УДК 004.852 DOI: 10.18698/2541-8009-2017-2-63

АЛГОРИТМ ПОСТРОЕНИЯ ГРАФА ТЕКСТА НА ОСНОВЕ СИНТАКСИЧЕСКИХ СВЯЗЕЙ

Б.Н. Шафорост borisshaforost@mail.ru

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Аннотация

Рассмотрен алгоритм построения графа текста, основанный на синтаксических связях между словами, где вершинами являются слова, а дугами — связи между словами. Подобный граф позволяет изобразить синтаксическую конструкцию исследуемого текста и наглядно оценить уровень значимости каждого отдельного слова по количеству связей вершины. Предложенный алгоритм дает возможность построить граф текста высокой синтаксической сложности и значительного объема, что особенно актуально для решения некоторых практических задач компьютерной лингвистики. Последовательно рассмотрены этапы построения графа. Подробно описан метод определения координат для каждой вершины с учетом позиций уже построенных вершин

Ключевые слова

Граф, координаты, алгоритм, вершина, дуга, синтаксическая связь, лингвистика, визуализация

Поступила в редакцию 21.12.2016

Задача построения графа является одной из важнейших в современном программировании, поскольку граф позволяет наглядно оценить структуру рассматриваемой системы и взаимосвязь отдельных ее частей, выявить ключевые элементы системы на основе анализа связей, а также оценить степень удаленности элементов друг от друга. Задачей исследования является построение графа некоторого текста на основе синтаксических связей между словами. Слова будут вершинами, а в качестве дуг будут выступать синтаксические связи между ними. Подобный граф может быть использован для разрешения ряда задач в современной компьютерной лингвистике, в частности, для выделения ключевых слов текста.

Чтение исходных данных. В качестве исходных данных использованы документы системы синтаксического анализа русских текстов на базе корпуса «СинТагРус», созданной в Лаборатории компьютерной лингвистики Института проблем передачи информации им. А.А. Харкевича РАН. В базе представлены различные тексты с синтаксической разметкой. Для каждого предложения текста определен уникальный номер (S ID), и внутри предложения каждое слово имеет уникальный номер (ID). Также отмечены синтаксические связи внутри предложений: выделено корневое слово (root), которое не имеет собственной ссылки, а все остальные слова внутри данного предложения имеют ссылку (DOM) на какое-либо слово (на номер слова) внутри данного

предложения [1, 2].

Рассмотрим построение графа двумя способами. Первый — предложения не будут связаны между собой, а при визуализации будут выглядеть как отдельные графы. Второй — построение цельного графа всего текста, а для связи предложений будем использовать корневые элементы. Каждый последующий корневой элемент должен ссылаться на предыдущий. Данный подход является обоснованным, поскольку именно корневое слово является главным в предложении, в нем отражена ключевая информация. Поскольку предложения в тексте связаны (расположены) последовательно, то и корневые элементы свяжем так же.

Сначала необходимо перевести имеющийся документ в массивы данных. Во-первых, потребуется номер (ID) слова. Для построения всего графа необходимо, чтобы каждый элемент имел уникальный ID внутри всего текста, а не внутри предложения, как в исходном документе. Однако, вводить эти данные в массив нет необходимости, поскольку можно использовать индексы для однозначной идентификации элемента. Во-вторых, потребуется массив, который будет содержать ссылку (DOM, то есть ID слова, на которое ссылается наше слово, в нашем случае — индекс элемента, на который ссылаемся). Также для первого способа построения графа, необходимо ввести дополнительно массив флагов, показывающих наличие или отсутствие связи для исключения из визуализации связей между предложениями [3]. Псевдокод для вышеперечисленных действий будет следующим:

```
    Декларируем целочисленную переменную ПредыдКорень = 0
    Декларируем целочисленную переменную ДобавНомер = 0
    Декларируем целочисленный массив АбсССылки
    Декларируем массив флагов ДугаЕсть
Цикл:
    Пока не достигнут конец документа Делаем:
            Если текущий элемент не первый, и номер предложения предыдущего
            элемента меньше номера предложения текущего элемента Делаем:
                ДобавНомер = индекс предыдущего элемента
            Если текущий элемент корневой Делаем:
            АбсССылки[индекс текущего элемента] = (Делаем считывание ссылки
            из документа)  + ПредыдКорень
                ПредыдКорень = индекс текущего элемента
                ДугаЕсть[индекс текущего элемента] = ложь
            Иначе Делаем:
                АбсССылки[индекс текущего элемента] = (Делаем считывание ссылки
                из документа)  + ДобавНомер
                ДугаЕсть[индекс текущего элемента] = истина
Конец цикла
```

