ISSN 1818–1015 (Print)
ISSN 2313–5417 (Online)

Министерство образования и науки Российской Федерации
Ярославский государственный университет им. П. Г. Демидова

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ

Том 25 № 4 (76) 2018

Основан в 1999 году
Выходит 6 раз в год

Адрес редакции: ЯрГУ, ул. Советская, 14, г. Ярославль, 150003, Россия
Website: http://mais-journal.ru, e-mail: mais@uniyar.ac.ru; телефон (4852) 79-77-73

Научные статьи в журнал принимаются по электронной почте. Статьи должны содержать УДК, аннотации на русском и английском языках и сопровождаться набором текста в редакторе LaTeX. Плата с аспирантов за публикацию рукописей не взимается.

12+

СОДЕРЖАНИЕ

Моделирование и анализ информационных систем. Т. 25, №4. 2018

Параллельное программирование

Верификация программ

Анализ сигналов

Теория графов

Технология блокчейн

Вопросно-ответные системы

Динамические модели процессов

Тезаурусы

Свидетельство о регистрации СМИ ПИ № ФС 77 – 66186 от 20.06.2016 выдано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций. Учредитель – Федеральное государственное бюджетное образовательное учреждение высшего образования "Ярославский государственный университет им. П. Г. Демидова". Подписной индекс – 31907 в Объединенном каталоге "Пресса России". Редактор, корректор А.А. Аладьева. Редактор перевода Э.И. Соколова. Подписано в печать 20.08.2018. Дата выхода в свет 30.08.2018. Формат 60x84$^1/_8$. Усл. печ. л. 13,48. Уч.-изд. л. 12,0. Объем 116 с. Тираж 46 экз. Свободная цена. Заказ 067/018. Адрес типографии: ул. Советская, 14, оф. 109, г. Ярославль, 150003, Россия. Адрес издателя: Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, г. Ярославль, 150003, Россия.

ISSN 1818–1015 (Print)
ISSN 2313–5417 (Online)

P.G. Demidov Yaroslavl State University

MODELING AND ANALYSIS OF INFORMATION SYSTEMS

Volume 25 No 4 (76) 2018

Founded in 1999
6 issues per year

Editorial Office Address: P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia
Website: http://mais-journal.ru, e-mail: mais@uniyar.ac.ru

Contents

Modeling and Analysis of Information Systems. Vol. 25, No 4. 2018

Parallel Programming

Program Verification

Signal Analysis

Graph Theory

Blockchain Technology

Question Answering Systems

Dynamic Process Modeling

Thesauri

Моделирование и анализ информационных систем. Т. 25, № 4 (2018), с. 347–357
Modeling and Analysis of Information Systems. Vol. 25, No 4 (2018), pp. 347–357

Параллельное программирование

DOI: 10.18255/1818-1015-2018-4-347-357

УДК 004.052.42

Оптимизация инварианта цикла в языке Пифагор

Васильев В. С., Легалов А. И.

получена 15 марта 2018

Аннотация. В работе рассматриваются методы преобразования программ, эквивалентные оптимизации инварианта цикла, применительно к функционально-потоковой модели параллельных вычислений, реализованной в языке программирования Пифагор. В императивных языках при оптимизации инварианта из цикла выносятся вычисления, не зависящие от изменяемых в нем переменных. Особенностью языка функционально-потокового параллельного программирования Пифагор является отсутствие явно задаваемых циклических вычислений (оператора цикла). Тем не менее, повторяющиеся вычисления в этом языке можно задать рекурсивно или за счет применения специфических языковых конструкций (параллельных списков). Оба механизма обеспечивают возможность параллельного выполнения. В случае оптимизации рекурсивной функции повторяющиеся операции выносятся во вспомогательную функцию, а основная функция выполняет лишь вычисление инварианта. При оптимизации внутри параллельных списков вычисление инварианта перемещается в дополнительную функцию, содержащую вызов функции, использующую данный параллельный список. В статье приводится определение «инварианта» применительно к языку Пифагор, алгоритмы его оптимизации, а также примеры программ, их графовых представлений (граф программных зависимостей) до и после оптимизации. Алгоритм оптимизации, описанный для вычислений над параллельными списками, применим только для языка Пифагор, так как опирается на специфические структуры данных и модель вычислений этого языка. Вместе с тем, алгоритм преобразования рекурсивных функций может быть применим и для других языков программирования.

Ключевые слова: функционально-потоковое параллельное программирование, язык программирования Пифагор, оптимизация кода, оптимизация циклов, оптимизация инварианта, граф программных зависимостей

Для цитирования: Васильев В. С., Легалов А. И., "Оптимизация инварианта цикла в языке Пифагор", *Моделирование и анализ информационных систем*, **25**:4 (2018), 347–357.

Об авторах:
Васильев Владимир Сергеевич, orcid.org/0000-0002-3340-6678, аспирант,
Сибирский федеральный университет, Институт космических и информационных технологий
ул. Академика Киренского, 26, г. Красноярск, 660074 Россия, e-mail: vsvasilev@sfu-kras.ru

Легалов Александр Иванович, orcid.org/0000-0002-5487-0699, д-р техн. наук, профессор,
Сибирский федеральный университет, Институт космических и информационных технологий
ул. Академика Киренского, 26, г. Красноярск, 660074 Россия, e-mail: legalov@mail.ru

Благодарности:
Исследование выполняется при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00288.

Введение

Оптимизация представляет собой процесс эквивалентного преобразования, в результате которого устраняются избыточные вычисления, не влияющие на ход выполнения программы, а следовательно, улучшаются требуемые характеристики. Одним из таких преобразований, широко используемых в императивном программировании, является оптимизация инварианта цикла [1]. Вычисление называется инвариантом цикла (loop-invariant computation), если для определенной группы операторов дает один и тот же результат независимо от того, сколько раз выполнится тело цикла. При оптимизации инварианта цикла (loop-invariant code motion) такие вычисления выносятся из тела цикла и размещаются перед ним, за счет чего сокращается объем производимых вычислений [2].

Особенности оптимизации инварианта цикла в императивных языках можно рассмотреть на следующем простом примере для языка программирования C++:

```
int inv (int n, int p) {
  int s = 0;
  int i = 0;




  while (i < n-2) {
    s += p*3;
    ++i;
  }
  return s;
}
```

```
int inv (int n, int p) {
  int s = 0;
  int i = 0;

  int inv_n = n-2;
  int inv_p = p*3;

  while (i < inv_n) {
    s += inv_p;
    ++i;
  }
  return s;
}
```

Фрагмент кода, размещенный в левой части, содержит цикл, в котором на каждой итерации вычисляются операции *n-2* и *p*3*, т. к. внутри цикла значения переменных n и p не изменяются, возможен их вынос из цикла. Оптимизированный вариант программы показан в правой части листинга.

Язык Пифагор разрабатывается как инструмент для написания архитектурно-независимых параллельных программ. В его основе лежит принцип управления вычислениями по готовности данных (dataflow) [3]. Программа на данном языке состоит из функций, каждая из которых однозначно отображается в граф потока данных (dataflow graph, ГПД), отражающий зависимости по данным между операторами [4]. Формируемый ГПД является ацикличным, так как язык применяет принцип единственного использования вычислительных ресурсов, по сути усиливающий принцип единственного присваивания [5]. Именно на этом графе можно проводить разнообразные оптимизационные преобразования, обеспечивающие в дальнейшем порождение более эффективного исполняемого кода. Специфика лежащей в основе языка модели вычислений позволяет формировать повторяющиеся вычисления либо с помощью рекурсии, либо за счет операций над параллельными списками.

1. Оптимизация инварианта в рекурсивной функции

Инвариантом рекурсивной функции, по аналогии с инвариантом цикла, будем считать вычисления, результат которых не будет изменяться в ходе повторного выполнения внутри рекурсивных вызовов. Такие вычисления зависят от констант и значений, неизменяемых между рекурсивными вызовами.

Представленная ниже на языке Пифагор функция описывает вычисления, аналогичные рассмотренному выше примеру на языке C++. На рис. 1 приведен ГПД, соответствующий этой функции. Передаваемый в функцию аргумент является списком, содержащим четыре значения, которые обозначены аналогично переменным из предшествующего примера.

```
rec_inv << funcdef X {
  i << X:1;
  n << X:2;
  s << X:3;
  p << X:4;
  [((i, (n,2):-):[=>, <]):?]^(
    s,
    {
      block {
        next_s << (s, (p,3):*):+;
        break << ((i,1):+, n, next_s, p)
          :rec_inv;
      }
    }
  ):. >> return;
}
```

В приведенном примере:

- неизменяемыми аргументами являются *«n»* и *«p»* (второй и четвертый элементы списка *«X»* соответственно);
- инвариантами являются вычисления *«(n, 2):–»*, *«(p, 3):*»*, а также операции внутри параллельного списка *«[=>, <]»*. Из рис. 1 видно, что эти вычисления не зависят от изменяемых аргументов рекурсивной функции (*«i»* и *«s»*).

Несмотря на то что формирование параллельного списка *«[=>, <]»* формально является инвариантом, его нельзя оптимизировать, так как значение инварианта передается в функцию в качестве дополнительного аргумента, но согласно алгебре эквивалентных преобразований параллельный список при выполнении такой операции раскроется [5].

Для выделения фрагмента кода, являющегося инвариантом в рекурсивной функции, необходимо:

- построить множество *Args* из узлов, являющихся аргументами функции;

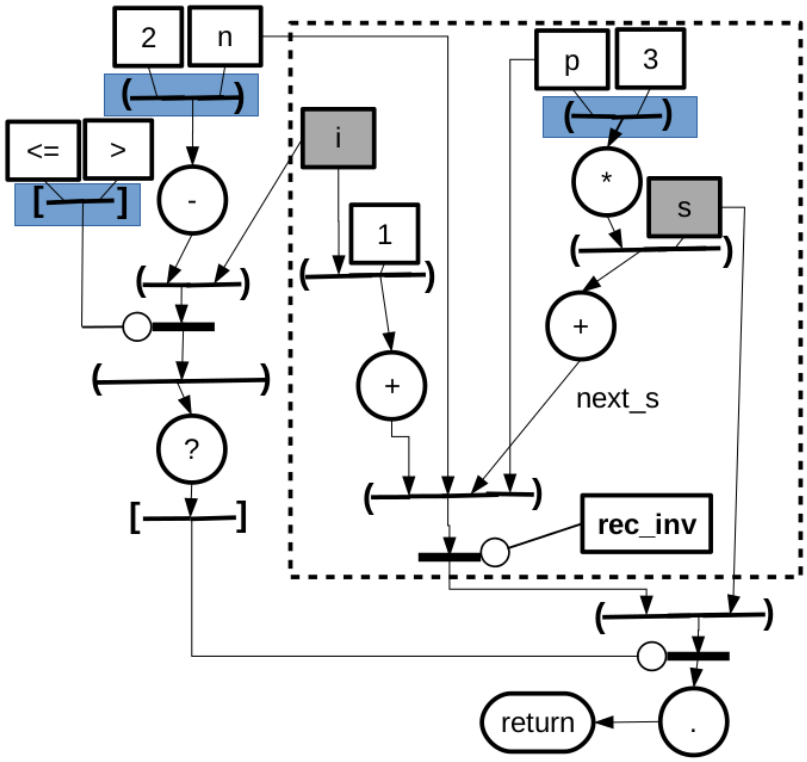

Рис. 1. ГПД рекурсивной функции с инвариантом
Fig. 1. The DFG of a recursive function with invariant

- из множества *Args* выделить подмножество *ConstArgs* неизменяемых аргументов;
- из множества операторов программы выделить инвариант (множество *Invs*), используя следующий прием: узел графа принадлежит этому множеству, если не является узлом формирования параллельного списка и имеет зависимости по данным только от констант, являющихся элементами множества *ConstArgs* и других элементов множества *Invs*.

Для оптимизации функции F, содержащей инвариант, выполняется создание вспомогательной функции G, в которую переносятся вычисления инвариантов из функции F и добавляется узел вызова функции F с передачей в нее результатов вычисления в качестве дополнительных аргументов. В функции F при этом обращения к значениям инвариантов заменяются на обращения к соответствующим аргументам. На рис. 2 приведен ГПД рассмотренной выше функции после оптимизации. В следующем листинге представлен текст функции на языке Пифагор, описывающий проведенные преобразования.

```
rec_inv_opt_h << funcdef X {
  i << X:1;
  n << X:2;
  s << X:3;
```

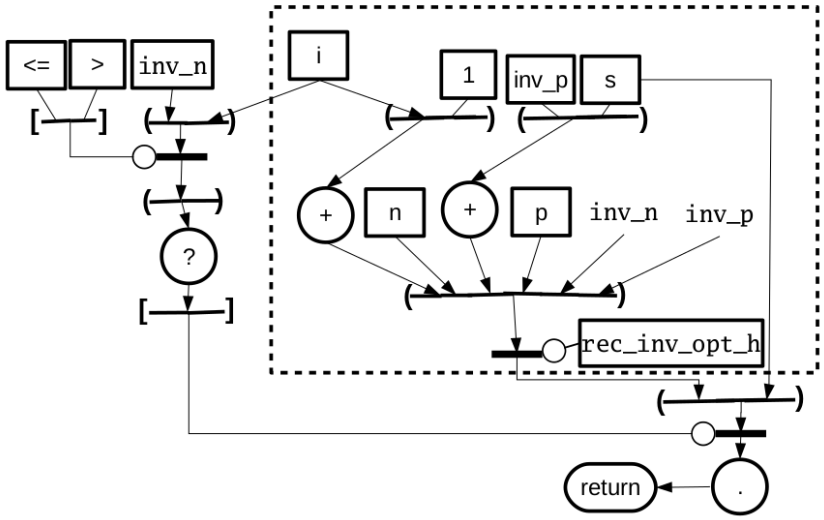

Рис. 2. ГПД рекурсивной функции после оптимизации инварианта
Fig. 2 The DFG of the recursive function after optimization of the invariant

```
  p << X:4;
  inv_n << X:5;
  inv_p << X:6;
  [((i, inv_n):[=>, <]):?]^(
    s,
    {
      block {
        next_s << (s, inv_p):+;
        break << ((i,1):+, n, next_s, p, inv_n, inv_p)
          :rec_inv_opt_h;
      }
    }
  ):. >> return;
}

rec_inv << funcdef X {
  n << X:2;
  p << X:4;
  inv_n << (n,2):-;
  inv_p << (p,3):*;
  return << (X:[], inv_n, inv_p)
    :rec_inv_optimization_h;
}
```

Следует отметить что результаты оптимизации формируются в промежуточном представлении без порождения исходных текстов функций. В данном случае и ниже

текстовое представление функций приведено только для наглядной демонстрации результатов проводимых оптимизаций.

2. Оптимизация инварианта в вычислениях с параллельными списками

В ряде случаев рекурсивные вычисления в языке Пифагор возможно заменить операциями над параллельными списками, обеспечивающими описание одновременной обработки всех данных одной функцией. При этом удается добиться более высокой эффективности распараллеливания, чем при использовании рекурсии, в частности, возможна трансляция таких конструкций в параллельные циклы ряда языков и библиотек параллельного программирования. В соответствии с алгеброй преобразований функционально-потоковой модели параллельных вычислений выполнение функции над параллельным списком преобразуется во множество параллельных операций, в которых данная функция одновременно применяется ко всем элементам этого параллельного списка:

$$[x_1, x_2, ...x_n] : f -> [x_1 : f, x_2 : f, ...x_n : f].$$

Функция выполняется над каждым элементом, который может являться как атомом, так и списком данных, независимо. Возникновение дублирующих вычислений можно рассмотреть на следующем примере. Пусть имеется список данных $XList = (x_1,\ x_2,\ ...\ x_n)$, каждый элемент которого необходимо домножить на некоторое значение Y. Для выполнения такой операции приведенным выше способом необходимо сформировать параллельный список из пар (x_i, Y) и применить к нему операцию умножения. Подобная ситуация возникает достаточно часто. При этом на месте операции умножения может стоять любая другая функция, например, *foo*. Обобщенный фрагмент кода в этом случае будет выглядеть следующим образом:

```
Len << XList:|;
YList << (Y, Len):dup;
(XList, YList):#:[]:foo;
```

В данном случае определяется длина *Len* списка *XList*, элементы которого необходимо обработать. Значение *Len* используется при выполнении операции дублирования (*dup*), формирующей список данных *YList*, состоящий из *Len* дубликатов *Y*. Списки *XList* и *YList* помещаются в список, к которому применяется операция транспонирования ($\#$), в результате чего формируется искомый список пар. Он преобразуется в параллельный список оператором *[ ]*, к которому и применяется целевая функция *foo*. Все элементы списка данных *YList* имеют одинаковое значение Y, это значение будет являться вторым аргументом функции *foo*. Таким образом, вычисления, зависящие от Y и констант, будут являться инвариантом и могут быть оптимизированы.

Для предлагаемого метода оптимизации ключевое значение имеет то, что часть данных дублируется (выполняется операция *dup*). Дублируемые данные являются неизменяемыми параметрами. Инвариантом в данном случае являются вычисления

в функции *foo*, зависящие только от констант, неизменяемых параметров и других инвариантов. Именно этот инвариант и может быть перемещен из функции *foo* в вызывающую функцию.

