3. Осовский, С. Нейронные сети для обработки информации [Текст] / Осовский С.: Пер. с польского И.Д. Рудинского. – М.: Финансы и статистика, 2002. – 344 с..

4. Солдатова, О.П. Основы нейроинфоматики [Текст] : учеб. пособие / О.П. Солдатова. – Самара: Изд-во Самар, гос. аэрокосм, ун-та, 2006. – 132 с. : ил.– ISBN 5-7883-0467-9.

Р. Р. Вафин, Р. В. Насыров

ОБОСНОВАНИЕ ВЫБОРА АЛГОРИТМА СИНТАКСИЧЕСКОГО АНАЛИЗА ПРЕДЛОЖЕНИЙ ДЛЯ ПОСТРОЕНИЯ НЕПРОЕКТИВНОГО ДЕРЕВА

(Уфимский государственный авиационный технический университет)

В настоящее время задача извлечения информации из текста на естественном языке является актуальной во многих сферах. Например, в медицине извлечение информации применяется для задач анализа текста медицинских выписок, историй болезни, формализации различных медицинских документов, например клинических рекомендаций.

С точки зрения лингвистического подхода извлечение информации состоит из нескольких этапов: лексический, морфологический, синтаксический и семантический анализ [1]. Для текстов на русском языке первые два этапа не представляют особой сложности, так как существующие инструменты позволяют с довольно большой точностью проводить лексический и морфологический анализ.

Существенная проблема встречается на этапе синтаксического анализа. Синтаксический анализ это построение грамматической структуры предложения и выявление синтаксических связей между словами в предложении [2]. Синтаксический анализатор должен построить синтаксическую структуру предложения. В автоматическом синтаксическом анализе, как правило, применяют синтаксические деревья, которые отражают структуру и/или связи в предложении.

В данной работе ставится задача разбора и сравнения методов синтаксического анализа предложений для определения наиболее подходящих из них в качестве инструмента анализа предложений на русском языке.

Существуют два наиболее распространенных подхода к отражению синтаксической структуры предложения: деревья непосредственно составляющих и деревья зависимостей.

Дерево непосредственно составляющих состоит из нетерминальных символов (различных групп, например именная, предложная, глагольная т.д.) и терминальных символов, самих слов предложения. Для построения деревьев такого типа преимущественно используются алгоритмы, основанные на формальной грамматике, в частности на контекстно-свободных грамматиках Хомского. Существенных ограничением является то, что дерево составляющих

не может отразить связи между словами в разных частях предложения. С точки зрения формальной грамматики, это полученное дерево является проективным.

В русском языке доля непроективных деревьев довольно высока, поэтому деревья непосредственно составляющих не могут в полной мере отразить структуру предложения. Для этой цели могут применяться деревья зависимостей. Деревья зависимостей, в отличии от деревьев непосредственно составляющих, состоят только из терминальных символов, а дуги между вершинами имеют тип связи (объект, субъект и т.д.). Данный подход позволяет выявить зависимости между различными парами слов в предложении. В общем случае дерево зависимостей может быть как проективным, так и непроективным.

Один из распространенных классов алгоритмов анализа являются алгоритмы, основанные на переходах (англ. transition-based). Данные алгоритмы основаны на предсказании последовательности переходов, которые позволяют привести исходное состояние системы в конечное состояние, создавая синтаксическое дерево.

В общем случае система переходов это множество $S = (C, T, c_s, C_t)$, где C – это множество состояний, состоящих, как минимум, из буфера оставшихся вершин β и множество дуг зависимостей A

T – это множество переходов, каждый их которых является функцией, которая получает на вход состояние системы и возвращает новое состояние $t : C \to C$

c_S – это функция инициализации, которая отображает множество слов в предложении, $x = (w_0, w_1, \ldots, w_n)$ на множество вершин $\beta = [1, \ldots, n]$, где каждая вершина это вектор признаков слова (часть речи, число, род и т.д.)