В результате получаем массив АбсССылки, внутри которого каждый элемент (кроме первого корневого) имеет ссылку на индекс другого элемента этого массива, а первый корневой элемент обозначен как 0. Если в языке

программирования индексация массива начинается с 0, то следует уменьшить все считанные ссылки на 1, тогда ссылочная целостность сохранится, а первый корневой элемент будет иметь значение внутри массива –1. В ходе настоящей работы (Язык C#) было сделано именно так. Будем рассматривать именно такой способ индексации. Также получен дополнительный массив флагов ДугаЕсть, необходимый для визуализации графа, который имеет флаг Истина во всех случаях, когда элемент некорневой.

Определение координат. Необходимо определить координаты для каждой вершины. Единицы их измерения могут быть различны, поэтому обозначим их как $+1x$ — смещение по горизонтали вправо и $+1y$ — смещение по вертикали вверх. Соответственно, $-1x$ — смещение влево, $-1y$ — вниз.

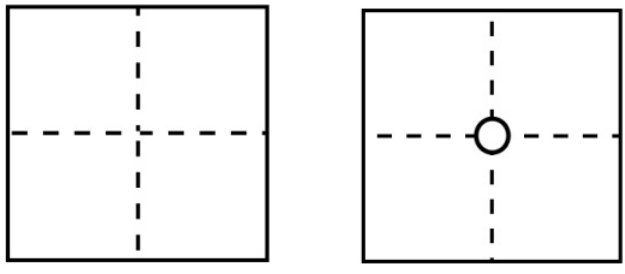
Каждую построенную вершину (изначально — первый корневой элемент) признаем будущим строителем и формируем массив из таких вершин будущих строителей (массив указателей на них, то есть массив их ID, в нашем случае — их индексов). Когда «закончатся» все вершины, которые ссылаются на текущую вершину-строитель, необходимо скопировать массив будущих строителей в массив текущих, очистить массив будущих строителей и выполнить аналогичное построение для всех элементов массива текущих строителей, одновременно пополняя массив будущих строителей. Данное действие будет выполняться до тех пор, пока массив будущих строителей не окажется пуст, то есть до тех пор, пока не будут построены все конечные элементы (на которые уже нет ссылок). Работа с массивами текущих и будущих строителей осуществляется с помощью основной функции построения, выбор позиции для построения — с помощью функции ОпредПозиц, а проверка, является ли выбранная позиция свободной, либо следует искать другую — в функции СвободнаЛиПоз.