Для выявления неизменяемых параметров в функции, реализующей вычисления над параллельным списком, оптимизатор выполняет поиск в ГПД фрагментов, общий вид которых показан на рис. 3. При этом *YList* отражает дублированные данные, в общем случае таких списков может быть несколько.

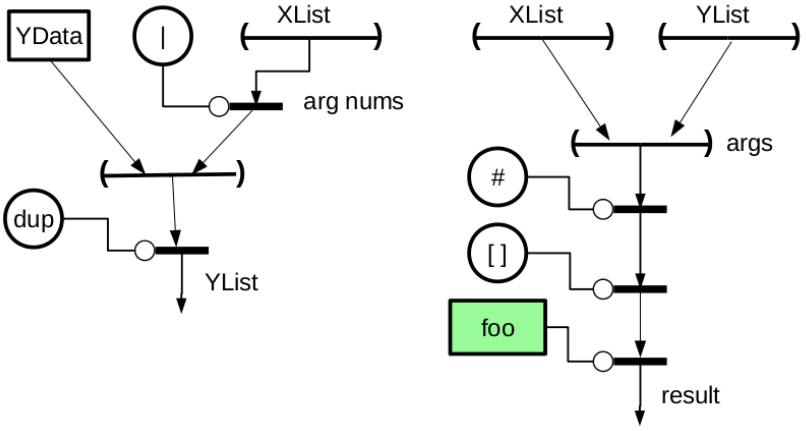

Рис. 3. Фрагмент графа, задающий повторяющиеся вычисления на списках
Fig. 3 A fragment of the graph that specifies repeating computations on lists

Поиск инвариантов внутри функции *foo* выполняется аналогично тому, как это описано для рекурсивных функций, отличается лишь способ определения неизменяемых аргументов (*ConstArgs*). Вычисление инварианта переносится в вызывающую функцию, результат вычисления передается в *foo* в качестве дополнительного аргумента.

В качестве примера рассмотрим функцию вращения трехмерной фигуры, заданной списком точек, вокруг оси. Функция принимает два аргумента — фигуру (*Figure*) и значение угла в радианах (*Alpha_rad*), на который требуется выполнить поворот. Для поворота точки используется функция *x_rotate*, для ее применения ко всем точкам определяется количество точек в фигуре (*PointsCount*), создается список из *PointsCount* углов (*Angles*). Для поворота точки матрица-строка, составленная из координат точки, умножается на матрицу вокруг соответствующей оси:

```
figure_rotate << funcdef X {
  Figure << X:1;
  Alpha_rad << X:2;

  PointsCount << Figure:|;
  Angles << (Alpha_rad, PointsCount):dup;

  return << (Figure, Angles):#:[]:x_rotate;
}
```

```
// вращение точки X в трехмерном пространстве
// вокруг оси абсцисс на Alpha_rad радиан
x_rotate << funcdef X {
  Point3d << X:1;
  Alpha_rad << X:2;

  cosA << Alpha_rad:cos;
  sinA << Alpha_rad:sin;

  rotMartix << (
    (1, 0, 0),
      (0, cosA, sinA:-),
    (0, sinA, cosA)
  );

  return << ((Point3d), rotMartix):matrix_mul;
}
```

При оптимизации функции вращения фигуры вокруг оси формируется граф, эквивалентный следующему фрагменту кода:

```
1 figure_rotate_inv_opt << funcdef X {
2   Figure << X:1;
3   Alpha_rad << X:2;
4
5   PointsCount << Figure:|;
6   Angles << (Alpha_rad, PointsCount):dup;
7
8   cosA << Alpha_rad:cos;
9   sinA << Alpha_rad:sin;
10
11  rotMartix << (
12     (1, 0, 0),
13     (0, cosA, sinA:-),
14     (0, sinA, cosA)
15   );
16
17   rotMatrices << (rotMartix, PointsCount):dup;
18
19   return << (Figure, Angles, rotMatrices):#:[]:x_rotate_inv_opt;
20 }
21
22 x_rotate_inv_opt << funcdef X {
23   Point3d << X:1;
24   Alpha_rad << X:2;
25   rotMartix << X:3;
```

```
26
27   return << ((Point3d), rotMartix):matrix_mul;
28 }
```

Инвариантом являлись вычисления синуса, косинуса угла и операции формирования матрицы вращения. В оптимизированном коде все вычисления производятся единственный раз вне зависимости от числа точек, но полученная матрица дублируется (список *rotMatrices*), так как они вынесены в функцию *figure_ rotate_ inv_ opt* (строки *5–17*).

Таким образом, если функция F содержит вызов функции *foo* над параллельным списком и имеет место инвариант, то оптимизация может выполняться по следующему алгоритму.

1. В оптимизируемой функции F необходимо найти вызов функции, на вход которой подан список, одним или несколькими из элементов которого является результат операции *dup*.

2. Сформировать список *ConstArgNums* номеров неизменяемых аргументов функции F.

3. В функции *func* найти и сохранить в списке *Invs* инварианты, с учетом того, что аргументы с номерами из списка *ConstArgNums* являются константами. Узел *Inv* принадлежит множеству *Invs* если он:

   - является актором;
   - находится в нулевом задержанном списке;
   - зависит по данным только от констант, неизменяемых аргументов (список *ConstArgNums*) или других инвариантов;
   - не является операцией группировки в параллельный список.

4. Переместить инварианты из функции *func* в функцию F.

5. Найти использования перемещенных инвариантов в функции *func*, сформировать список *UsingInvs* из узлов, значений которых не хватает в функции *func*. Для рассмотренного выше примера инвариантом будет являться, например, операция группировки в список *(0, cosA, sinA:-)*, однако напрямую это значение в функции *x_ rotate_ inv_ opt* не используется, поэтому в список *UsingInvs* он не добавляется.

6. В функцию F добавить результаты вычисления узлов множества *UsingInvs* в качестве параметров функции *func*. Для этого выполняется дублирование встроенной функцией *dup*. На примере рассматриваемой функции вращения добавляем в *figure_ rotate_ inv_ opt*:

   ```
   rotMatrices << (rotMartix, PointsCount):dup;
   ```

7. В функцию *func* добавить код получения вычисленных значений инвариантов из списка-аргумента функции. В частности для функции вращения добавим в *x_ rotate_ inv_ opt*:

```
rotMartix << X:3;
```

3. Заключение

Показана возможность проведения преобразований, эквивалентных оптимизации инварианта цикла, в программах языка Пифагор для двух случаев: рекурсивной функции и параллельных списков. Оптимизация инварианта в рекурсивной функции может быть применима к другим языкам программирования, в то время как оптимизация для параллельных списков основывается на специфических языковых конструкциях и может использоваться только в языке Пифагор. В обоих вариантах оптимизации рассматривается функция, тело которой будет выполнено многократно. В этой функции выполняется поиск фрагментов, результат вычисления которых будет одинаковым для всех обращений к функции. Эти фрагменты перемещаются из функции и вычисляются единожды (и передаются в функцию в виде дополнительных аргументов) — следовательно, преобразование сокращает объем вычислений. Представленные методы оптимизации проводятся после трансляции функций, обеспечивая тем самым архитектурно-независимый анализ разработанного кода. Все последующие трансформации функционально-потоковых параллельных программ, включая верификацию, тестирование, отладку, а также преобразование в архитектурно-зависимые представления могут проводиться после предложенных методов оптимизации.

Список литературы / References

[1] Ахо А. В., Лам М. С., Сети Р., Ульман Дж. Д., *Компиляторы: принципы, технологии и инструментарий*, 2-е изд., Вильямс, М., 2008, 1184 с.; In English: Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd Edition, Addison Wesley, 2006, 1000 pp.

[2] Дортман П. А., "Подходы к оптимизации программ в системе SFP", *Программные средства и математические основы информатики*, Новосибирск, 2004, 43–49; [Dortman P. A., "Program optimizacion in SFP", *Software tools and mathematical foundations of informatics*, Novosibirsk, 2004, 43–49, (in Russian).]

[3] Легалов А. И., Матковский И. В., Кропачева М. С., Удалова Ю. В., Васильев В. М., "Технологические аспекты создания, преобразования и выполнения функционально-потоковых параллельных программ", *Научный сервис в сети Интернет: все грани параллелизма: Труды Международной суперкомпьютерной конференции*, Изд-во МГУ, М., 2013, 443–447; [Legalov A. I., Matkovsky I. V., Kropacheva M. S., Udalova Y. V., Vasiliev V. M., "Tekhnologicheskie aspekty sozdaniya, preobrazovaniya i vypolneniya funkcionalno-potokovyh parallelnyh programm", *Nauchnyy servis v seti Internet: vse grani parallelizma: Trudy Mezhdunarodnoy superkompyuternoy konferentsii*, Izd-vo MGU, M., 2013, 443–447, (in Russian).]

[4] Легалов А. И., Васильев В. С., Матковский И. В., Ушакова М. С., "Инструментальная поддержка создания и трансформации функционально-потоковых параллельных программ", *Труды Института системного программирования РАН*, **29**:5 (2017), 165–184; [Legalov A. I., Vasilyev V. S., Matkovskii I. V., Ushakova M. S., "Support tools for

creation and transformation of functional-dataflow parallel programs", *Proceedings of ISP RAS*, **29**:5 (2017), 165–184, (in Russian).]

[5] Легалов А. И., "Функциональный язык для создания архитектурно-независимых параллельных программ", *Вычислительные технологии*, **10**:1 (2005), 71–89; [Legalov A. I., "Functional language for creating of architectural independent parallel programs", *Computational Technologies*, **10**:1 (2005), 71–89, (in Russian).]

Vasilev V. S., Legalov A. I., "Loop-invariant Optimization in the Pifagor Language", *Modeling and Analysis of Information Systems*, **25**:4 (2018), 347–357.

DOI: 10.18255/1818-1015-2018-4-347-357

Abstract. The paper considers methods of program transformation equivalent to optimizing the cycle invariant, applied to the functional data-flow model implemented in the Pifagor programming language. Optimization of the cycle invariant in imperative programming languages is reduced to a displacement from the cycle of computations that do not depend on variables that are changes in the loop. A feature of the functional data flow parallel programming language Pifagor is the absence of explicitly specified cyclic computations (the loop operator). However, recurring calculations in this language can be specified recursively or by applying specific language constructs (parallel lists). Both mechanisms provide the possibility of parallel execution. In the case of optimizing a recursive function, repeated calculations are carried out into an auxiliary function, the main function performing only the calculation of the invariant. When optimizing the invariant in computations over parallel lists, the calculation of the invariant moves from the function that executes over the list items to the function containing the call. The paper provides a definition of "invariant" applied to the Pifagor language, algorithms for its optimization, and examples of program source codes, their graph representations (the program dependence graph) before and after optimization. The algorithm shown for computations over parallel lists is applicable only to the Pifagor language, because it rests upon specific data structures and the computational model of this language. However, the algorithm for transforming recursive functions may be applied to other programming languages.

Keywords: data driven functional parallel programming, Pifagor programming language, code optimization, loop optimization, invariant optimization, program dependence graph

On the authors:
Vladimir S. Vasilev, orcid.org/0000-0002-3340-6678, graduate student,
Siberian Federal University, Institute of Space and Information Technology,
26 Kirenskogo str., Krasnoyarsk 660074, Russia, e-mail: ksv@akadem.ru

Alexander I. Legalov, orcid.org/0000-0002-5487-0699, doctor of science,
Siberian Federal University, Institute of Space and Information Technology,
26 Kirenskogo str., Krasnoyarsk 660074, Russia, e-mail: legalov@mail.ru

Acknowledgments:
The research is supported by RFBR (research project No 17-07-00288).

Моделирование и анализ информационных систем. Т. 25, №4 (2018), с. 358–381
Modeling and Analysis of Information Systems. Vol. 25, No 4 (2018), pp. 358–381

Верификация программ

DOI: 10.18255/1818-1015-2018-4-358-381

УДК 004.052.42

Верификация программ со взаимной рекурсией на языке Пифагор

Ушакова М. С., Легалов А. И.

получена 23 марта 2018

Аннотация. В работе рассматривается верификация программ со взаимной рекурсией для языка функционально-потокового параллельного программирования Пифагор. В языке используется модель представления программы в виде графа потока данных (информационного графа), в котором нет дополнительных управляющих связей, а присутствуют только информационные зависимости. Это позволяет упростить процесс верификации, так как не требует анализа возникающих в традиционных архитектурах дополнительных ресурсных конфликтов.

Доказательство корректности программы опирается на удаление взаимных рекурсий посредством преобразования программы. Универсальным способом удаления взаимной рекурсии произвольного количества функций является построение универсальной рекурсивной функции, которая выполняет работу всех исходных функций и принимает, кроме аргумента выполняемой функции, натуральное число, являющееся номером выполняемой функции. В ряде случаев, когда присутствует косвенная рекурсия, можно использовать более простой способ преобразования — объединение кода функций, при котором происходит объединение тел вызывающих друг друга функций.

Для преобразования произвольной рекурсии в прямую предлагается построение графа всех связанных функций и последующая трансформация данного графа путём удаления функций, не связанных с рассматриваемой, объединения косвенно рекурсивных функций и построения универсальной рекурсивной функции. Доказывается, что изменение функции на языке Пифагор при объединении кода и построении универсальной рекурсивной функции не влияет на корректность исходной программы. Приводится пример доказательства частичной корректности программы на языке Пифагор, осуществляющей синтаксический разбор простого арифметического выражения. После построения графа всех связанных функций рассматриваются два способа доказательства: с использованием объединения кода функций и с построением универсальной рекурсивной функции.

Ключевые слова: функционально-потоковое параллельное программирование, язык программирования Пифагор, корректность рекурсий, удаление взаимной рекурсии, универсальная рекурсивная функция

Для цитирования: Ушакова М. С., Легалов А. И., "Верификация программ со взаимной рекурсией на языке Пифагор", *Моделирование и анализ информационных систем*, **25**:4 (2018), 358–381.

Об авторах: Ушакова Мария Сергеевна, orcid.org/0000-0003-4234-2714, аспирант,
Сибирский федеральный университет, Институт космических и информационных технологий
ул. Академика Киренского, 26, г. Красноярск, 660074 Россия, e-mail: ksv@akadem.ru

Легалов Александр Иванович, orcid.org/0000-0002-5487-0699, д-р техн. наук, профессор,
Сибирский федеральный университет, Институт космических и информационных технологий
ул. Академика Киренского, 26, г. Красноярск, 660074 Россия, e-mail: legalov@mail.ru

Благодарности:
Исследование выполняется при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00288.

Введение

В связи с возрастающими требованиями к надежности программного обеспечения, наряду с традиционными методами тестирования, всё чаще стали использоваться методы формальной верификации программ. Формальная верификация — это доказательство корректности программы, которое заключается в установлении соответствия между программой и ее спецификацией, описывающей цель разработки [1]. Методы формальной верификации позволяют доказать отсутствие ошибок в программе, в то время как тестирование лишь выявляет ошибки, но не даёт гарантии их отсутствия.

Распараллеливание программ позволяет существенно увеличить их производительность на современных вычислительных системах. Однако параллелизм приводит к значительному усложнению разработки и особенно отладки. В основном программы пишутся на императивных языках программирования. По сравнению с последовательными, параллельные императивные программы могут содержать новые виды ошибок, которые трудно выявить при тестировании. Также резко увеличивается сложность формальной верификации параллельных императивных программ.

Вместе с тем всё большую популярность приобретает функциональное программирование, которое ориентируется на отношение между данными. В рамках этого направления формальные методы верификации развиваются достаточно интенсивно. Основополагающей работой в данной области является работа Бойера и Мура [2]. В системе NQTHM они реализовали метод автоматизированного доказательства функций, написанных на языке LISP. Другой пример — доказательство утверждений для программ на Haskell [3].

Одним из функциональных языков является язык функционально-потокового параллельного программирования Пифагор [5]. Его специфика — богатый набор операторов, обеспечивающий описание параллелизма с сохранением представления программы в виде зависимости функций и управлением по готовности данных. Это позволяет изначально создавать программы с максимальным параллелизмом за счёт его формирования на уровне операций. Данный подход также обеспечивает написание программы без учёта ресурсных ограничений, что позволяет упростить процесс верификации, так как не требует анализа возникающих в традиционных архитектурах дополнительных ресурсных конфликтов. После верификации возможны дальнейшие преобразования исходных программ в программы для реальных архитектур параллельных вычислительных систем путём «сжатия» параллелизма с учётом ограниченных ресурсов.

Для функционально-потоковых параллельных программ предложен метод формальной верификации, позволяющий доказывать корректность программ [6], а также разрабатывается инструментальное средство для поддержки этого процесса [7]. Для дальнейшего развития данной системы требуется включать дополнительные алгоритмы, расширяющие функциональность и упрощающие работу пользователя системы. Одной из задач, которую может выполнить система, является устранение взаимной рекурсии нескольких функций (например, функция f вызывает функцию g, а функция g вызывает функцию f). Существует два основных способа решения данной проблемы [8]. Первый способ — совместное доказательство корректности всех функций во взаимной рекурсии, которое обычно требует проведения доказа-

тельства с помощью одновременной индукции (simultaneous induction) [9]. Второй способ — удаление взаимной рекурсии посредством преобразования программы, в результате которого получается одна рекурсивная функция [10]. По сути, это два одинаковых метода, которые различаются только внешним представлением доказательства для пользователя. Недостаток первого способа — необходимость доказывать несколько утверждений одновременно, а недостаток второго способа — результирующая функция может получиться достаточно сложной.

