов в секунду; недостатком подхода является отсутствие возможности генерации слов.

– Программа Linguistica [18] осуществляет анализ форм слов без словаря (для анализа используется большой корпус текстов на данном языке), скорость анализа достигает 100 тыс. слов в секунду, однако, учитывая метод анализа, генерация слов в этом программном комплексе также невозможна. Другим недостатком является то, что результаты анализа, во-первых, определяют лишь основы, но не грамматическую информацию, а во-вторых, получены не точно, а с некоторой вероятностью.

– Утилита командной строки Mystem [19] работает аналогично pymorphy2, показывает несколько большую скорость (около 80 тысяч слов в секунду), и, помимо прочего, результаты представляет в виде текстового файла, то есть требует от использующего её программиста определения собственной системы для представления грамматической информации.

– Метод, предложенный А.В. Пруцковым, является универсальным, допускает как анализ, так и синтез, однако имеет небольшую скорость (2-3 тысячи слов в секунду [4]).

Предлагаемый в данной статье метод анализа форм слов, во-первых, обеспечивает сравнимую с существующими аналогами (уже для текста из 100 тысяч слов скорость анализа достигает 100 тысяч слов в секунду без применения полного банка словоформ) скорость анализа, во-вторых, обладая достоинствами (универсальностью и двунаправленностью) изначально предложенного в [4] алгоритма, лишён его основного недостатка (низкой скорости).

Выводы. Предлагаемые в данной статье модификации алгоритма [4] обеспечивают выигрыш (в худшем случае пятикратный) в скорости при незначительном дополнительном расходе оперативной памяти (подход с применением «встречных» префиксных деревьев), и скорость, на два порядка большую – при использовании полного банка словоформ (в этом случае расход памяти существенно увеличивается – 250 Мб против чуть менее 20 при словаре из 6000 начальных форм слов). При этом не наблюдается падение скорости анализа при увеличении размеров анализируемых текстов, что даёт основания считать предложенный метод предпочтительным при анализе текстов больших объёмов.

Хотя алгоритм анализа, основанный на использовании «встречных» префиксных деревьев, использует особенности постфиксного базиса преобразований строк (подразумевается, что исследуемый язык характеризуется постфиксным формообразованием) и является менее общим по сравнению с алгоритмом, описанным в [4], его использование предпочтительно для языков с постфиксным формообразованием, к которым, в том числе, относится и русский язык.

Таким образом, при использовании модели описания формообразования естественных язы-

ков [6] для достижения максимальной скорости определения форм слов языка с постфиксным формообразованием в условиях нехватки памяти следует применять метод анализа с применением «встречных» префиксных деревьев, а при большом количестве доступной оперативной памяти и критичности скорости анализа – метод анализа с использованием банка всех известных форм слов языка.

Библиографический список

1. *Пруцков А.В., Розанов А.К.* Методы морфологической обработки текстов // Прикаспийский журнал: управление и высокие технологии. 2014. № 3 (27). С. 119-133.

2. *Миронов В.В., Заволокин А.И., Розанов А.К.* Электронная информационно-поисковая система «Русско-английский математический словарь» // Информатизация образования и науки. 2013. № 19. С. 167-176.

3. *Миронов В.В., Заволокин А.И., Розанов А.К.* Проблема формализации правил русско-английского и англо-русского переводов текстов // Информатизация образования и науки. 2014. № 22. С. 149-160.

4. *Пруцков А.В.* Алгебраическое представление модели формообразования естественных языков // Cloud Of Science. 2014. Т. 1. № 1. С. 88-97.

5. *Пруцков А.В.* Методы поиска решений в лингвистических автоматизированных обучающих системах // Научно-техническая информация. Сер. 2: Информационные процессы и системы. 2006. № 4. С. 15-18.

6. *Розанов А.К.* Организация словаря в системах генерации и определения форм слов естественных языков // Вестник Рязанского государственного радиотехнического университета. 2014. № 49. С. 55-63.

7. *Миронов В.В., Розанов А.К.* Подходы к оптимизации алгоритма определения форм слов естественных языков, основанного на цепочках последовательных преобразований строк // Информатизация образования и науки. 2015. № 25. С. 43-54.

8. *Ахманова О.С.* Словарь лингвистических терминов. 2-е изд., стер. М.: Едиториал УРСС, 2004. 571 с.

9. *Розанов А.К.* Представление правил префиксных и постфиксных преобразований строк на основе префиксных деревьев // Новые информационные технологии в научных исследованиях: материалы XVIII Всероссийской науч.-техн. конф. студентов, молодых ученых и специалистов / Рязань, РГРТУ, 2013. С. 53-55.

10. *Fredkin Edward*. "Trie Memory" // Communications of the ACM #3 (9), September 1960. P. 490-499, Association for Computing Machinery, New York, USA, 1960.

11. *Гельбух А.Ф., Сидоров Г.О.* К вопросу об автоматическом морфологическом анализе флективных языков // Труды международной конф. Диалог – 2005. М. 2005. С. 92-96.

12. std:map container reference, std:map:at // материалы сайта CppReference.com. 2015. URL: http://en.cppreference.com/w/cpp/container/map/at (дата обращения: 23.02.2015).

13. System.Collections.Generic.Dictionary // MSDN Help. URL: http://msdn.microsoft.com/ru-ru/library/xfhwa508.aspx (дата обращения: 12.04.2015).

14. Java HashMap Reference // Java Documentation. URL: http://docs.oracle.com/javase /7/docs/api/java/util/HashMap.html (дата обращения: 12.04.2015).

15. Языкознание. Бол. энцикл. словарь / гл. ред. В.Н. Ярцева. 2-е изд. М.: Бол. рос. энцикл., 1998. 685 с.

16. Русская грамматика: научные труды. В 2-х т. / Брызгунова Е.А., Габучан К.В. (ред.). М.: Ин-т русского языка имени В.В. Виноградова, 2005. 1496 с.

17. Морфологический анализатор pymorphy2 // URL: https://pymorphy2.readthedocs.org/ (дата обращения: 29.05.2015).

18. The Linguistica project (University of Chicago) // URL: http://linguistica.uchicago.edu/

19. Морфологический анализатор Mystem // URL: https://tech.yandex.ru/mystem/