Дуга соединяет две вершины, поэтому для определения дуги (с учетом того, что решено перейти к единому массиву ссылок), достаточно координат текущей вершины (откуда дуга выходит) и координат иной вершины (с которой дуга соединяет текущую вершину). Если текущая вершина будет определена, то координаты иной вершины зафиксируем в массивах АркаХ и АркаУ. Псевдокод алгоритма построения будет следующим:

```
Основная функция
Декларируем целочисленный массив ТекущПоиск
Декларируем целочисленный массив СледПоиск
Декларируем численные массивы КоордХ, КоордУ, инициализируем их нулями.
Декларируем численные массивы АркаХ, АркаУ (численные — значит допустимы и дробные числа)
Цикл:
    Пока не достигнут конец массива АбсССылки Делаем:
        Если текущий элемент = –1 (первый корень) Делаем:
        СледПоиск[0] = индекс текущего элемента в массиве АбсССылки
        Прерывание цикла (нашли первый корневой элемент)
Конец цикла
```

```
Цикл:
    Пока массив СледПоиск не пуст Делаем
        Копируем Массив СледПоиск в массив ТекущПоиск
        Обнуляем массив СледПоиск
    Цикл:
        Пока не достигнут конец массива ТекущПоиск Делаем:
            Цикл:
                Пока не достигнут конец массива АбсССылки Делаем:
                Если Текущий элемент массива АбсССылки = текущему
        элементу массива ТекущПоиск Делаем:
                СледПоиск[добавляем в конец массива, при необходимости
                изменяем его размер] = текущий индекс массива АбсССылки
                Вызов функции ОпредПозиц(текущий индекс массива
            АбсССылки,   текущий элемент массива ТекущПоиск)
            Конец цикла
    Конец цикла
Конец цикла
```

Далее рассмотрим алгоритм функции выбора позиции для вершины ОпредПозиц. Представим себе квадрат. Мысленно разделим его пополам по вертикали и горизонтали. Получим четыре маленьких одинаковых квадрата, стоящих рядом (рис. 1, *а*). Каждый из квадратов двумя сторонами будет соприкасаться с другими квадратами, а двумя другими сторонами — выходить во вне исходного квадрата. Нас интересуют внешние грани. У каждого из четырех маленьких квадратов таких две, то есть у исходного квадрата их восемь.



а *б*

Рис. 1. Квадрат, разделенный на четыре части (*а*)
и положение вершины-строителя на нем (*б*)

Наш алгоритм построения предполагает, что исходная вершина (вершина-строитель, в начале алгоритма — это первый корневой элемент, значение в массиве АбсССылки равно –1) находится в центре исходного квадрата, то есть на пересечении разделения по вертикали и горизонтали (рис. 1, *б*).

Будем считать, что нам необходимо разместить позицию следующего элемента на внешней границе исходного квадрата, а если на этой границе не будет места (вся граница занята построенными ранее вершинами с учетом шага 1x и 1y), то увеличим размер квадрата и будем искать до тех пор, пока не найдем свободное место (рис. 2).

Как показала практика, при визуализации больших графов, наилучший результат (наименьшее число пересечений дуг) достигается в том случае, когда сначала пробуют новую вершину поставить на одну из внешних сторон четырех маленьких квадратов, но не в углы маленьких квадратов (при условии, что размер квадрата уже хотя бы раз был увеличен. Иначе это действие невозможно из-за установленного шага 1x и 1y), далее в углы большого квадрата, а затем — вдоль осей горизонтали и вертикали. То есть, логически разделяем все возможные построения (без перехода к новому размеру квадрата) на 16 частей (относительно центра исходного квадрата):

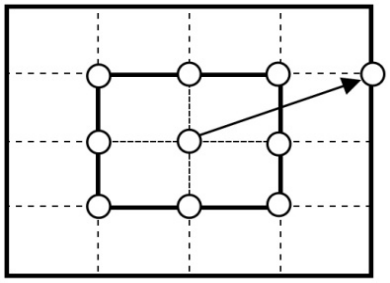
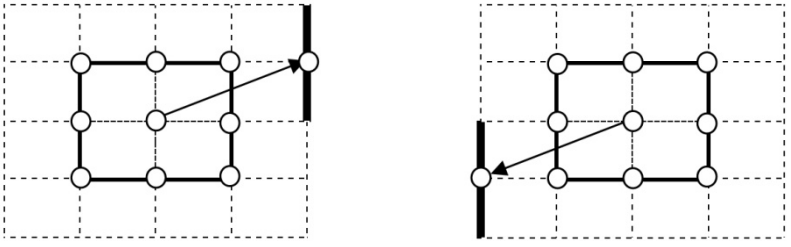