В работе рассматривается второй способ решения проблемы взаимной рекурсии для программ на языке Пифагор. Основная причина выбора состоит в том, что этот способ делает доказательство более наглядным, так как не требуется одновременно доказывать утверждения для каждой из рекурсивных функций. В связи с тем, что проблема доказательства корректности программы является неразрешимой, то есть не существует универсального алгоритма решения этой проблемы, то невозможно сделать процесс доказательства полностью автоматическим. Поэтому пользователь должен иметь представление о теоремах, на которых основана работа алгоритма, и должен принимать участие в доказательстве, если система не может выполнить его автоматически.

Помимо доказательства корректности, необходимо доказывать завершение программы. Эта задача также является неразрешимой. Доказательство завершения функциональной программы сводится к построению порядка полной фундированности (порядок $\succ$ является отношением полной фундированности, если не существует бесконечной убывающей цепи $x_1 \succ x_2 \succ \ldots$) на множестве допустимых аргументов функции и проверке того, что аргумент каждого рекурсивного вызова меньше, чем входной аргумент [11, 12]. При наличии взаимной рекурсии необходим свой порядок для аргументов каждой из функций. Если провести преобразование и устранить взаимную рекурсию, то достаточно построить один порядок на множестве аргументов результирующей функции.

Данный метод может быть реализован в автоматическом режиме и интегрирован в инструментальное средство доказательства корректности программ на языке Пифагор.

1. Определения

В работе термин «функция» используется в нескольких смыслах. В математическом смысле — это однозначное отображение множества допустимых значений аргумента на множество значений функции. Когда речь идёт о программной реализации алгоритма, то термин «функция» или «функция на языке Пифагор» используется в рамках понимания функции в программировании — как именованного фрагмента кода на языке программирования, возвращающего значение по своему имени.

Под рекурсией понимается такой способ организации обработки данных, при котором функция может вызвать одну или несколько своих копий непосредственно или с помощью других функций [13]

Рекурсия может быть прямой или косвенной. *Прямая рекурсия* — это рекурсивная функция, в теле которой присутствует вызов самой себя. *Косвенная рекурсия* — это рекурсивная функция, в теле которой нет вызовов самой себя, но эта функ-

ция вызывается вновь через цепочку вызовов других функций [14]. Введем понятие *смешанной рекурсии* как функции, в которой присутствует прямой и косвенный рекурсивный вызов.

Также используем термины прямая и косвенная связь для разных функций. Функция A *прямо связана* с функцией B, если B вызывается непосредственно в теле A. Функция A *косвенно связана* с B, если вызов B опосредован вызовами других функций. A и B — *связанные* функции, если они связаны прямо или косвенно. Если функция A не связана с функцией B ни прямо, ни косвенно, то назовём A — *независимой* от B функцией.

Пусть A — рассматриваемая рекурсивная функция. Обозначим через $\Omega(A)$ *множество всех связанных функций* — функций, которые вызываются из A, как непосредственно в её теле, так и через цепочку вызовов других функций. Саму функцию A также отнесём к $\Omega(A)$. Для каждой функции в $\Omega(A)$ укажем её имя и после него в круглых скобках — список вызываемых непосредственно в её теле рекурсивных функций. Например, запись вида $\Omega(A) = \{A(B,D), B(A,B,C), C(A), D()\}$ означает, что рассматриваемая функция A является косвенной рекурсией, в её теле присутствует вызов функции B и D; B — смешанная рекурсия, вызывающая себя и функции A, C; C — косвенная рекурсия, вызывающая функцию A; D — нерекурсивная функция, независимая от других функций. Множество связанных функций $\Omega(A)$ можно изобразить в виде *дерева возможных вызовов*. Это дерево, корень которого помечен именем рассматриваемой функции A; с каждым узлом, помеченным именем функции X, связаны дочерние узлы, соответствующие вызываемым в теле X функциям. Пример такого дерева для множества $\Omega(A) = \{A(B,D), B(A,B,C), C(A), D()\}$ приведён на рис. 1a.

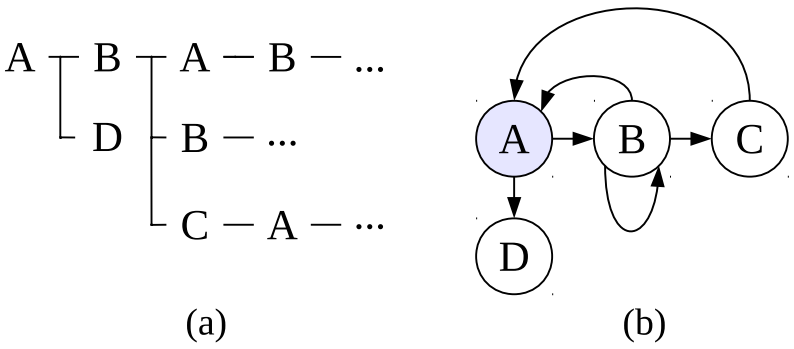

Рис. 1. Различные варианты представления множества связанных функций $\Omega(A) = \{A(B,D), B(A,B,C), C(A), D()\}$; (a) — дерево возможных вызовов функций; (b) — граф всех связанных функций, серым отмечена вершина с именем рассматриваемой функции

Fig. 1. Different variants of representation of the set of connected functions $\Omega(A) = \{A(B,D), B(A,B,C), C(A), D()\}$; (a) — the tree of possible function calls; (b) — the graph of all connected functions, the node with the name of the considered function is marked with gray

Бесконечное дерево возможных вызовов функций можно свернуть в *граф всех связанных функций*. Это ориентированный граф, вершины которого помечены именами функций $\Omega(A)$, а дуги направлены из вызывающей функции в вызываемую. Пример такого графа приведён на рис. 1b.

Функции A и B *взаимно рекурсивные*, если функция A связана с B и функция B связана с A. Обозначим отношение взаимной рекурсии $\leftrightarrow$. Если считать, что отношение взаимной рекурсии рефлексивно ($A \leftrightarrow A$), то отношение $\leftrightarrow$ будет отношением эквивалентности, которое разделяет множество всех связанных функций на непересекающиеся классы. На графе всех связанных функций эти классы будут соответствовать компонентам сильной связности [10].

2. Преобразование произвольной рекурсии в прямую

2.1. Универсальная рекурсивная функция

Одним из способов сведения произвольной рекурсии к прямой является построение универсальной рекурсивной функции [15]. Пусть $\mathfrak{F} = \{f_1, f_2, \ldots, f_k\}$ — множество функций $f_i : M \to M$, $i = 1, 2, \ldots, k$, M — произвольное множество. Каждой функции f_i приписано число $i \in \mathbb{N}$, называемое номером функции. Функция $F : \mathbb{N} \times M \to M$ называется *универсальной рекурсивной функцией* (УРФ) для множества $\mathfrak{F}$, если

$$F(n, x) = \begin{cases} f_1(x), & n = 1; \\ \ldots \\ f_k(x), & n = k. \end{cases}$$

Например, рассмотрим две связанные функции A и B. В общем случае функция A является смешанной рекурсией, вызывает функцию B, которая тоже является смешанной рекурсией. Это случай взаимной рекурсии двух функций. Множество связанных рекурсивных функций $\Omega(A)$ имеет вид: $\{A(A, B), B(A, B)\}$. Дерево возможных вызовов функций приведено в левой части рис. 2.

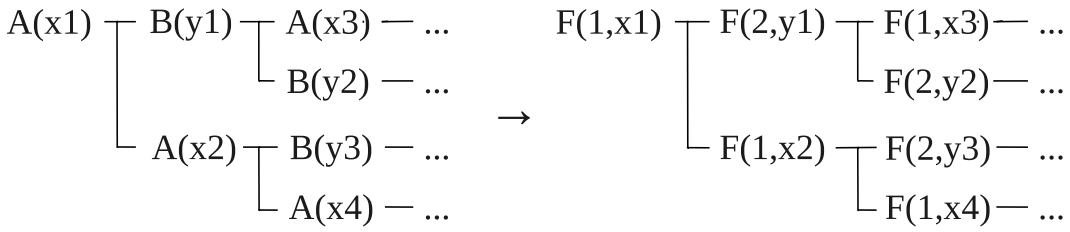

Рис. 2. Схема удаления взаимной рекурсии двух смешанно рекурсивных функций A и B с помощью универсальной рекурсивной функции F; после имени каждой функции в скобках указаны её аргументы

Fig. 2. The scheme of elimination of mutual recursion of functions A and B with the help of the universal recursive function F; after the name of each function its arguments are given in brackets

Устраним взаимную рекурсию с помощью универсальной рекурсивной функции. Зададим универсальную рекурсивную функцию $F(n, x)$, у которой n — номер рекурсивной функции, объединяемой в УРФ, а x — аргумент для функции с номером n; n будет принимать значение 1, когда надо выполнить код функции A, и 2, если

требуется выполнить код функции B. Дерево возможных вызовов универсальной рекурсивной функции F приведено в правой части рис. 2. Построение УРФ для большего числа функций аналогично.

2.2. Объединение функций

В простых случаях косвенную рекурсию можно свести к прямой рекурсии с помощью *объединения* тел связанных функций в одну функцию. Подобный метод используется в [10] для логических программ и называется развёрткой (unfolding).

Поясним данный способ преобразования на примере двух функций. Пусть дана функция A, которая, в общем случае, является смешанной рекурсией. В ней присутствует вызов функции B, которая является косвенной рекурсией и прямо связана с функцией A. То есть в функции B нет вызова самой себя, но присутствует вызов функции A. В этом случае косвенную рекурсию можно свести к прямой рекурсии объединением тел функций A и B в одну функцию AB. Таким образом, код функции B «встраивается» в код функции A в месте вызова B. Изменение графа двух связанных функций описанного случая приведено на рис. 3a. Исходный граф связанных функций для $\Omega(A) = \{A(A, B), B(A)\}$, после объединения кода функций A и B в функцию AB получаем $\Omega(AB) = \{AB(AB)\}$.

Если рассматриваемое множество связанных функций состоит из более чем двух функций, то объединение кода функций не приведёт к дублированию кода в том случае, если только одна функция связана с рассматриваемой косвенной рекурсией B. Это эквивалентно тому, что на графе всех связанных функций вершина B имеет только одну входную дугу. В остальных случаях косвенную рекурсию лучше устранять с помощью универсальной рекурсивной функции, описанной ранее.

Пример объединения функций для случая трёх связанных функций приведён на рис. 3b, граф связанных функций для исходного $\Omega(A) = \{A(B), B(A, B, C), C(A)\}$. Функция C является косвенной рекурсией и имеет одну входную дугу на графе связанных функций. Поэтому её код может быть «встроен» в функцию B, узел которой является смежным с узлом C по его единственной входной дуге. Полученной в результате объединения функции присваивается имя BC, изменённое $\Omega(A) = \{A(BC), BC(A, BC)\}$. Для дальнейшего преобразования полученной рекурсии в прямую необходимо построить УРФ.

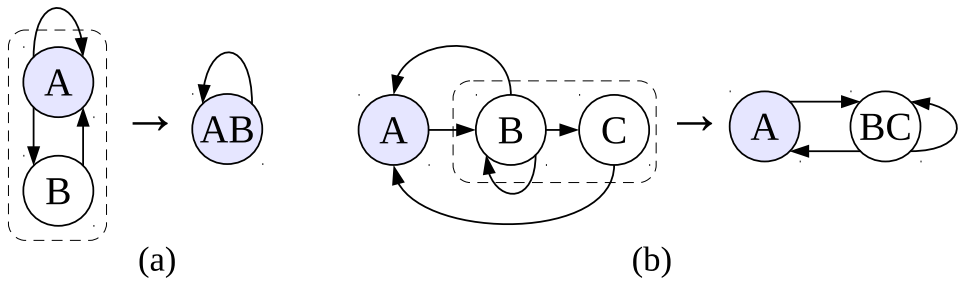

Рис. 3. Удаление косвенной рекурсии с помощью объединения кода функций

Fig. 3. The elimination of indirect recursion by merging of functions code

2.3. Алгоритм преобразования произвольной рекурсии в прямую

Резюмируя вышесказанное, можно предложить следующий алгоритм преобразования произвольной рекурсии в прямую.

Дана произвольная рекурсивная функция A со множеством связанных функций $\Omega(A)$. Требуется преобразовать её в прямую рекурсию.

1. На основе $\Omega(A)$ строим граф всех связанных функций $G(A)$.

2. В $\Omega(A)$ находим все нерекурсивные функции. В $G(A)$ таким функциям будут соответствовать вершины, не входящие ни в один цикл. То есть для вершины X, помеченной нерекурсивной функцией, не существует пути, который привёл бы опять в вершину X. Все найденные нерекурсивные функции являются независимыми от A и удаляются из множества $\Omega(A)$. В результате чего получаем новое множество связанных функций $\Omega_1(A)$ с графом связанных функций $G_1(A)$.

3. Удаляем из $\Omega_1(A)$ все рекурсивные функции, не связанные с A. Для этого в графе $G_1(A)$ находим все узлы, для которых любой цикл, проходящий через этот узел, не проходит через вершину A. Все независимые от A рекурсивные функции удаляются из множества $\Omega_1(A)$. В результате получаем множество связанных функций $\Omega_2(A)$ с графом связанных функций $G_2(A)$.

4. В $G_2(A)$ находим все функции, имеющие только одну входную дугу. Код этих косвенно рекурсивных функций может быть объединён с кодом функции смежной по входной дуге рассматриваемого узла. После объединения получаем $\Omega_3(A)$.

5. Задаём универсальную рекурсивную функцию F для всех функций из множества $\Omega_3(A)$.

Полученная в результате функция F будет прямой рекурсией.

Замечание 1. При доказательстве корректности функции A вначале необходимо доказать корректность всех независимых от A нерекурсивных и рекурсивных функций, которые удаляются из множества всех связанных функций в пунктах 2 и 3.

Замечание 2. Если узел нерекурсивной функции, найденной в пункте 2, имеет одну входную дугу, то код этой функции можно объединить с кодом функции, смежной по входной дуге.

Замечание 3. Если пункт 4 пропустить (после пункта 3 сразу переходить к пункту 5), то все функции из $\Omega_2(A)$ войдут в универсальную рекурсивную функцию.

В качестве иллюстрации работы алгоритма на рис. 4а изображен граф всех связанных функций для функции A со множеством

$$\Omega(A) = \{A(A,B,C,D,E), B(A,C,H,I), C(B), D(J), E(F,G), F(E,F,G), \\ G(), H(C), I(A), J()\}.$$

Три узла в графе (D, J и G, выделенные пунктиром) не лежат ни в каком цикле. Следовательно, функции, которыми помечены эти узлы, являются нерекурсивными и удаляются из множества $\Omega(A)$. Полученное $\Omega_1(A)$ имеет вид

$$\{A(A,B,C,E), B(A,C,H,I), C(B), E(F), F(E,F), H(C), I(A)\}.$$

На следующем этапе удаляются все рекурсивные функции, независимые от рассматриваемой функции A. Из графа связанных функций $\Omega_1(A)$ (рис. 4b) видно, что не существует цикла, проходящего через узлы E или F и узел A. Значит, рекурсивные функции E и F не зависят от A и удаляются из $\Omega_1(A)$, получается $\Omega_2(A) = \{A(A,B,C), B(A,C,H,I), C(B), H(C), I(A)\}$ (рис. 4c). Два узла I и H в графе для $\Omega_2(A)$ имеют по одной входной дуге, поэтому их код может быть объединён со смежным по их входной дуге узлу. Для обеих вершин это узел B. Назовём функцию, полученную в результате объединения, B_1, тогда $\Omega_3(A) = \{A(A,B_1,C), B_1(A,C), C(B_1)\}$ (рис. 4d). Далее, для трёх оставшихся функций формируется УРФ.