$C_t \subseteq C$ – это множество конечных состояний [5].

Синтаксический анализ начинается с формирования начального состояния, после чего начальное состояние последовательно преобразуется путем применения к нему последовательности переходов. Каждый переход меняет состояние системы, пока не будет достигнуто конечное состояние. При достижении конечного состояния в множестве дуг A будет содержаться построенное дерево зависимостей для исходного предложения.

Определение последовательности переходов осуществляется при помощи оракула - функции, которая принимает на вход состояние системы и возвращает функцию перехода $o : C \to T$. В общем случае оракул является черным ящиком и может представлять собой нейронную сеть [6], машину опорных векторов [5] и т.д.

Для обучения оракула производится подготовка обучающей выборки из исходного банка деревьев (англ. treebank) который состоит из пар предложений и соответствующих им деревьев. Слова обрабатываются в таком же порядке, как и алгоритм парсера, и на каждом шаге определяется действие, которое бы

привело к построению дерева из обучающей выборки. Полученный набор признаков и примененных переходов используется для обучения оракула.

Основное различие алгоритмов, основанных на переходах, состоит в разных определениях состояния системы и различных множествах переходов. Оракулы, как правило, не рассматриваются в рамках самого алгоритма, а рассматриваются в рамках метода анализа, который является совокупностью системы переходов и конкретной реализации оракула, иногда с конкретной обученной моделью.

В зависимости от реализации состояния, системы делятся на основанные на стеке (stack-based) и основанные на списках (list-based). В системах основанных на стеке состояние является множеством состоящим из входного буфера, промежуточного стека и множества дуг зависимостей. Данные системы позволяют строить только проективные деревья [5]. В системах основанных на списках вместо одного промежуточного стека используются два списка, а также позволяют строить как проективные, так и непроективные деревья.

Другой подход к построению деревьев зависимостей это графовый (англ. graph-based). В рамках данного подхода строится некоторая модель, которая позволяет количественно оценивать различные деревья (графы) для предложения.

Приведем пример модели [10].

$$\text{score\_tree}(y) = \sum_{(i,j)\in y} \text{score}(i,j) \tag{1}$$

Здесь y – граф зависимостей; (i,j) – ориентированная дуга графа $y : (i,j) \in y$, обозначающая синтаксическую зависимость.

Обычно вес дуги задается линейной функцией:

$$\text{score}(i,j) = \vec{w} \cdot f(i,j) \tag{2}$$

Здесь $\vec{w}$ – вектор весов признаков – параметр модели; $f(i,j)$ – функция получения вектора признаков дуги (i,j).

Пусть имеется предложение s и пусть $Y(s)$ – множество всех деревьев, которые можно построить на предложении s. Задачу синтаксического анализа в введенных терминах можно записать следующим образом:

$$\text{parse}(s) = \underset{y\in Y(s)}{arg\,max}(\text{score\_tree}(y,\vec{w})) \tag{3}$$

Сложность данного алгоритма $O(n^3)$.

Другим примером графового алгоритма является алгоритм максимального остовного дерева (англ. Maximum Spanning Tree (MST)) [8]. В начале строится полносвязный граф и веса всех дуг задаются формулой (2). Для нахождения дерева с максимальным суммарным весом используется алгоритм Chu-Liu-Edmonds [11, 12] для нахождения максимального остовного ориентированного дерева (англ. maximum spanning tree in directed graphs). Данный алгоритм имеет квадратичную сложность $O(n^2)$.

Рассмотрим различные реализации приведенных алгоритмов. Критериями сравнения алгоритмов построения дерева зависимостей являются возможность построения непроективных деревьев, сложность алгоритма. Результаты анализа показаны в таблице 1.