Рис. 2. Увеличение размера квадрата

1) верхняя часть правой грани (рис. 3, *а*);



а *б*

Рис. 3. Верхняя часть правой (*а*) и нижняя часть левой (*б*) граней

2) нижняя часть левой грани (рис. 3, *б*). Двигаемся в противоположную сторону, потому что если встретились с другой вершиной, то предполагаем, что в данном направлении будут еще вершины, поэтому нужно выбрать другое направление. Такой подход позволяет снизить вероятность пересечений дуг или слишком тесного расположения вершин графа, которое снижает его наглядность и затрудняет построение;

3) левая часть верхней грани;

4) правая часть нижней грани;

5) нижняя часть правой грани;

6) верхняя часть левой грани;

7) правая часть верхней грани;

8) левая часть нижней грани. Как было отмечено ранее, первые восемь вариантов возможны только в том случае, если хотя бы один раз переходили к новому размеру квадрата. Однако первые восемь вариантов не включали крайние точки. Это сделано, чтобы при невозможности построить вершину при текущем размере квадрата, и, соответственно, при увеличении размеров квадрата, минимизировать пересечения и наложения дуг за счет появления иных углов между дугами и осями горизонтали и вертикали (рис. 4);

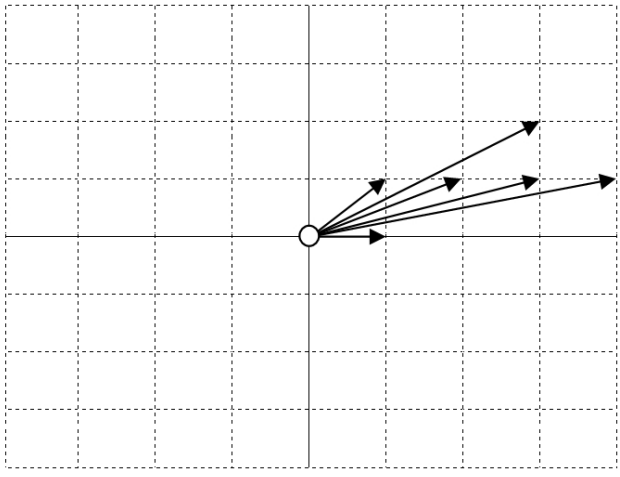


Рис. 4. Возможные направления дуг

9) правый верхний угол — точка пересечения граней (рис. 5, *а*);

а *б*

Рис. 5. Правый верхний угол (*а*) и направление по горизонтали вправо (*б*)

10) левый нижний угол;
11) левый верхний угол;
12) правый нижний угол;
13) по горизонтали вправо (рис. 5, *б*);
14) по горизонтали влево;
15) по вертикали вверх;
16) по вертикали вниз.

Когда все свободные места в рамках данного квадрата заняты, то увеличиваем размер квадрата и повторяем поиск (см. рис. 2). Псевдокод функции выбора позиции будет следующим:

```
Функция ОпредПозиц (ИндексТек, Строитель)
АркаХ[ИндексТек] = КоордХ[Строитель]
АркаУ[ИндексТек] = КоордУ[Строитель]
Декларируем целочисленную переменную НомерКруга = 1
Бесконечный цикл:      (для построения графа любой сложности)
    Цикл: (Декларируем переменную ТекУ = +1у;  До тех пор пока ТекУ < +1у* НомерКруга)
```

```
        Если Функция СвободнаЛиПоз (ИндексТек, Строитель, +1х * НомерКруга,
ТекУ) вернет истину Делаем:
                Выход из функции
        ТекУ = ТекУ +1у
    Конец цикла  (1:  Верхняя часть правой грани)
    Цикл: (Декларируем переменную ТекУ = +1у;  До тех пор пока ТекУ < +1у*
НомерКруга)
        Если Функция СвободнаЛиПоз (ИндексТек, Строитель, –1х * НомерКруга,
    –ТекУ) вернет истину Делаем:
                Выход из функции
        ТекУ = ТекУ +1у
    Конец цикла  (2:  Нижняя часть левой грани)
    Цикл: (Декларируем переменную ТекХ = +1х;  До тех пор пока ТекХ < +1х*
НомерКруга)
        Если Функция СвободнаЛиПоз (ИндексТек, Строитель, -1 ТекХ, +1у *
НомерКруга) вернет истину Делаем:
                Выход из функции
        ТекХ = ТекХ +1х
    Конец цикла  (3:  Левая часть верхней грани)
Варианты 4–16 — по аналогии с 1–3.
НомерКруга = НомерКруга +1
Граница бесконечного цикла
```