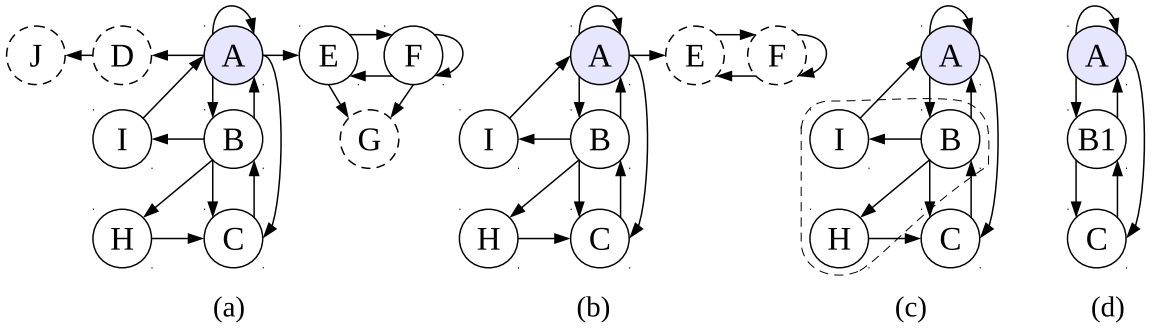

Рис. 4. Пример изменения графа связанных функций при преобразовании рекурсии A в прямую

Fig. 4. The example of modifications of the connected functions graph during transformations of the function A into the direct recursion

3. Преобразование рекурсивных функций на языке Пифагор

Язык программирования Пифагор хорошо подходит для проведения преобразования программы с удалением взаимной рекурсии. В языке присутствует только один оператор, применяющий функцию к аргументу, — оператор интерпретации. У оператора два входа, на которые поступают функция и аргумент функции. Оператор интерпретации имеет постфиксную и префиксную формы, которые обозначаются знаками «:» и «^» соответственно. Далее в тексте используется только постфиксная форма оператора. Например, X:F — применение функции F к аргументу X. Аргумент всегда один, но он может быть списком данных с достаточно сложной структурой, которая может интерпретироваться как передача нескольких аргументов. Таким образом, у каждой функции во взаимной рекурсии один аргумент, и после преобразования в результирующей УРФ также будет один аргумент.

Рассмотрим удаление различных видов рекурсий из программ на языке Пифагор.

3.1. Объединение кода функций

В начале рассмотрим случай косвенной рекурсии. Дано две функции A и B на языке Пифагор, с исходным кодом:

```
A << funcdef x
{
    (
        {x:g0},
        {x:g1:A},
        {x:g2:B}
    ):
    [(c0, c1, c2):?]:.:p >> return;

}
```

```
B << funcdef y
{
    (
        {y:e0},
        {y:e1:A}
    ):
    [(d0, d1):?]:.:q >> return;

}
```

В данном коде g_i, c_i, $i = 1, 2, 3$ и e_j, d_j, $j = 0, 1$, p, q — некоторые функции, независимые от функций A и B. Множество $\Omega(A) = \{A(A, B), B(A)\}$, соответствующее дерево всех связанных функций приведено на рис. 3а.

Для удаления косвенной рекурсии объединим код функций A и B в функцию AB, как показано ниже:

```
AB << funcdef x
{
    (
        {x:g0},
        {x:g1:AB},
        {
            (
                {x:g2:e0},
                {x:g2:e1:AB}
            ):
            [(d0, d1):?]:.:q
        }
    ):
    [(c0, c1, c2):?]:.:p >> return;

}
```

Покажем, что объединение кода не изменяет корректность исходной программы. Пусть условие корректности рассматриваемой функции A задано в виде тройки:

$$\boxed{P_A}\ \mathrm{A}\ \boxed{Q_A}.$$

При объединении кода получаем функцию AB с условием корректности

$$\boxed{P_{AB}}\ \text{AB}\ \boxed{Q_{AB}},$$

где $P_{AB} = P_A$, $Q_{AB} = Q_A$. Предположим, что корректность AB доказана, требуется показать, что A корректна. Очевидно, что из истинности $(P_{AB} \Rightarrow Q_{AB})$ следует истинность формулы $(P_A \Rightarrow Q_A)$, значит, функция A корректна.

3.2. Построение универсальной рекурсивной функции

Далее рассмотрим удаление взаимной рекурсии. Возьмём схему рекурсии, представленную на рис. 2. В общем случае исходный код функций A и B на языке Пифагор будет следующим:

```
A << funcdef x
{
    (
        {x:g0},
        {x:g1:A},
        {x:g2:B}
    ):
    [(c0, c1, c2):?]:.:p >> return;
}
```

```
B << funcdef y
{
    (
        {y:e0},
        {y:e1:A},
        {y:e2:B}
    ):
    [(d0, d1, d2):?]:.:q >> return;
}
```

где g_i, c_i, e_i, d_i, $i = 1, 2, 3$, p, q — некоторые функции, независимые от функций A и B.

Для удаления взаимной рекурсии введём универсальную рекурсивную функцию F следующим образом:

```
F << funcdef nx
{
    n<<nx:1;
    x<<nx:2;
    (
        {
            (
                {x:g0},
                {(1,x:g1):F},
                {(2,x:g2):F}
            ):
            [(c0, c1, c2):?]:.:p
        },
        {
            (
                {x:e0},
                {(1,x:e1):F},
                {(2,x:e2):F}
            ):
            [(d0, d1, d2):?]:.:q >> return;
```

```
        }
    ):
    [
        ((n,1):=,(n,2):=):?
    ]:. >> return;
}
```

Полученная функция F является прямой рекурсией.

Покажем, что объединение функций в УРФ не изменяет корректность исходной программы. Пусть условие корректности функции A задано в виде тройки $\boxed{P_A}$ A $\boxed{Q_A}$, а функции B — $\boxed{P_B}$ B $\boxed{Q_B}$. Зададим для функции $F(n, x)$ следующее предусловие и постусловие:

$$P_F = ((n = 1) \wedge P_A) \vee ((n = 2) \wedge P_B),$$

$$Q_F = ((n = 1) \wedge Q_A) \vee ((n = 2) \wedge Q_B).$$

Тогда, при верном указании номера функции $n = 1$, получаем, что из корректности функции F будет следовать корректность исходной функции A:

$$(P_F \wedge (n = 1)) \Rightarrow P_A,$$

$$(Q_F \wedge (n = 1)) \Rightarrow Q_A.$$

4. Пример доказательства корректности рекурсивной функции на языке Пифагор

Доказательство корректности рекурсивной программы рассмотрим на примере решения задачи распознавания простого арифметического выражения, порождаемого в соответствии со следующим правилом в нотации Бэкуса–Наура:

```
<выражение> ::= x | +<выражение><выражение> | *<выражение><выражение>      (1)
```

где x — терминальный операнд. Для упрощения примера введём следующие ограничения: функция должна принимать на вход строку с правильно построенным выражением и возвращать пустую строку, никакой обработки ошибок не требуется.

Распознавание данного выражения определяется следующими функциями на языке Пифагор:

```
parse<<funcdef str  // Основная функция parse, принимающая строку str
                    // с арифметическим выражением
{   s1   << str:1;  // Сохранение первого символа строки str в s1
    case << ( (s1,'+'):=, // Сравнение s1 с тремя допустимыми
              (s1,'*'):=, // вариантами значений
              (s1,'x'):=
            ):?;          // Выбор номера пути вычисления и
                          // сохранение номера в case
    act  << ( {str:fp},   // Формирование списка act трёх различных путей
```

```
            {str:fm},   // вычислений, зависящих от значения s1
            {str:fn}
          );
    act:case:.   // Выбор нужного пути вычисления и активация вычисления
    >> return;   // Возврат результата вычислений
}
fp<<funcdef str1 // Функция fp для разбора сложения, принимающая str1
{    str1:-1:parse:parse // Из строки str1 удаляется первый символ '+'
     >> return;          // и с помощью двух последовательных вызовов
                         // функции parse удаляется 1-е и 2-е слагаемое
}
fm<<funcdef str2 // Функция fm для разбора умножения, принимающая str2
{    str2:-1:parse:parse >> return;   } // Аналогична fp
fn<<funcdef str3 // Функция fn для разбора формального символа x
{    str3:-1 >> return; // Из входной строки str2 удаляется один символ
}
```

Основная функция, которая получает строку с арифметическим выражением, имеет имя *parse*. Также имеется три вспомогательные функции, которые отвечают за разбор одного из видов выражений: fp разбирает применение функции сложения, fm — функции умножения, а fn — формального символа «x». Функция *parse* определяет тип выражения по первому символу входной строки и передает эту строку соответствующей вспомогательной функции. И, в качестве ответа, возвращает результат работы вспомогательной функции. Функции fp, fm разбирают получаемую строку, удаляя из неё первый символ операции, а затем дважды применяя к ней функцию *parse*. Первый вызов функции *parse* удаляет из строки первое слагаемое (множитель), второй вызов — второе слагаемое (множитель), оставшийся «хвост» строки возвращается в качестве ответа. Функция fn разбирает получаемую строку удалением из неё первого символа и возвращает оставшийся «хвост» строки. Таким образом, если входное выражение для функции *parse* не содержит ошибок, то функция вернёт пустую строку, иначе результат непредсказуем.

На естественном языке спецификация к функции *parse* будет следующей. Входным аргументом для функции является строка символов *str*, в которой можно выделить две подстроки str_1 и *tail* таких, что $str = str_1 \circ tail$, где «$\circ$» обозначает объединение (конкатенацию) двух строк. Строка str_1 удовлетворяет правилу (1), а строка *tail* содержит произвольные символы, в том числе она может быть пустой. В результате работы функция *parse* возвращает строку *tail*.

Запишем спецификацию функции *parse* на формальном языке с помощью предусловия и постусловия [6,16]. Для записи формул используем язык исчисления предикатов с равенством [17]. Будем придерживаться следующих соглашений по записи формул и терминологии.

1. Для любой переменной указываем её тип.

   Простой (конкретный) тип — это множество значений этого типа вместе с совокупностью операций и отношений, в которых эти значения могут участвовать [18]. Из простых типов, описанных в [17], используем $\mathbb{N}$ — множество

натуральных чисел, $\mathbb{Z}$ — множество целых чисел, *char* — множество символов, *bool* — множество логических констант $\{true, false\}$.

Родовой (составной) тип — это (частичная) функция, которая по конкретным типам строит новый тип. Если применить эту функцию к нескольким конкретным типам, то получим составной конкретный тип. Родовым типом является декартово произведение n-множеств, его обозначаем как $T_1 \times T_2 \times \cdots \times T_n$ (или кратко $\prod_{i=1}^{n} T_i$), где $T_i,\ i = 1, \ldots, n$ — конкретные типы. Элементы декартова произведения n множеств $\prod_{i=1}^{n} T_i$ — упорядоченные n-ки или векторы. Пишем, что вектор $(t_1, ..., t_n) \in \prod_{i=1}^{n} T_i$, если $t_i \in T_i,\ i = 1, \ldots, n$.

2. Списки данных языка Пифагор также являются векторами, для удобства тип, которому принадлежит список данных из n-элементов, обозначаем как $(datalist \prod_{i=1}^{n} T_i)$. Строка в языке Пифагор является частным случаем списка данных, содержащим символы типа *char*. Строки записываем как совокупность символов, заключенную в кавычки «“» и «”». Например, строка *str*=“123text” длины $n = 7$ будет иметь тип $(datalist \prod_{i=1}^{7} char)$.

3. Функция (в формуле) — однозначное отображение одного вектора длины n в вектор длины m. Если f — функциональный символ для обозначении функции, $(x_1, \ldots, x_n)$ — исходный вектор типа $\prod_{i=1}^{n} X_i$, $(y_1, \ldots, y_m)$ — результирующий вектор типа $\prod_{i=1}^{m} Y_i$, то запись $f(x_1, \ldots, x_n)$ используется для обозначения результирующего вектора $(y_1, \ldots, y_m)$.

4. Множество функций, отображающих вектор размера n в вектор размера m, является декартовым произведением $n + m$ множеств, обозначаем $(X \to Y)$, где $X = \prod_{i=1}^{n} X_i$, $Y = \prod_{i=1}^{m} Y_i$.

5. Для краткости формулы на языке логики заменяем функциями (от свободных переменных формулы), которые принимают значения в булевском множестве *bool*: если формула истинна для данного значения переменных, то функция принимает значение $true$.

6. При записи формул упускаем скобки после кванторов, которые применяются ко всему выражению, следующему после них.

Опишем множество функций, аргументом которых является целое число n_1, а результатом — строка длиной $(2n_1 - 1)$, удовлетворяющая условию (1):

$$\forall n_1 \in \mathbb{N}\ \exists f \in (\mathbb{N} \to (datalist \prod_{i=1}^{2n_1-1} char))\ \left(f(1) = \text{“x”}\right) \wedge$$
$$\wedge \left(f(2) = \text{“+xx”} \vee f(2) = \text{“*xx”}\right) \wedge$$
$$\wedge \left(\bigvee_{i=1}^{n_1-1} \left(f(n_1) = \text{“+”} \circ f(i) \circ f(n_1 - i)\right) \vee \left(f(n_1) = \text{“*”} \circ f(i) \circ f(n_1 - i)\right)\right), \quad (2)$$

где символ «$\circ$» соответствует операции конкатенации строк.

Обозначим (2) с помощью функции $F(f) \in \left(\mathbb{N} \to (datalist \prod_{i=1}^{2n_1-1} char)\right) \to bool$. Тогда предусловие P к функции $parse$ можно записать на языке логики следующим образом:

$$F(f) \wedge \left(\exists n \in \mathbb{N}\ \exists m \in \mathbb{Z}\ \left((m > 0) \wedge \forall str \in (datalist \prod_{i=1}^{2n-1+m} char)\right.\right.$$
$$sublist(str, 1, 2n-1) = f(n) \wedge \exists tail \in (datalist \prod_{i=1}^{m} char)$$
$$\left.\left. tail = sublist(str, 2n, 2n-1+m)\right)\right),$$

где $sublist(s, p, q)$ — функция, возвращающая подстроку строки s, начиная с позиции p до q; возвращает пустую строку, если $p = q$.

Предусловие P описывает структуру множества функций F, и требования к входному аргументу str: любой входной аргумент str является строкой длины $2n - 1 + m$, в которой можно выделить подстроку, длиной $2n - 1$, удовлетворяющую условию (1) (совпадает с $f(n)$ — результатом, возвращаемым одной из функций f множества F), оставшаяся часть строки $tail$ имеет длину m.

Постусловие Q функции $parse$ зададим следующей формулой:

$$return = tail,$$

где $return$ обозначает результат вычислений программы.

Для доказательства корректности косвенно рекурсивной функции $parse$ преобразуем её в прямую рекурсию. Множество всех связанных функций для $parse$ имеет вид:

$$\Omega(parse) = \{parse(fp, fm, fn), fp(parse), fm(parse), fn()\},$$

граф всех связанных функций приведён на рис. 5.

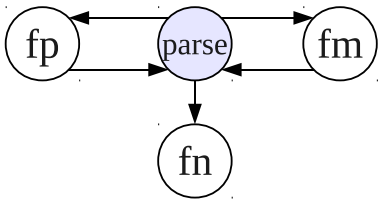

Рис. 5. Граф всех связанных функций для *parse*

Fig. 5. The graph of all connected functions for the function *parse*

4.1. Преобразование рекурсии *parse* в прямую при объединении кода

Связи функций из $\Omega(parse)$ допускают объединение кода функций для удаления косвенной рекурсии. В $\Omega(parse)$ функция fn является нерекурсивной, воспользуемся замечанием 2 алгоритма преобразования рекурсий из раздела 2.3. и объединим код функции fn с кодом *parse*. Полученную функцию обозначим $parsefn$,

$$\Omega(parsefn) = \{parsefn(fp, fm), fp(parsefn), fm(parsefn)\}.$$

В $\Omega(parsefn)$ отсутствуют независимые рекурсивные функции. Функции fp и fm имеют одну входную дугу, поэтому, согласно пункту 4 алгоритма преобразования рекурсий, объединим их код с кодом функции $parsefn$. Полученной функции дадим имя $parseUn$. Эта функция является прямой рекурсией и не требует задания УРФ. Ниже приведён исходный код $parseUn$:

```
parseUn<<funcdef str // Функция parseUn, принимающая строку str с
                     // арифметическим выражением
{   s1   << str:1;   // Сохранение первого символа строки str в s1
    case << ( (s1,'+'):=, // Сравнение s1 с тремя допустимыми
              (s1,'*'):=, //  вариантами значений
              (s1,'x'):=
            ):?;          // Выбор номера пути вычисления и
                          //сохранение номера в case
    act << (  // Формирование списка act трёх различных путей вычислений
            {str:-1:parseUn:parseUn }, //Результат объединения кода с fp
            {str:-1:parseUn:parseUn }, //Результат объединения кода с fm
            {str:-1}                   //Результат объединения кода с fn
          );
    act:case:.           // Выбор нужного пути и запуск вычисления
    >> return;           // Возврат результата вычислений
}
```

Докажем частичную корректность $parseUn$ с помощью метода, описанного в [6, 7]. Суть метода заключается в разметке дуг информационного графа программы [5] формулами, при использовании аксиом встроенных функций и теорем для функций

с ранее доказанной корректностью, а также преобразовании графа. В результате получается несколько полностью размеченных информационных графов с разметкой (ИГР), каждый из которых сворачивается в формулу. Если все, полученные формулы, тождественно истинны, то программа частично корректна.

Предусловие и постусловие *parseUn* совпадают с предусловием и постусловием *parse* (см. раздел 2.3.). Информационный граф *parseUn* приведён на рис. 6а. Для удобства все дуги графа пронумерованы. Считаем, что номер выходной дуги задаёт и номер оператора (вершины).