В результате анализа приведенных алгоритмов было выявлено, что временная сложность алгоритмов основанных на переходах линейна либо квадратична, но умножается на сложность оракула, который в зависимости от реализации может иметь довольно высокую сложность. Сложность графовых алгоритмов выше, и является квадратичной и кубической. Для русского языка, как языка с довольно высоким процентом приложений со свободным порядком слов наиболее подходящими алгоритмами являются Covington non-projective [9], Planar[7] и графовый алгоритм Ryan McDonald[8]. В зависимости от сложности выбранного оракула наиболее быстрым может быть либо второй либо третий алгоритм.

Таблица 1. Сравнение алгоритмов синтаксического анализа

Алгоритм	**Метод построения дерева**	**Сложность (n – число лексем, oracle – сложность оракула)**	**Построение непроектив - ного дерева**
Nivre arc-eager [4,5]	transition-based	$O(n \cdot oracle)$	-
Nivre arc-standard [5]	transition-based	$O(n \cdot oracle)$	-
Covington non-projective [9]	transition-based	$O(n^2 \cdot oracle)$	+
Covington projective [9]	transition-based	$O(n^2 \cdot oracle)$	-
Planar (implemented by Carlos Gómez-Rodríguez) [7]	transition-based	$O(n \cdot oracle)$	+
Graph based parsing by Ryan McDonald using Spanning Tree Algorithm Chu-Liu-Edmonds [8]	graph-based	$O(n^2)$	+
Jason Eisner's parse algorithm [13]	graph-based	$O(n^3)$	-

Существуют существенные ограничения, связанные с синтаксической омонимией, которые невозможно решить только при помощи синтаксического анализа, и для повышения точности анализа необходимо применять гибридные методы анализа, например синтаксически-семантический анализ [10] или глубокий синтаксический анализ [3].

Литература

1. Большакова Е.И. Автоматическая обработка текстов на естественном языке и анализ данных : учеб. пособие / Большакова Е.И., Воронцов К.В., Ефремова Н.Э. – М.: Изд-во НИУ ВШЭ, 2017 – 269 с.

2. Анисимов А. В., Марченко А. А. Система обработки текстов на естественном языке //Искусственный интеллект. – 2002. – №. 4. – С. 157-163.

3. Попов А. М., Протопопова Е. В., Букия Г. Т. Ещё раз о способах снятия структурной омонимии: выбор единственной структуры в парсере Hurma //Труды объединённой научной конференции" Интернет и современное общество". – 2016. – С. 65-73.

4. Goldberg Y., Nivre J. A dynamic oracle for arc-eager dependency parsing //Proceedings of COLING 2012. – 2012. – С. 959-976.

5. Nivre J. Algorithms for deterministic incremental dependency parsing //Computational Linguistics. – 2008. – Т. 34. – №. 4. – С. 513-553.

6. Parsing Universal Dependency Treebanks using Neural Networks and Search-Based Oracle Milan Straka

7. Gómez-Rodríguez C., Nivre J. A transition-based parser for 2-planar dependency structures //Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. – Association for Computational Linguistics, 2010. – С. 1492-1501.

8. McDonald R. et al. Non-projective dependency parsing using spanning tree algorithms //Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing. – Association for Computational Linguistics, 2005. – С. 523-530.

9. Covington M. A. A fundamental algorithm for dependency parsing //Proceedings of the 39th annual ACM southeast conference. – 2001. – С. 95-102.

10. Шелманов А. О. Исследование методов автоматического анализа текстов и разработка интегрированной системы семантико-синтаксического анализа //дисс. канд. техн. наук. – 2015. – Т. 5. – С. 17.

11. Edmonds J. Optimum branchings //Journal of Research of the national Bureau of Standards B. – 1967. – Т. 71. – №. 4. – С. 233-240.

12. Chu Y. J., Liu T. H. On the Shortest Arborescence of a Directed Graph // Science Sinica. – 1965. – Т. 14. – С. 1396-1400.

13. Eisner J. Bilexical grammars and their cubic-time parsing algorithms //Advances in probabilistic and other parsing technologies. – Springer, Dordrecht, 2000. – С. 29-61.