Функция СвободнаЛиПоз осуществляет проверку возможности занятия новой вершиной некоторой позиции. В случае, если ни одна из существующих вершин не занимает эту позицию, функция возвращает Истина и устанавливает координаты вершины, иначе возвращает Ложь, и функция ОпредПоз продолжает поиск свободной позиции. Псевдокод будет следующим:

```
Функция СвободнаЛиПоз (инд, Строитель, СмещХ, СмещУ)
Цикл:
    Пока не достигнут конец массива АбсСылки  Делаем:
        Если (КоордХ[текущий элемент массива Абссылки] = КоордХ[Строитель] +
        СмещХ И
        КоордУ[текущий элемент массива Абссылки] = КоордУ[Строитель] +
    СмещУ) Делаем:
            Выход из функции со значением Ложь
Конец цикла
КоордХ[инд] = КоордХ[Строитель] + СмещХ
КоордУ[инд] = КоордУ[Строитель] + СмещУ
Выход из функции со значением Истина
```

Визуализация. Итак, имея координаты всех вершин и дуг, можно приступить к визуализации. Для этой цели была использована библиотека OpenGL (Tao Framework для C#). Сначала последовательно рисуем все вершины в соответствии с их координатами, затем — все связи. Далее используем массив ДугаЕсть, который получили в начале работы. Если строить полносвязный граф всего текста (связь корень-корень между предложениями), то в случае, когда элемент массива ДугаЕсть равен Ложь (корневой элемент), можно выделить дугу другим цветом, чтобы показать, что данная связь является связью между предложениями. В случае, если

изображаем отдельные предложения, дугу вообще не рисуем. Также в обоих случаях не изображаем дугу первого корневого элемента (имеет в массиве АбсСылки значение –1), потому что он не имеет собственной ссылки и с него начинается алгоритм построения графа. Визуализация результата работы предложенного алгоритма представлена на рис. 6.

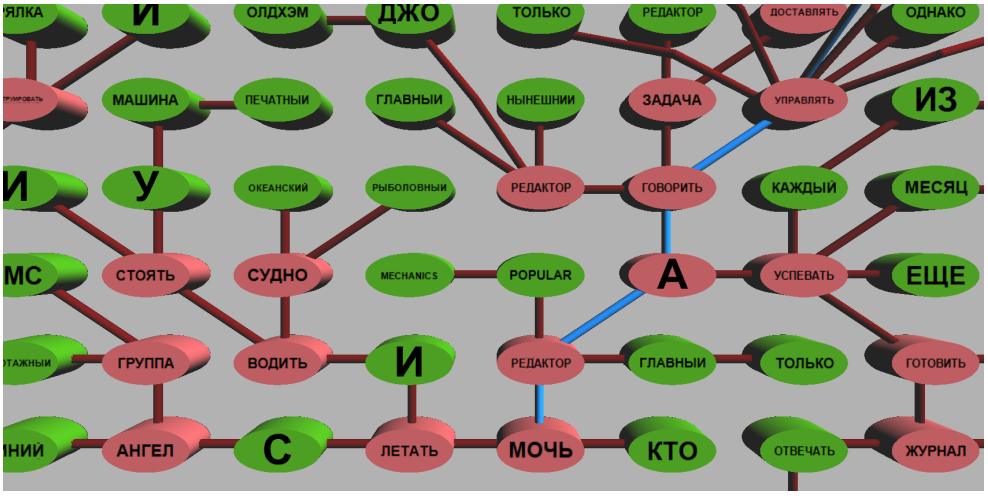


Рис. 6. 3*D*-визуализация результата работы алгоритма построения графа текста на основе синтаксических связей (скриншот)