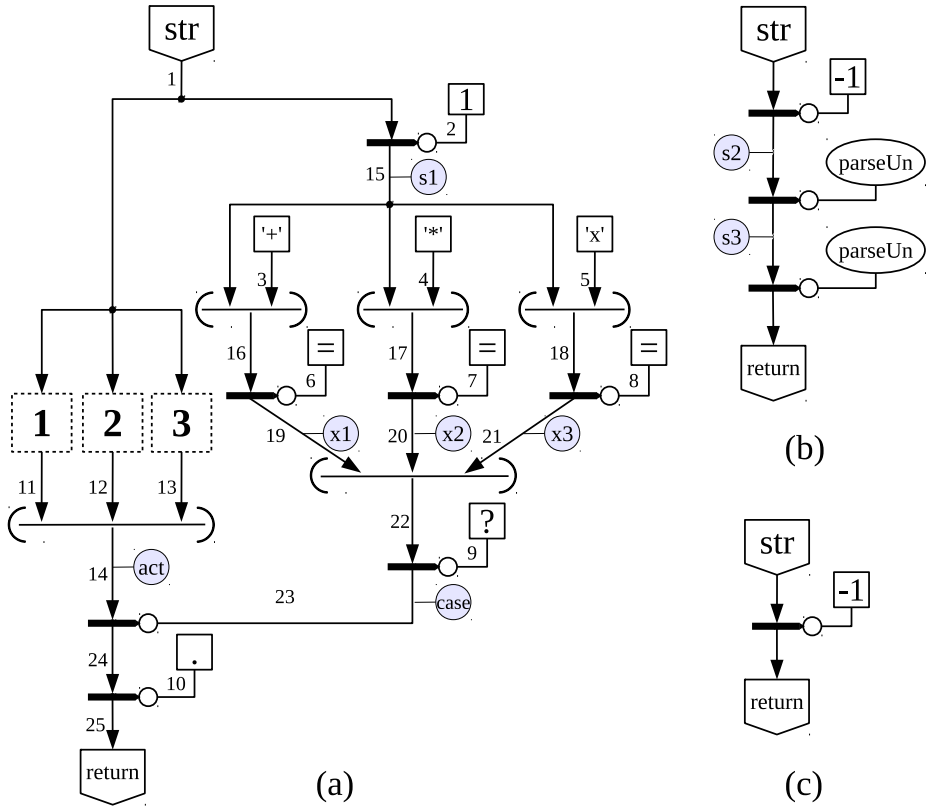

Рис. 6. Информационные графы функции *parseUn*, к некоторым дугам приписаны идентификаторы, обозначенные серыми кружками; (a) — исходный информационный граф функции *parseUn*, все дуги графа пронумерованы, задержанные списки обозначены пунктиром и пронумерованы; (b), (c) — информационные графы функции *parseUn* после расщепления и раскрытия задержанных списков

Fig. 6. Data flow graphs of the function *parseUn*, some arcs have identifiers that are denoted by gray circles; (a) — the initial data flow graph of the function *parseUn*, all arcs are enumerated, delay lists are marked with the dashed line and enumerated; (b), (c) — data flow graphs of the function *parseUn* after splitting and delay release

Дуги графа *parseUn* размечаются в порядке готовности данных: если все входные дуги узла размечены, можно размечать выходной узел. Узел 1 размечен предусловием, дуги 2–10 являются выходными дугами констант и получают автоматическую разметку, дуги 11–13 — выходные дуги задержанных списков, которые до снятия задержки являются константами. Дуга 14 с идентификатором *act* — выходная дуга списка данных также размечается автоматически. Дуга 15 с идентификатором

s_1 может быть размечена на основе аксиом функции выбора элемента из списка. В результате к дуге s_1 приписывается формула $(s_1 \in char \wedge s_1 = select(str, 1))$, где $select(x, i)$ — функция выбора i-го элемента из списка x. Дуги 16–18 являются выходными дугами списков данных и размечаются автоматически, после разметки дуги s_1. Далее, на основе аксиом для функции «=», можно разметить дуги 19–21. В результате к дуге x_1 припишутся две формулы:

$$(x_1 \in bool) \wedge (x_1 = true) \wedge (s_1 = \text{`}+\text{'}),$$

$$(x_1 \in false) \wedge (x_1 = false) \wedge \neg(s_1 = \text{`}+\text{'}),$$

где одинарные кавычки «'» используются для записи символьной константы типа $char$. К дугам x_2 и x_3 будут приписаны аналогичные формулы, в которых заменён идентификатор дуги и символьная константа на '$*$' и 'x' соответственно.

Дуга 22 размечается автоматически. К 23 дуге с идентификатором $case$, при разметке на основе аксиомы для функции «?», приписываются три формулы:

$$(case \in int) \wedge (case = 1) \wedge (x_1 = true) \wedge (x_2, x_3 = false),$$

$$(case \in int) \wedge (case = 2) \wedge (x_2 = true) \wedge (x_1, x_3 = false),$$

$$(case \in int) \wedge (case = 3) \wedge (x_3 = true) \wedge (x_1, x_2 = false).$$

Применение $case$ как функции к аргументу arg при выполнении оператора интерпретации 24 приведёт к расщеплению исходного графа на три, в каждом из которых произойдёт раскрытие задержанного списка при применении функции «.». Пусть номера полученных графов совпадают с номерами их задержанных списков. Первый и второй полученные графы одинаковые и приведены на рис. 6b, а третий — на рис. 6c.

Рассмотрим первый граф. Его предусловие P_1 будет следующим: $P \wedge (s_1 \in char) \wedge s_1 = select(str, 1) \wedge (s_1 = \text{`}+\text{'})$, а постусловие $Q_1 = Q$. То есть предусловие P_1 отличается от исходного предусловия P, тем, что в нём присутствуют формулы, которые описывают причины выбора данного пути вычисления: первый символ строки str равен символу '+'. Для разметки дуги s_2 используются аксиомы функции выбора элемента из списка, которая при отрицательном входном аргументе удаляет элемент из списка. В результате к дуге s_2 приписывается формула:

$$s_2 \in (datalist \prod_{i=0}^{2n-2+m} char) \wedge s_2 = sublist(str, 2, 2n - 1 + m).$$

Далее к результату s_2 применяется функция $parseUn$, то есть производится рекурсивный вызов. Для доказательства частичной корректности программы достаточно предположить, что все рекурсивные вызовы функции возвращают верный результат, если их аргумент удовлетворяет предусловию. Покажем, что s_2 удовлетворяет предусловию $parseUn\ P$. Так как строка str удовлетворяет предусловию и первый её символ $s_1 = \text{`}+\text{'}$ (следует из P_1), значит, $f(n)$ (из P) по правилу (2) будет иметь вид $f(n) = \text{“}+\text{”} \circ f(k) \circ f(n-k)$, для некоторого k. На языке логики это утверждение можно выразить следующей формулой:

$$\exists k \in \mathbb{N}\ (k < n) \wedge f(n) = \text{“}+\text{”} \circ f(k) \circ f(n-k).$$

А следовательно, при удалении первого символа из $str = f(n) \circ tail$, получим $s_2 = f(k) \circ f(n-k) \circ tail$. Таким образом, в начале строки s_2 стоит подстрока $f(k)$, удовлетворяющая (1), а «хвост» $tail_1 = f(n-k) \circ tail$ имеет длину $m_1 = 2(n-k) - 1 + m$. Тогда по индуктивному предположению s_3 — результат применения функции $parseUn$ к s_2, будет удовлетворять постусловию: $s_3 = tail_1$. И к дуге s_3 будет приписана формула:

$$\exists k \in \mathbb{N}\ (k < n) \wedge (\exists m_1 \in \mathbb{Z}\ (m_1 > 0) \wedge sublist(s_2, 1, 2k-1) = f(k) \wedge \\ \wedge s_3 = sublist(s_2, 2k, 2k-1+m_1))\,.$$

Таким образом, $s_3 = f(n-k) \circ tail$, а значит, тоже удовлетворяет предусловию функции $parseUn$. Тогда по индуктивному предположению имеем, что после применения $parseUn$ к s_3, $return = tail$. Постусловие Q всегда следует из этой формулы, следовательно, первый ИГР эквивалентен тождественно истинной формуле.

Второй граф (рис. 6b) полностью совпадает с первым. Отличие затрагивает только его предусловие P_2, в котором символ ‘+’, заменён на ‘$*$’. Остальные формулы разметки полностью совпадают с соответствующими формулами первого графа, поэтому второй ИГР также эквивалентен тождественно истинной формуле.

Рассмотрим третий граф (рис. 6c). Его предусловие P_3 имеет вид: $(P \wedge (s_1 \in char) \wedge s_1 = select(str, 1) \wedge (s_1 = \text{‘}x\text{’}))$, а постусловие $Q_3 = Q$. К входной строке str применяется функция удаления первого элемента, в результате к дуге $return$ приписывается формула $return = sublist(str, 2, 2n-1+m)$. Исходя из (2) и того, что $s_1 = \text{‘}x\text{’}$, $str = \text{‘}x\text{’} \circ tail$. Поэтому при удалении первого символа str получаем, что $return = tail$. Следовательно, третий ИГР эквивалентен тождественно истинной формуле. Из истинности всех трёх ИГР следует частичная корректность программы $parseUn$, а значит, функция $parse$ тоже частично корректна.

4.2. Преобразование рекурсии *parse* в прямую с помощью универсальной рекурсивной функции

Метод объединения кода функций применим в достаточно простых случаях, в сложных ситуациях необходимо использовать более сложный, но универсальный метод построения универсальной рекурсивной функции. Данный метод можно применить и для удаления взаимной рекурсии в функции $parse$. Для этого применим алгоритм преобразования произвольной рекурсии в прямую из раздела 2.3. к $\Omega(parse)$ ещё раз. В $\Omega(parse)$ функция fn является нерекурсивной, удалим её из $\Omega(parse)$ (её корректность докажем отдельно), получим

$$\Omega_1(parse) = \{parse(fp, fm), fp(parse), fm(parse)\}.$$

В $\Omega_1(parse)$ отсутствуют независимые от $parse$ рекурсивные функции. Пропустим шаг 4 алгоритма преобразования рекурсий и сразу зададим УРФ $pURF$. Ниже приведён исходный код $pURF$:

```
pURF<<funcdef sn{ // Функция pURF принимающая список (N, str),
                  // N - номер функции, объединённой в УРФ, str - строка
    N<<sn:1;      // Сохранение первого аргумента в N
```

```
    s<<sn:2;      // Сохранение второго аргумента в s
    csU << (
        (N,1):=,
        (N,2):=,
        (N,3):=   // Сравнение N с тремя допустимыми вариантами значений
    ):?;          // Выбор номера пути вычисления и его сохранение в csU
    actU << (
         {  block{ // Тело функции parse с заменой всех вызовов функций
                   // на вызов УРФ
                s1<<s:1;
                case << ( (s1,'+'):=, (s1,'*'):=, (s1,'x'):= ):?;
                act<< ( {(2,s):pURF}, {(3,s):pURF}, {s:fn} );
                act:case:.>>break;
            }
         },
         { // Тело функции fp с заменой всех вызовов функций на вызов УРФ
            ( 1, (1, s:-1):pURF ):pURF
         },
         { // Тело функции fm с заменой всех вызовов функций на вызов УРФ
            ( 1, (1, s:-1):pURF ):pURF
         }
    ); // Формирование списка actU 3-х различных путей вычислений
       // в зависимости от N
    actU:csU:. // Выбор нужного пути вычисления и активация вычисления
    >> return; // Возврат результата вычислений
}
```

Информационный граф функции $pURF$ приведён на рис. 7. Согласно разделу 3. предусловие и постусловие функции $pURF$ формируются из предусловий и постусловий входящих функций. Зададим предусловие и постусловие для fp (для fm всё аналогично). Функция fp должна принимать строку str_1, которая начинается с символа '+', далее идут две подстроки, удовлетворяющие (1), а оставшаяся часть строки $tail$ может содержать любые символы. В результате работы функция должна вернуть $tail$. На формальном языке предусловие P_{fp} и постусловие Q_{fp} соответственно выражаются следующими формулами:

$$P(str_1) \wedge (\exists k \in \mathbb{N}\ (k < n) \wedge str_1 = \text{“}+\text{”} \circ f(k) \circ f(n-k) \circ tail)\,,$$

$$return = tail,$$

где $P(str_1)$ — предусловие функции $parse$, в котором имя переменной str заменено на str_1. Тогда предусловие функции $pURF$ с входным аргументом $sn = (N, s)$ будет следующим:

$$((N = 1) \wedge P(s)) \vee ((N = 2) \wedge P_{fp}(s)) \vee ((N = 3) \wedge P_{fm}(s)).$$

После упрощения получаем предусловие P_{URF}:

$$P(s) \wedge ((N = 1) \vee (N = 2 \wedge select(s, 1) = \text{‘}+\text{’}) \vee (N = 3 \wedge select(s, 1) = \text{‘}*\text{’}))\,.$$

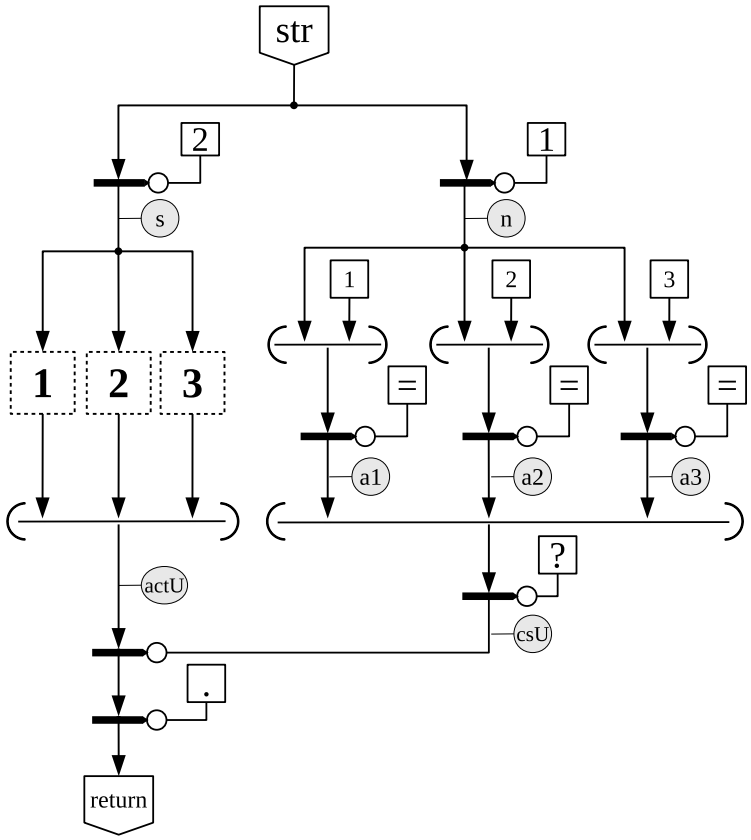

Рис. 7. Информационный граф функции $pURF$; к некоторым дугам приписаны идентификаторы, обозначенные серыми кружками; задержанные списки обозначены пунктиром и пронумерованы

Fig. 7. Data flow graph of the function $pURF$; some arcs has identifiers that are denoted by gray circles; delay lists are marked with the dashed line and enumerated;

Это означает, что строка s удовлетворяет $P(s)$, то есть представима в виде $f(n) \circ tail$; при $N = 2$ первый символ s равен ‘+’, а при $N = 3$ — ‘$*$’.

Постусловие функции $pURF$ имеет вид $((N = 1) \wedge Q) \vee ((N = 2) \wedge Q_{fp}) \vee ((N = 3) \wedge Q_{fm})$, и после упрощения приводится к формуле $return = tail$, то есть $Q_{URF} = Q$.

После автоматической разметки выходных дуг константных операторов и списков данных в графе $pURF$ размечаются дуги a_i, $i = 1, 2, 3$. К каждой дуге a_i приписывается пара формул:

$$(a_i \in bool) \wedge (a_i = true) \wedge (N = i),$$

$$(a_i \in bool) \wedge (a_i = false) \wedge \neg(N = i).$$

Далее, к дуге csU приписывается три формулы:

$$(csU \in int) \wedge (csU = i) \wedge (a_i = true) \bigwedge_{j \neq i} (a_j = false),\ i = 1, 2, 3.$$

Применение csU как функции к аргументу $argU$ приводит к расщеплению исходного графа на три, в каждый из которых попадёт один задержанный список.

При применении функции «.» произойдёт раскрытие задержанных списков. Назовём полученные в результате снятия задержки графы G_1, G_2 и G_3, где индекс совпадает с номером задержанного списка, попавшего в граф. Граф G_1 совпадёт с исходным графом функции $parseUn$, приведённым на рис. 6a, если в нём заменить идентификатор входного аргумента str на s (в этих графах отличаются константы задержанных списков, но на рисунке это не отражено). Графы G_2 и G_3 одинаковые и приведены на рис. 8a.

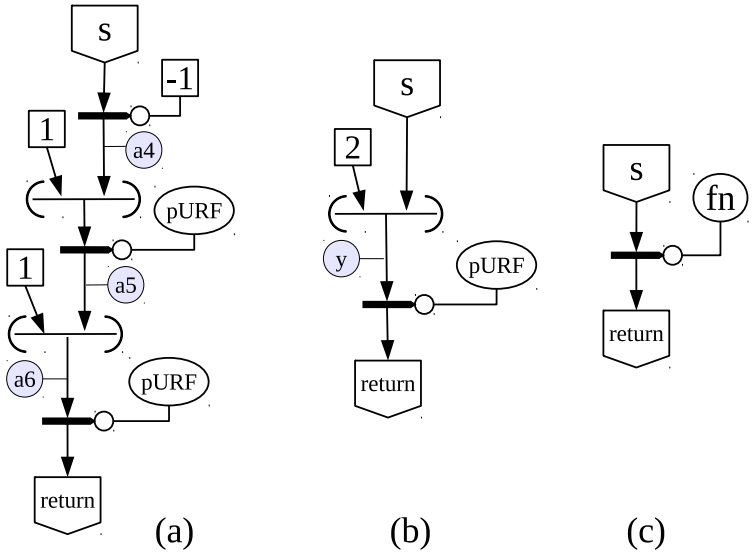

Рис. 8. Модифицированные информационные графы функции $pURF$

Fig. 8. Modified data flow graphs of the function $pURF$

Рассмотрим граф G_1. Его предусловие P_{URF1} имеет вид: $P_{URF} \wedge (N = 1)$. А после упрощения может быть записано как $P(s) \wedge (N = 1)$. Граф G_1 размечается так же, как исходный граф $parseUn$. Применение $case$ к act приводит к расщеплению графа на три, и после снятия задержки получаются графы, которые назовём G_{11}, G_{12} и G_{13}, где вторая цифра в индексе соответствует номеру попавшего в граф задержанного списка. Граф G_{11} приведен на рис. 8b. Граф G_{12} такой же, как G_{11}, только константа «2» заменяется на «3». Граф G_{13} приведён на рис. 8c.

Разметим граф G_{11}. Предусловие G_{11} имеет вид:

$$P_{URF} \wedge (N = 2) \wedge (select(s, 1) = \text{`}+\text{'}).$$

Значит, для аргумента (N, s), $N = 2$, а строка s представима как $f(n) \circ tail$ и первый её символ равен '+'. Это эквивалентно тому, что существует натуральное число k, такое что s представимо как $\text{`}+\text{''} \circ f(k) \circ f(n-k) \circ tail$.

В графе G_{11} константа «2» и аргумент s формируют список данных, которому присваивается идентификатор y. Далее к y применяется рекурсивный вызов функции $pURF$. Покажем, что y удовлетворяет предусловию функции $pURF$. Действительно, из предусловия G_{11} следует $P_{URF}(2, s)$ — предусловие функции $pURF$, для которого входной аргумент $(N, s) = (2, s)$. Тогда по индуктивному предположению рекурсивный вызов $pURF(y)$ вернёт верный ответ и к дуге $return$ припишется

формула: $return = tail$. Следовательно, постусловие выполнено и полностью размеченный граф G_{11} эквивалентен тождественно истинной формуле.

Для G_{12} всё аналогично, поэтому после разметки он также будет соответствовать тождественно истинной формуле.

Рассмотрим граф G_{13} (рис. 8c). Предусловие G_{13} имеет вид:

$$P_{URF} \wedge (N = 3) \wedge (select(s, 1) = \text{`}x\text{'}).$$

К аргументу s применяется функция fn, а результат её работы присваивается $return$.

Докажем корректность функции fn. Зададим предусловие P_{fn} и постусловие Q_{fn} для функции fn:

$$P(str_3) \wedge (str_3 = \text{“}x\text{”} \circ tail)\,,$$

$$return = tail.$$

В теле функции fn единственная функция, которая применяется ко входному аргументу, — это функция удаления первого элемента из списка. В результате из строки str_3 удаляется первый символ ‘x’, поэтому результат $return = sublist(s, 2, 1 + m) = tail$, что и требуется в постусловии. Следовательно, функция fn корректна.

На основе доказанной теоремы (корректности fn) можно разметить выходную дугу графа G_{13}: $return = tail$. Из последней формулы следует постусловие Q_{URF}. Значит, полностью размеченный граф G_{13} сворачивается в тождественно истинную формулу.

Остаётся разметить графы G_2 и G_3 (рис. 8a). Рассмотрим граф G_2, для G_3 всё аналогично. Предусловие P_{URF2} имеет вид: $P_{URF} \wedge (N = 2)$. После упрощения P_{URF2} можно записать $P(s) \wedge (N = 2) \wedge (select(s, 1) = \text{`}+\text{'})$, откуда следует, что для некоторого натурального k строку s можно представить в виде “ $+$ ” $\circ\, f(k) \circ f(n-k) \circ tail$.

В начале к аргументу s применяется функция удаления первого элемента из списка, и результату $f(k) \circ f(n-k) \circ tail$ присваивается идентификатор a_4. Список данных $(1, a_4)$ — допустимый аргумент для рекурсивного вызова функции $pURF$, поэтому по индуктивному предположению результат a_5 удовлетворяет постусловию $pURF$ и равен $f(n-k) \circ tail$. Новый формируемый список данных $(1, a_4)$ также удовлетворяет предусловию $pURF$, поэтому по индуктивному предположению $return$ будет равен $tail$. Откуда следует постусловие Q_{URF}. Значит, полностью размеченный G_2 (и аналогичный ему G_3) эквивалентен тождественно истинной формуле.

Таким образом, корректность $pURF$ доказана, из этого следует корректность функции $parse$.

5. Заключение

В работе рассмотрен алгоритм преобразования произвольной рекурсии в прямую при помощи универсальной рекурсивной функции. Для функций на языке Пифагор показано, как изменяется предусловие и постусловие программы при построении УРФ и объединении кода функций. Доказано, что из корректности УРФ следует корректность исходных функций. Рассмотрен пример доказательства корректности программы на языке Пифагор при объединении кода функций и построении УРФ.

Преобразование рекурсий может проводиться автоматически, поэтому описанный алгоритм можно внедрить в инструментальное средство поддержки доказательства программ на языке Пифагор.

Список литературы / References

[1] Непомнящий В. А., Рякин О. М., *Прикладные методы верификации программ*, Радио и связь, М., 1988, 255 с.; [Nepomnyashchiy V. A., Ryakin O. M., *Prikladnye metody verifikatsii programm*, Radio i svyaz, M., 1988, 255 pp., (in Russian).]

[2] Boyer R.S., Moore J.S., *A computational logic*, Academic Press, New York, 1979, 420 pp.

[3] Vazou N., Seidel E.L., Jhala R., Vytiniotis D., Peyton-Jones S., "Refinement Types for Haskell", *SIGPLAN Not.*, **49**:9 (2014), 269–282.

[4] Giesl J., "Termination of nested and mutually recursive algorithms", *Journal of Automated Reasoning*, **19**:1 (1997), 1–29.

[5] Легалов А. И., "Функциональный язык для создания архитектурно-независимых параллельных программ", *Вычислительные технологии*, **10**:1 (2005), 71–89; [Legalov A. I., "Functional language for creating of architectural independent parallel programs", *Computational Technologies*, **10**:1 (2005), 71–89, (in Russian).]

[6] Кропачева М. С., Легалов А. И., "Формальная верификация программ, написанных на функционально-потоковом языке параллельного программирования", *Моделирование и анализ информационных систем*, **19**:5 (2012), 81–99; English transl.: Kropacheva M. S., Legalov A. I., "Formal Verification of Programs in the Functional Data-Flow Parallel Language", *Automatic Control and Computer Sciences*, **47**:7 (2013), 373–384.

[7] Ushakova M. S., Legalov A. I., "Automation of Formal Verification of Programs in the Pifagor Language", *Modeling and Analysis of Information Systems*, **22**:4 (2015), 578–589.

[8] Giesl J., "Termination of nested and mutually recursive algorithms", *Journal of Automated Reasoning*, **19**:1 (1997), 1–29.

[9] Bevers E., Lewi J., "Proving termination of (conditional) rewrite systems", *Acta Informatica*, **30**:6 (1993), 537–568.

[10] Plumer L., *Termination Proofs for Logic Programs*, Lecture Notes in Artificial Intelligence, **446**, 1990, 142 pp.

[11] Giesl J., "Termination Analysis for Functional Programs Using Term Orderings", *International Static Analysis Symposium*, LNCS, **983**, 1995, 154–171.

[12] Walther C., "On Proving the Termination of Algorithm by Machine", *Artificial Intelligence*, **71**:1 (1994), 101–157.

[13] Кушниренко А. Г., Лебедев Г. В., *Программирование для математиков*, Наука, М., 1988, 384 с.; [Kushnirenko A. G., Lebedev G. V., *Programmirovanie dlya matematikov*, Nauka, M., 1988, 384 pp., (in Russian).]

[14] Головешкин В. А., Ульянов М. В., *Теория рекурсии для программистов*, Физматлит, М., 2006, 296 с.; [Goloveshkin V. A., Ul'yanov M. V., *Teoriya rekursii dlya programmistov*, Fizmatlit, M., 2006, 296 pp., (in Russian).]

[15] Мальцев А. И., *Алгоритмы и рекурсивные функции*, Наука, М., 1986, 368 с.; [Mal'tsev A. I., *Algoritmy i rekursivnye funktsii*, Nauka, M., 1986, 368 pp., (in Russian).]

[16] Hoare C. A. R., "An axiomatic basis for computer programming", *Communications of the ACM*, **10**:12 (1969), 576–585.

[17] Ушакова М. С., "Семантика типов данных функционально-потокового языка параллельного программирования Пифагор", *Образовательные ресурсы и технологии*, 2016, № 2(14), 263–269; [Ushakova M. S., "Semantics of program data types for functional data-flow parallel programming language Pifagor", *Educational resources and technologies*, 2016, № 2(14), 263–269, (in Russian).]

[18] Шилов Н. В., *Основы синтаксиса, семантики, трансляции и верификации программ*, Издательство СО РАН, Новосибирск, 2009, 340 с.; [Shilov N. V., *Osnovy sintaksisa, semantiki, translyatsii i verifikatsii programm*, Izdatelstvo SO RAN, Novosibirsk, 2009, 340 pp., (in Russian).]

Ushakova M. S., Legalov A. I., "Verification of Programs with Mutual Recursion in the Pifagor Language", *Modeling and Analysis of Information Systems*, **25**:4 (2018), 358–381.

DOI: 10.18255/1818-1015-2018-4-358-381

Abstract. In the article, we consider verification of programs with mutual recursion in the data driven functional parallel language Pifagor. In this language the program could be represented as a data flow graph, that has no control connections, and has only data relations. Under these conditions it is possible to simplify the process of formal verification, since there is no need to analyse resource conflicts, which are present in the systems with ordinary architectures. The proof of programs correctness is based on the elimination of mutual recursions by program transformation. The universal method of mutual recursion of an arbitrary number of functions elimination consists in constructing the universal recursive function that simulates all the functions in the mutual recursion. A natural number is assigned to each function in mutual recursion. The universal recursive function takes as its argument the number of a function to be simulated and the arguments of this function. In some cases of the indirect recursion it is possible to use a simpler method of program transformation, namely, the merging of the functions code into a single function. To remove mutual recursion of an arbitrary number of functions, it is suggested to construct a graph of all connected functions and transform this graph by removing functions that are not connected with the target function, then by merging functions with indirect recursion and finally by constructing the universal recursive function. It is proved that in the Pifagor language such transformations of functions as code merging and universal recursive function construction do not change the correctness of the initial program. An example of partial correctness proof is given for the program that parses a simple arithmetic expression. We construct the graph of all connected functions and demonstrate two methods of proofs: by means of code merging and by means of the universal recursive function.

Keywords: data driven functional parallel programming, Pifagor programming language, correctness of recursions, elimination of mutual recursion, universal recursive function

On the authors:
Mariya S. Ushakova, orcid.org/0000-0003-4234-2714, graduate student,
Siberian Federal University, Institute of Space and Information Technology,
26 Kirenskogo str., Krasnoyarsk 660074, Russia, e-mail: ksv@akadem.ru

Alexander I. Legalov, orcid.org/0000-0002-5487-0699, doctor of science,
Siberian Federal University, Institute of Space and Information Technology,
26 Kirenskogo str., Krasnoyarsk 660074, Russia, e-mail: legalov@mail.ru

Acknowledgments:
The research is supported by RFBR (research project No 17-07-00288).

Моделирование и анализ информационных систем. Т. 25, № 4 (2018), с. 382–387
Modeling and Analysis of Information Systems. Vol. 25, No 4 (2018), pp. 382–387

Анализ сигналов

DOI: 10.18255/1818-1015-2018-4-382-387

УДК 519.248.6

Эффективный алгоритм определения уровня полезных сигналов при расшифровке магнитных и вихретоковых дефектограмм

Кузьмин Е. В., Горбунов О. Е., Плотников П. О., Тюкин В. А.

получена 7 мая 2018

Аннотация. Для обеспечения безопасности движения на железнодорожном транспорте регулярно проводится неразрушающий контроль рельсов с применением различных подходов и методов, включая методы магнитной и вихретоковой дефектоскопии. Актуальной задачей по-прежнему остается автоматический анализ больших массивов данных (дефектограмм), которые поступают от соответствующего оборудования. Под анализом понимается процесс определения по дефектограммам наличия дефектных участков наряду с выявлением конструктивных элементов рельсового пути. При этом в условиях значительных объемов поступающей на обработку информации наибольший интерес представляют быстрые и эффективные алгоритмы анализа данных. Данная статья является дополнением к предыдущей статье авторов, посвященной задаче автоматического определения порогового уровня амплитуд полезных сигналов при расшифровке дефектограмм магнитных и вихретоковых дефектоскопов, в которой был предложен алгоритм нахождения порогового уровня шума рельсов с его теоретическим обоснованием, а также рассматривались примеры работы алгоритма на фрагментах реальных магнитных и вихретоковых дефектограмм. В настоящей статье приводится простая и эффективная реализация этого алгоритма, которая с успехом применяется на практике при автоматическом анализе магнитных и вихретоковых дефектограмм.

Ключевые слова: неразрушающий контроль рельсов, магнитная и вихретоковая дефектоскопия, обнаружение дефектов, автоматический анализ магнитных и вихретоковых дефектограмм

Для цитирования: Кузьмин Е. В., Горбунов О. Е., Плотников П. О., Тюкин В. А., "Эффективный алгоритм определения уровня полезных сигналов при расшифровке магнитных и вихретоковых дефектограмм", *Моделирование и анализ информационных систем*, **25**:4 (2018), 382–387.

Об авторах:
Кузьмин Егор Владимирович, orcid.org/0000-0003-0500-306X, д-р физ.-мат. наук, профессор кафедры теоретической информатики, Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: kuzmin@uniyar.ac.ru, kuzminev@nddlab.com

Горбунов Олег Евгеньевич, orcid.org/0000-0001-6274-9971, канд. физ.-мат. наук, генеральный директор,
ООО «Центр инновационного программирования», NDDLab, ул. Союзная, 144, г. Ярославль, 150008 Россия,
e-mail: gorbunovoe@nddlab.com

Плотников Петр Олегович, orcid.org/0000-0001-5687-7969, инженер-технолог, ООО «Центр инновационного программирования», NDDLab, ул. Союзная, 144, г. Ярославль, 150008 Россия, e-mail: plotnikovpo@nddlab.com

Тюкин Вадим Александрович, orcid.org/0000-0001-9149-7435, руков. сектора разработки, ООО «Центр инновационного программирования», NDDLab, ул. Союзная, 144, г. Ярославль, 150008 Россия, e-mail: tyukinva@nddlab.com

Введение

Для обеспечения безопасности движения на железнодорожном транспорте регулярно проводится неразрушающий контроль рельсов с применением различных подходов и методов, включая методы магнитной и вихретоковой дефектоскопии. Актуальной задачей по-прежнему остается автоматический анализ [2–5] больших массивов данных (дефектограмм), которые поступают от соответствующего оборудования. Под анализом понимается процесс определения по дефектограммам наличия дефектных участков наряду с выявлением конструктивных элементов рельсового пути. При этом в условиях значительных объемов поступающей на обработку информации наибольший интерес представляют быстрые и эффективные алгоритмы анализа данных.

Данная статья является дополнением к статье [1], посвященной задаче автоматического определения порогового уровня амплитуд полезных сигналов при расшифровке дефектограмм магнитных и вихретоковых дефектоскопов, в которой был предложен алгоритм нахождения порогового уровня шума рельсов с его теоретическим обоснованием, а также рассматривались примеры работы алгоритма на фрагментах реальных магнитных и вихретоковых дефектограмм. В настоящей статье приводится простая и эффективная реализация этого алгоритма, которая с успехом применяется на практике при автоматическом анализе магнитных и вихретоковых дефектограмм.

В статье рассматриваются обобщения реальных устройств в виде абстрактных 8-разрядного магнитного и 10-разрядного вихретокового дефектоскопов. Значения амплитуд сигналов регистрируются дефектоскопами в виде натуральных чисел от 1 до 1024 в 10-разрядном случае и от 1 до 256 в случае 8-разрядного дефектоскопа.

При автоматическом анализе дефектограммы обычно разбиваются на фрагменты, которые, например, могут соответствовать 50-метровым участкам пути, т. е. при снятии показаний дефектоскопа с каждого миллиметра пути блок анализа представляет собой массив из 50000 элементов, где элемент массива — это значение амплитуды сигнала.