Выводы. Предложенный алгоритм позволяет выполнять построение планарного графа для текста неограниченной синтаксической сложности. Это достигается за счет того, что поиск координат для новой вершины не ограничен заранее определенным набором позиций и осуществляется до тех пор, пока свободная позиция не будет найдена. Конечно, такой подход не исключает возможной удаленности вновь построенной вершины от вершины-источника связи, однако гарантирует целостность графа и возможность его построения в одной плоскости, что является немаловажным для визуального восприятия. Кроме того, предложенный алгоритм может быть использован для построения графа любой системы элементов неограниченного размера, внутри которой каждый элемент, кроме стартового, имеет одну связь с некоторым элементом данной системы. В случае присутствия большего числа связей у элементов, возникает необходимость построения дополнительных связей, что не исключает возможности применения алгоритма для определения координат вершин графа.

Литература

1. *MSDN:* сеть разработчиков Microsoft. URL: https://msdn.microsoft.com (дата обращения: 10.12.2016).

2. *Синтаксически* размеченный корпус русского языка: информация для пользователей // Национальный корпус русского языка: веб-сайт.
URL: http://www.ruscorpora.ru/instruction-syntax.html (дата обращения: 10.12.2016).

3. *Псевдокод* (язык описания алгоритмов).
URL: https://ru.wikipedia.org/wiki/Псевдокод_(язык_описания_алгоритмов) (дата обращения: 10.12.2016).

Шафорост Борис Николаевич — студент кафедры «Программное обеспечение ЭВМ и информационные технологии» (второе высшее образование), МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Научный руководитель — Л.Л. Волкова, ассистент кафедры «Программное обеспечение ЭВМ и информационные технологии», МГТУ им. Н.Э. Баумана, Москва Российская Федерация.

ALGORITHM OF GENERATING A TEXT GRAPH BASED ON SYNTACTIC RELATIONS

B.N. Shaforost borisshaforost@mail.ru

Bauman Moscow State Technical University, Moscow, Russian Federation

Abstract

The article deals with an algorithm for generating a text graph based on syntactic relations between words. The vertices represent the words, and the arrows represent the relations between the words. This type of graph makes it possible to visualise the syntactic structure of the text being studied and visually evaluate the degree of importance for any given word via the number of edges connected to its vertex. The algorithm suggested makes it possible to generate graphs for texts characterised by a high syntactic complexity and a significant volume, which is especially relevant for solving certain practical problems of computational linguistics. We consider all the stages of graph generation in sequence. We describe in detail the method of determining coordinates of each vertex taking into account the positions of vertices already generated

Keywords

Graph, coordinates, algorithm vertex, arrow, syntactic relation, linguistics, visualisation

References

[1] MSDN: Microsoft developers network. URL: https://msdn.microsoft.com (accessed 10.12.2016).

[2] Sintaksicheski razmechennyy korpus russkogo yazyka: informatsiya dlya pol'zovateley [syntactically marked Russian language corpus]. Natsional'nyy korpus russkogo yazyka: website. URL: http://www.ruscorpora.ru/instruction-syntax.html (accessed 10.12.2016) (in Russ.).

[3] Psevdokod (yazyk opisaniya algoritmov) [Pseudocode (algorithms description language]. URL: https://en.wikipedia.org/wiki/Pseudocode (accessed 10.12.2016).

Shaforost B.N. — student of Computer Software and Information Technologies Department (second university degree program), Bauman Moscow State Technical University, Moscow, Russian Federation.

Scientific advisor — L.L. Volkova, assistant of Computer Software and Information Technologies Department, Bauman Moscow State Technical University, Moscow Russian Federation.