Как и ранее, под пороговым уровнем шума понимается отклонение Level от среднего значения μ сигналов рассматриваемого фрагмента дефектограммы (данные с 50-метрового участка), при котором сигналы со значениями амплитуд из диапазона $[\mu - \mathrm{Level}; \mu + \mathrm{Level}]$ являются шумом рельсов.

Несмотря на то что шум рельсов характеризуется нормальным законом распределения вероятностей, из-за наличия сильных сигналов (от дефектов и конструктивных элементов) не представляется возможным применение в чистом виде правила трех сигм, согласно которому около 99,73% сигналов шума лежит в интервале $[\mu - 3\sigma; \mu + 3\sigma]$, где μ — это выборочное среднее, а σ — среднее квадратическое отклонение. Другими словами, сильные сигналы оказывают значительное влияние на результат вычисления среднего квадратического отклонения по всей выборке.

Для построения приемлемого уровня шума $\mathrm{Level} \approx 3\sigma$ необходимо при вычислении приближенного значения σ исключить из рассмотрения (сильные) сигналы анализируемой выборки, которые не укладываются в рамки закона нормального распределения. Таким образом, для нахождения порога Level предлагается использовать следующий итерационный алгоритм, реализующий эту идею.

Алгоритм

Основные этапы алгоритма нахождения порогового уровня шума рельсов Level:

1. Получить массив значений амплитуд сигналов $X[1\,..\,50000]$ (данные от дефектоскопа с 50-метрового участка рельсового пути).

2. Вычислить среднее арифметическое значение μ элементов массива X.

3. По элементам массива X, значения которых лежат в диапазоне $[\mu - i; \mu + i]$, где $i \in \mathbb{N}$ изначально равно 1, построить среднее квадратическое отклонение σ от значения μ. Увеличить i на 1.

4. Повторять пункт 3 до тех пор, пока не будут выполнены условия $3 \cdot \sigma < i$ и $3 \cdot \sigma' \geqslant (i-1)$, где σ' — это значение σ, полученное на предыдущем шаге; при числе элементов со значением из $[\mu - i; \mu + i]$ более 30% от общего числа.

5. Присвоить переменной Level значение $3 \cdot \sigma$.

6. Выдать значение Level в качестве искомого порогового уровня шума.

Ниже приведена эффективная реализация описанного алгоритма в псевдокоде.

```
PROGRAM NoiseLevel
CONST N=1024; L=50000;
VAR X: array [1..L] of integer;     /* массив значений амплитуд сигналов          */
    A: array [1..N] of integer = 0; /* A[k] -- количество сигналов с амплитудой k */
    h, i, k, s, f, cnt, cntall: word;
    mu, sig, oldsig, sum, Level: double;
BEGIN_PROGRAM
  load(X); h=size(X); cntall=0;
  for i=1:h
    A[X[i]]=A[X[i]]+1; cntall=cntall+1;
  end;
  cnt=0; sum=0;
  for k=1:N
    cnt=cnt+A[k]; sum=sum+A[k]*k;
  end;
  mu=sum/cnt; k=round(mu); /* round() -- округление до ближайшего целого */
  sum=(k-mu)*(k-mu)*A[k]; cnt=A[k]; sig=sqrt(sum/cnt);
  for k=1:N
    s=round(mu-k); f=round(mu+k);
    if s>=1 cnt=cnt+A[s]; sum=sum+(s-mu)*(s-mu)*A[s]; end;
    if f<=N cnt=cnt+A[f]; sum=sum+(f-mu)*(f-mu)*A[f]; end;
    oldsig=sig; sig=sqrt(sum/cnt);
    if cnt>0.3*cntall & 3*sig<k & 3*oldsig>=(k-1) break; end;
  end;
  Level=3*sig;
END_PROGRAM.
```

Этот алгоритм имеет в своей основе ту же самую идею, что и описанный ранее алгоритм из статьи [1]. Но в отличие от прошлой реализации алгоритма, которая была призвана адекватно отразить интересные свойства нормального распределения, в том числе и свойства определенных функций $\varphi(x)$ и $\psi(x)$, эта реализация является

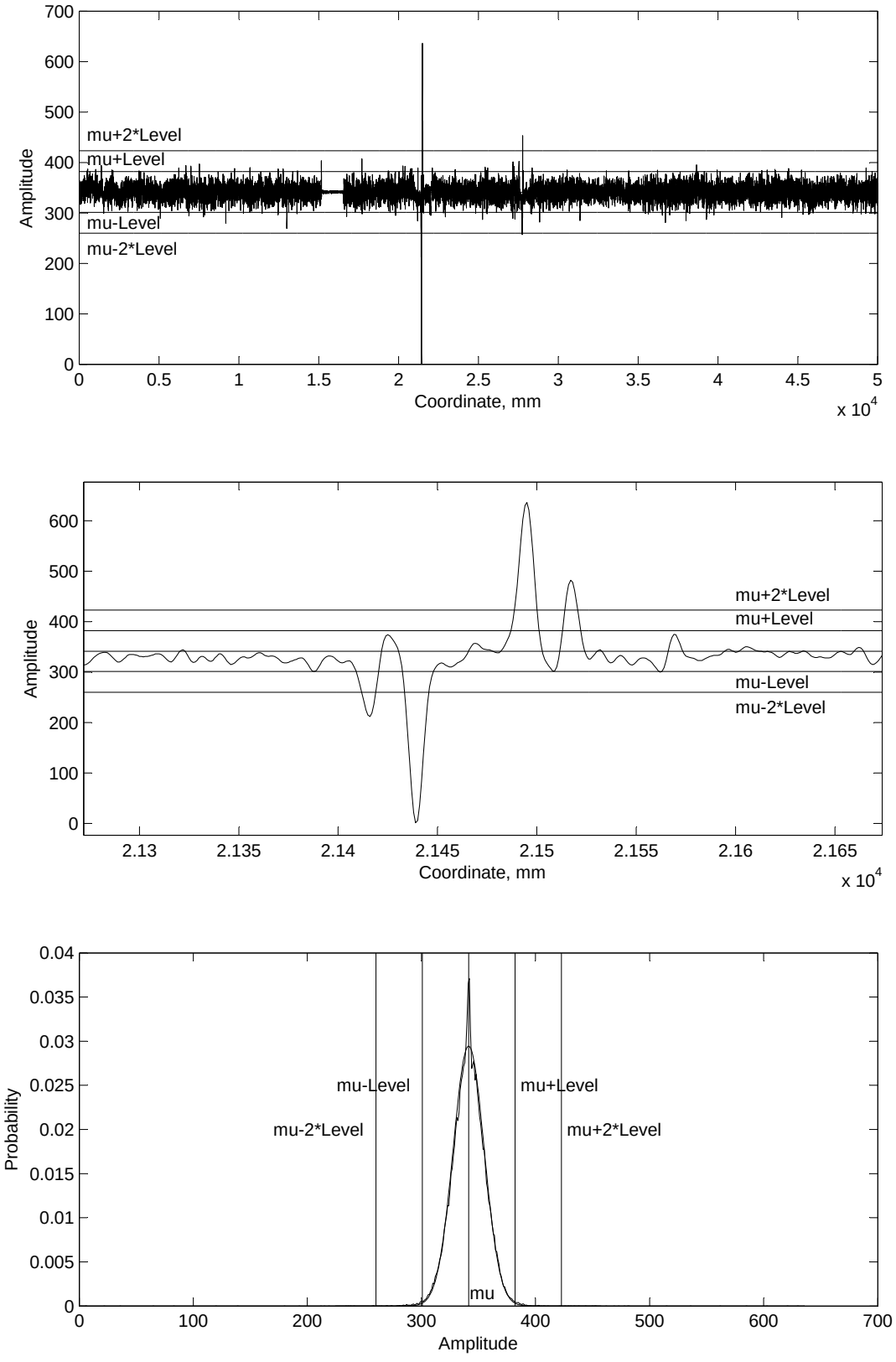

Рис. 1. Представления данных дефектограмм с результатами работы алгоритма
Fig. 1. Representations of flaw detector data and algorithm work results

крайне простой и эффективной. По сути, для вычисления порогового уровня шума Level необходимо построить среднеквадратическое отклонение σ от μ по данным из диапазона $[\mu - \text{Level}; \mu + \text{Level}]$, где μ — среднее арифметическое значение элементов исходного массива амплитуд X. При этом вычисление σ производится с использованием вспомогательного массива A, который хранит «частоты» соответствующих его индексам амплитуд сигналов.

Условия остановки алгоритма имеют прежнее обоснование [1] и ориентированы на дефектограммы, которые содержат от $\approx 90\%$ до $\approx 50\%$ сигналов шума (требование обработки более 30% всех сигналов фрагмента данных).

Пример. На верхнем графике рис. 1 представлена дефектограмма, которая была записана 10-разрядным вихретоковым дефектоскопом на 50-метровом участке рельсового пути. Средний график — фрагмент записи, соответствующий сварному стыку рельсов. Регистрация данных проводилась каждый миллиметр пути (ось X). Амплитудное значение сигнала, полученное на одном шаге сканирования, откладывается по оси Y. Нижний график рис. 1 — оценка плотности распределения вероятности появления некоторой амплитуды на рассматриваемом участке рельсов. На всех графиках рис. 1 показаны результаты алгоритма в виде линий отсечки шума рельсов и уровня начала полезных сигналов: $\mu = 341{,}4012$, $\sigma = 13{,}5476$, $\text{Level} = 40{,}6428$.

На рассмотренном примере видно, что шум рельсов (в приближении) подчиняется закону нормального распределения вероятностей. График оценки плотности распределения шума близок к графику плотности нормального распределения с параметрами μ и σ, где μ — математическое ожидание, а σ — среднее квадратическое отклонение амплитуд сигналов шума ($\text{Level} = 3\sigma$).

Список литературы / References

[1] Кузьмин Е. В., Горбунов О. Е., Плотников П. О., Тюкин В. А., “Об определении уровня полезных сигналов при расшифровке магнитных и вихретоковых дефектограмм”, *Модел. и анализ информ. систем*, **24**:6 (2017), 760–771; [Kuzmin E. V., Gorbunov O. E., Plotnikov P. O., Tyukin V. A., “On Finding a Threshold of Useful Signals in the Analysis of Magnetic and Eddy Current Defectograms”, *Modeling and Analysis of Information Systems*, **24**:6 (2017), 760–771, (in Russian).]

[2] Марков А. А., Кузнецова Е. А., *Дефектоскопия рельсов. Формирование и анализ сигналов. Кн. 1. Основы*, КультИнформПресс, СПб., 2010; [Markov A. A., Kuznetsova E. A., *Rails flaw detection. Formation and analysis of signals. Book 1. Principles*, KultInformPress, St. Petersburg, 2010, (in Russian).]

[3] Марков А. А., Кузнецова Е. А., *Дефектоскопия рельсов. Формирование и анализ сигналов. Кн. 2. Расшифровка дефектограмм*, Ультра Принт, СПб., 2014; [Markov A. A., Kuznetsova E. A., *Rails flaw detection. Formation and analysis of signals. Book 2. Data interpretation*, Ultra Print, St. Petersburg, 2014, (in Russian).]

[4] Тарабрин В. Ф., Зверев А. В., Горбунов О. Е., Кузьмин Е. В., “О фильтрации данных при автоматической расшифровке дефектограмм АПК «АСТРА»”, *В мире неразрушающего контроля*, **64**:2 (2014), 5–9; [Tarabrin V. F., Zverev A. V., Gorbunov O. E., Kuzmin E. V., “About Data Filtration of the Defectogram Automatic Interpretation by Hardware and Software Complex ”ASTRA””, *NDT World*, **64**:2 (2014), 5–9, (in Russian).]

[5] Тарабрин В. Ф., Кузьмин Е. В., Горбунов О. Е., Зверев А. В., “Об определении динамического порога уровня сигналов при автоматической расшифровке дефектограмм АПК «АСТРА»”, *Сборник тезисов научных докладов XX Всероссийской научно-техн. конф. по неразрушающему контролю и технической диагностике*, Спектр,

M., 2014, 145–147; [Tarabrin V. F., Kuzmin E. V., Gorbunov O. E., Zverev A. V., "Ob opredelenii dinamicheskogo poroga urovnya signalov pri avtomaticheskoy rasshifrovke defektogramm APK "ASTRA"", *Sbornik tezisov nauchnykh dokladov XX vserossiyskoy nauchno-technicheskoy konferentsii po nerazrushayuschemu kontrolyu i tekhnicheskoy diagnostike*, Spektr, Moscow, 2014, 145–147, (in Russian).]

[6] Вентцель Е. С., Овчаров Л. А., *Теория вероятностей и ее инженерные приложения*, Высшая школа, М., 2000; [Ventcel E. S., Ovcharov L. A., *Probability Theory and Its Engineering Applications*, Vysshaya Shkola Publishers, Moscow, 2000, (in Russian).]

[7] Гмурман В. Е., *Теория вероятностей и математическая статистика*, Высшая школа, М., 2004; [Gmurman V. E., *Probability Theory and Mathematical Statistics*, Vysshaya Shkola Publishers, Moscow, 2004, (in Russian).]

[8] Феллер В., *Введение в теорию вероятностей и ее приложения*, В 2-х томах. Пер. с англ., Мир, М., 1984; In English: Feller W., *An Introduction to Probability Theory and Its Applications*. V. I–II, John Willey & Sons, 1970–1971.

Kuzmin E. V., Gorbunov O. E., Plotnikov P. O., Tyukin V. A., "An Efficient Algorithm for Finding a Threshold of Useful Signals in the Analysis of Magnetic and Eddy Current Defectograms", *Modeling and Analysis of Information Systems*, **25**:4 (2018), 382–387.

DOI: 10.18255/1818-1015-2018-4-382-387

Abstract. To ensure traffic safety of railway transport, non-destructive testing of rails is regularly carried out by using various approaches and methods, including magnetic and eddy current flaw detection methods. An automatic analysis of large data sets (defectgrams) that come from the corresponding equipment is still an actual problem. The analysis means a process of determining the presence of defective sections along with identifying structural elements of railway tracks on defectograms. At the same time, under the conditions of significant volumes of incoming information, fast and efficient algorithms of data analysis are of most interest. This article is an addition to the previous article devoted to the problem of automatic determination of a threshold level of amplitudes of useful signals (from defects and structural elements of a railway track) during the analysis of defectograms (records) of magnetic and eddy current flaw detectors, which contains an algorithm for finding the threshold level of a rail noise and its theoretical justification with examples of its operation on several fragments of real magnetic and eddy current defectograms. The article presents a simple and effective implementation of the algorithm, which is successfully used in practice for the automatic analysis of magnetic and eddy current defectograms.

Keywords: nondestructive testing, magnetic and eddy current testing, rail flaw detection, automated analysis of magnetic and eddy current defectograms

On the authors:
Egor V. Kuzmin, orcid.org/0000-0003-0500-306X, doctor of science, associate professor,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003 Russia, e-mail: kuzmin@uniyar.ac.ru, kuzminev@nddlab.com

Oleg E. Gorbunov, orcid.org/0000-0001-6274-9971, PhD, general director,
Center of Innovative Programming, NDDLab,
144 Soyuznaya str., Yaroslavl, 150008, Russia, e-mail: gorbunovoe@nddlab.com

Petr O. Plotnikov, orcid.org/0000-0001-5687-7969, production engineer,
Center of Innovative Programming, NDDLab,
144 Soyuznaya str., Yaroslavl, 150008, Russia, e-mail: plotnikovpo@nddlab.com

Vadim A. Tyukin, orcid.org/0000-0001-9149-7435, head of software development,
Center of Innovative Programming, NDDLab,
144 Soyuznaya str., Yaroslavl, 150008, Russia, e-mail: tyukinva@nddlab.com

Моделирование и анализ информационных систем. Т. 25, №4 (2018), с. 388–401
Modeling and Analysis of Information Systems. Vol. 25, No 4 (2018), pp. 388–401

Теория графов

DOI: 10.18255/1818-1015-2018-4-388-401

УДК 519.17

Остовное дерево в делимом кратном графе

Смирнов А. В.

получена 29 июля 2018

Аннотация. В статье рассматриваются неориентированные кратные графы произвольной натуральной кратности $k > 1$. Кратный граф содержит ребра трех типов: обычные, кратные и мультиребра. Ребра последних двух типов представляют собой объединение k связанных ребер, которые соединяют 2 или $k + 1$ вершину соответственно. Связанные ребра могут использоваться только согласованно. Если вершина инцидентна кратному ребру, то она может быть инцидентна другим кратным ребрам, а также она может быть общим концом k связанных ребер мультиребра. Если вершина является общим концом мультиребра, то она не может быть общим концом никакого другого мультиребра.

Особое внимание уделяется классу делимых кратных графов, которые отличаются возможностью выделения k частей, согласованных на всех связанных ребрах и не содержащих общих ребер. Каждая из частей является обычным графом.

Вводится понятие кратного дерева, определяюся его основные свойства. В отличие от обычных деревьев количество ребер в кратных деревьях не фиксировано. Для делимых деревьев в работе приводится и обосновывается оценка минимального и максимального количества ребер.

Далее определяются понятия остовного дерева и полного остовного дерева. Для делимых графов доказывается критерий полноты остовного дерева. Также доказано, что полное остовное дерево всегда существует в делимом графе.

Если кратный граф является взвешенным, то для него можно поставить задачу о минимальном остовном дереве, а также о минимальном полном остовном дереве. В работе предложен эвристический алгоритм поиска минимального полного остовного дерева в делимом графе.

Ключевые слова: кратный граф, кратное дерево, делимый граф, остовное дерево, полное остовное дерево, минимальное остовное дерево

Для цитирования: Смирнов А. В., "Остовное дерево в делимом кратном графе", *Моделирование и анализ информационных систем*, **25**:4 (2018), 388–401.

Об авторах:
Смирнов Александр Валерьевич, orcid.org/0000-0002-0980-2507, канд. физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: alexander_sm@mail.ru

Благодарности:
Работа выполнена при поддержке Российского фонда фундаментальных исследований (проект № 17-07-00823 А).

Введение

В данной статье будет дано понятие *кратного дерева* и рассмотрена задача о *минимальном остовном дереве в делимом кратном графе*. Определение кратного графа

кратности $k > 1$ и делимого кратного графа было сформулировано в статье [1]. Там же была рассмотрена задача о кратчайшем кратном пути между двумя вершинами. Кратные графы содержат три типа ребер (обычные, кратные и мультиребра) и являются обобщением обычных графов – по сути, обычный граф имеет кратность $k = 1$.

Среди других известных обобщений графов наиболее близкими нам концепциями являются мультиграфы, гиперграфы (см., например, [2] – [3]), а также метаграфы (см. [4] – [5]). Действительно, как и в мультиграфах, в кратных графах допускается наличие нескольких ребер между парой вершин (набор таких ребер мы будем в дальнейшем называть *кратным ребром*), однако в случае кратного графа количество таких ребер должно быть строго равным k. В кратных графах присутствуют *мультиребра*, соединяющие между собой $k + 1$ вершину. Но в отличие от гиперребер гиперграфа, мультиребро представляется в виде k связанных ребер, имеющих один общий конец, причем все эти k ребер должны использоваться согласованно. По сути, понятие мультиребра близко понятию ребра между вершиной и метавершиной в метаграфе. При этом в метаграфе, напомним, метапуть между двумя метавершинами фактически моделирует причинно-следственные связи в некоторой предметной области. Однако в кратном графе используется принципиально иной подход к определению пути: *кратный путь* должен состоять ровно из k обычных путей, проходящих по обычным ребрам, а также по связанным ребрам кратных и мультиребер; при этом пути должны быть согласованы (одинаковы) на кратных и мультиребрах. Поэтому кратный граф нельзя считать частным случаем метаграфа.

Отметим также, что частным случаем кратного графа является кратная сеть (см. [6] – [7]). Задача о наибольшем потоке в кратной сети обобщает классическую задачу (см. [8]) и имеет ряд приложений в сфере экономики, управления, финансов. В частности, кратные сети и потоки используются для поиска решения NP-полной задачи целочисленного сбалансирования трех- и четырехмерной матрицы (см., например, [9] – [10]).

1. Кратный граф и кратный путь

Прежде чем ввести определение кратного дерева, напомним несколько понятий, относящихся к кратным графам (подробней см. в [1]).

Определение 1. *В качестве* кратного графа *произвольной натуральной кратности $k > 1$ рассматривается граф $G(X, E)$, между вершинами которого могут быть ребра одного из 3 видов:*

1) обычное ребро *e^o; множество обычных ребер обозначим через E^o;*

2) кратное ребро *e^k между двумя вершинами, которое состоит из k одинаковых связанных ребер; связанные ребра кратного ребра могут использоваться только согласованно; множество кратных ребер обозначим через E^k;*

3) связанное ребро *e между двумя вершинами, имеющее один общий конец с другими $k - 1$ ребрами (у любых двух из k связанных ребер только один конец является общим); множество связанных общей вершиной ребер будем называть*

мультиребром e^m*; связанные ребра мультиребра могут использоваться только согласованно; множество мультиребер обозначим через* E^m.

Если вершина инцидентна какому-либо кратному ребру, то она может быть инцидентна другим кратным ребрам, а также она может быть общим концом какого-либо мультиребра.

Если вершина является общим концом какого-либо мультиребра, то она не может быть общим концом никакого другого мультиребра.

Мы ограничимся рассмотрением неориентированных кратных графов.

Определение 2. Делимым кратным графом *назовем такой граф, в котором между двумя концами одного мультиребра не существует пути, проходящего только по обычным ребрам.*

Очевидно, что при удалении всех мультиребер делимый граф распадется на n компонент связности (связность здесь понимается в том же смысле, что и для обычных графов), каждая из которых содержит только кратные ребра либо только обычные ребра. При этом связанные ребра каждого мультиребра можно пронумеровать от 1 до k таким образом, что каждой компоненте связности, содержащей только обычные ребра, будут инцидентны связанные ребра мультиребер с одинаковыми номерами.

Определение 3. Частью G_i $(i \in \overline{1,k})$ *делимого графа* $G(X,E)$ *назовем подграф, содержащий связанные ребра с номером* i *всех кратных и мультиребер, а также компоненты связности, состоящие из обычных ребер и инцидентные* i*-м связанным ребрам всех мультиребер.*

Очевидно, что каждая часть G_i является обычным графом. При этом возможность выделения частей G_i является особенностью делимых графов. В общем случае получить части G_i не удастся.

Определение 4. *Вершина является* висячей*, если она инцидентна только одному ребру.*

Дадим теперь определение кратного пути. Основное отличие кратного пути от пути в обычном графе состоит в том, что связанные ребра каждого кратного и мультиребра должны проходиться в этом пути согласованно.

Определение 5. $S(x,y) = \cup_{i=1}^{k} S^i(x,y)$ *является* кратным путем из вершины x в вершину y *в графе* $G(X,E)$*, если выполнены следующие условия:*

1) $S^i(x,y) = \big(\{x, v_1^i\}, \{v_1^i, v_2^i\}, \ldots, \{v_{p-1}^i, v_p^i\}, \{v_p^i, y\}\big)$*, где* $p \geq 0$*, – последовательность ребер, представляющая собой обычный (некратный) путь из* x *в* y*, где каждое ребро* $\{a,b\}$ *является либо обычным ребром в графе* $G(X,E)$*, либо* i*-м связанным ребром кратного или мультиребра. Если в путь* $S(x,y)$ *не входит ни одного кратного или мультребра, то* $S^2(x,y) = S^3(x,y) = \ldots = S^k(x,y) = \varnothing$*;*

2) вершины, не являющиеся общим концом связанных ребер мультиребра и не инцидентные кратным ребрам, могут встречаться в $S^i(x,y)$ *несколько раз, то есть* $S^i(x,y)$ *может содержать циклы;*

3) вершины, инцидентные кратным ребрам либо являющиеся общим концом мультиребра, не могут встретиться в $S^i(x,y)$ *дважды;*

4) любое обычное ребро может встречаться в $S^i(x,y)$ несколько раз, причем направления, в которых оно проходится в разных вхождениях, могут не совпадать;

5) обычное ребро, входящее в $S^i(x,y)$, может также входить в любой $S^j(x,y)$, $j \neq i$;

6) все пути $S^i(x,y)$ согласованы (одинаковы) на общей части. Это условие означает, что если связанное ребро какого-то кратного или мультиребра входит в некоторый путь $S^i(x,y)$, то остальные связанные ребра должны входить во все $S^j(x,y)$, $j \neq i$ (по одному связанному ребру в каждый $S^j(x,y)$). При этом порядок вхождения всех кратных и мультиребер во все $S^i(x,y)$ одинаков;

7) если $S(x,y)$ содержит мультиребро $\{x_0, \{x_1, \ldots, x_k\}\}$, проходимое в направлении от общего конца, то он не может содержать никакого другого мультиребра $\{y_0, \{x_1, \ldots, x_k\}\}$, проходимого в том же направлении. Аналогичное условие должно выполняться и в случае движения к общему концу.

Определение 6. *Кратный путь $S(x,y)$ является* кратным циклом, *если $x = y$ и $S(x,y) \neq \varnothing$.*

Теперь мы можем ввести понятие связности для кратного графа.

Определение 7. *Кратный граф $G(X,E)$ является* связным, *если одновременно выполнены два условия:*

1) для любых двух вершин $x \in X$, $y \in X$, каждая из которых либо инцидентна кратному ребру, либо является общим концом мультиребра, существует кратный путь $S(x,y)$;

2) невозможно выделить такой подграф $G' \subset G$, который будет содержать только обычные ребра, и при этом подграфы G' и $G \setminus G'$ не будут соединены ни одним ребром (обычным ребром или связанным ребром мультиребра).

В отличие от обычных графов связность кратного графа не предполагает наличие кратных путей из каждой вершины в каждую. Фактически в связном кратном графе между каждой парой вершин должен существовать обычный (некратный) путь, использующий связанные ребра кратных и мультиребер несогласованно, а кратные пути обязательно должны существовать только для пар вершин, каждая из которых инцидентна кратным ребрам или является общим концом мультиребра.

Отметим, что для делимого кратного графа определение связности может быть переписано в более простой форме, что обусловлено структурой графа.

Определение 8. *Делимый кратный граф $G(X,E)$ является* связным, *если одновременно выполнены два условия:*

1) для любых двух вершин $x \in X$, $y \in X$, каждая из которых либо инцидентна кратному ребру, либо является общим концом мультиребра, существует кратный путь $S(x,y)$;

2) каждая из частей G_i является связным графом.

2. Кратные деревья и их свойства

Определение 9. Кратное дерево – *это связный кратный граф без циклов.*

Пример 1.

Рассмотрим простые примеры кратных деревьев кратности 2 с похожей структурой (жирными линиями отмечены кратные ребра, раздваивающимися – мультиребра).

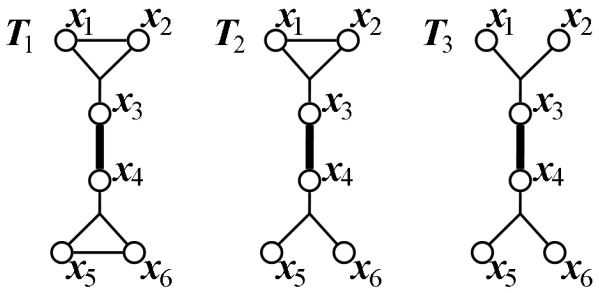

Рис. 1. Примеры кратных деревьев
Fig. 1. Examples of multiple trees

Все три представленых на рис. 1 графа являются деревьями: действительно, для каждого из них выполнено условие связности (в кратном смысле), и ни в одном из графов невозможно построить кратный цикл. При этом дерево T_3 является делимым, а деревья T_1 и T_2 – нет.

Хотя количество вершин во всех деревьях одинаково, количество ребер различно – это особенность кратных деревьев. Оценка минимального и максимального количества ребер в делимом дереве будет приведена и обоснована ниже.

Особо отметим, что в кратном дереве может не быть ни одной висячей вершины (пример – дерево T_1).

Определение 10. *Два кратных пути $S_1(x, y)$ и $S_2(x, y)$ назовем* подобными, *если для них выполнены следующие условия:*

1) порядок вхождения кратных и мультиребер в пути $S_1(x, y)$ и $S_2(x, y)$ одинаков;

2) путь $S_2(x, y)$ может быть получен перестановкой некратных участков пути $S_1(x, y)$ между простыми путями $S_1^i(x, y)$, причем менять местами можно только участки, заключенные между общими концами двух мультиребер либо между начальной/конечной вершиной пути и общим концом какого-либо мультиребра.

Пример 2.

Чтобы проиллюстрировать определение подобных путей, рассмотрим дерево T_1 из примера 1 (рис. 1). В этом дереве существует два различных кратных пути из x_1 в x_5. Первый путь $S_1(x_1, x_5)$ состоит из двух обычных путей

$$S_1^1(x_1, x_5) = \Big(\{x_1, x_3\}, \{x_3, x_4\}, \underline{\{x_4, x_5\}}\Big),$$

$$S_1^2(x_1, x_5) = \Big(\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \underline{\underline{\{x_4, x_6\}}}, \underline{\underline{\{x_6, x_5\}}}\Big),$$

а второй путь $S_2(x_1, x_5)$ состоит из двух обычных путей

$$S_2^1(x_1, x_5) = \Big(\{x_1, x_3\}, \{x_3, x_4\}, \underline{\underline{\{x_4, x_6\}}}, \underline{\underline{\{x_6, x_5\}}}\Big),$$

$$S_2^2(x_1, x_5) = \Big(\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \underline{\{x_4, x_5\}}\Big).$$

Нетрудно заметить, что путь $S_2(x_1, x_5)$ получается из пути $S_1(x_1, x_5)$ перестановкой некратных участков пути между простыми путями (участки, отмеченные одинарным и двойным подчеркиванием). При этом переставляемые участки заключены между общим концом мультиребра и конечной вершиной пути. Следовательно, пути $S_1(x_1, x_5)$ и $S_2(x_1, x_5)$ подобны.

Справедливо следующее утверждение.

Теорема 1. *Связный кратный граф $G(X, E)$ является деревом тогда и только тогда, когда для любой пары вершин $x \in X$ и $y \in X$ выполнено одно из двух условий ($P(x, y)$ – множество всех кратных путей из x в y):*

1) $P(x, y) = \varnothing$;

2) $P(x, y) = \{S_1(x, y), \ldots, S_p(x, y)\}$, при этом $S_i(x, y)$ подобен $S_j(x, y)$ для всех $i \neq j$.

Доказательство.

Необходимость. Пусть $G(X, E)$ – кратное дерево. Предположим противное: в графе $G(X, E)$ существуют две различные вершины x и y такие, что между ними можно построить кратные пути $S_1(x, y)$ и $S_2(x, y)$, не являющиеся подобными.

Это означает, что в путях $S_1(x, y)$, $S_2(x, y)$ обязательно найдутся непустые участки $S_1'(x_0, y_0) \subseteq S_1(x, y)$ и $S_2'(x_0, y_0) \subseteq S_2(x, y)$, не содержащие одинаковых ребер. Эти участки могут быть как кратными (содержать кратные и/или мультиребра), так и некратными (содержать только обычные ребра). x_o может совпадать с x, y_0 – с y.

В любом случае $S_1'(x_0, y_0)$, $S_2'(x_0, y_0)$ представляют собой кратные пути из x_0 в y_0. Граф неориентированный, поэтому, обратив $S_2'(x_0, y_0)$, мы получим путь $S_2'(y_0, x_0)$. Теперь у нас есть путь из x_0 в y_0 и путь из y_0 в x_0, не содержащие общих ребер. Объединение $S_1'(x_0, y_0)$ и $S_2'(y_0, x_0)$ дает кратный цикл, что приводит к противоречию.

Достаточность. Пусть для всех пар вершин выполнено условие теоремы. Предположим противное: граф $G(X, E)$ не является кратным деревом.

Это означает, что в графе $G(X, E)$ существует кратный цикл C, содержащий не менее трех вершин. Пусть вершины $x_0 \in C$, $y_0 \in C$. Тогда кратный цикл C можно представить в виде

$$C = S_1(x_0, y_0) \cup S_2(y_0, x_0),$$

где S_1, S_2 – кратные пути. При этом очевидно, что найдется ребро e такое, что $e \in S_1(x_0, y_0)$ и $e \notin S_2(y_0, x_0)$ (иначе $C = \varnothing$). Граф неориентированный, поэтому, обратив $S_2(y_0, x_0)$, мы получим путь $S_2(x_0, y_0)$. Поскольку $e \notin S_2(y_0, x_0)$, то $e \notin S_2(x_0, y_0)$. Следовательно, пути $S_1(x_0, y_0)$ и $S_2(x_0, y_0)$ не являются подобными, что приводит к противоречию.

Теорема доказана.

Следствие 1. *Связный делимый граф является деревом тогда и только тогда, когда для любой пары вершин $x \in X$ и $y \in X$ существует не более одного пути $S(x, y)$.*

Отметим, что в любом кратном дереве, как и в обычном, заведомо есть $|X| - 1$ обычное или связанное ребро. Однако количество ребер не фиксировано и может быть больше.

Пусть $T(X, E)$ – делимое кратное дерево кратности k. Обозначим через n_i количество вершин в частях T_i ($i \in \overline{1,k}$) этого дерева. Зная количество мультиребер в дереве, мы сможем оценить общее количество обычных и связанных ребер в нем.

Теорема 2. *В кратном делимом дереве количество обычных и связанных ребер* $c = |E^o| + k|E^k| + k|E^m|$ *оценивается по формуле*

$$\sum_{i=1}^{k} n_i - k \le c \le \sum_{i=1}^{k} n_i - k + k(|E^m| - 1)$$

при условии, что $|E^m| \ge 1$.

Доказательство. Сначала докажем, что

$$\sum_{i=1}^{k} n_i - k \le c.$$

Делимое кратное дерево является связным кратным графом, следовательно, каждая его часть T_i ($i \in \overline{1,k}$) – связный обычный граф. Значит, каждая из частей G_i содержит не менее $n_i - 1$ ребер, откуда следует требуемое соотношение.

Докажем теперь, что

$$c \le \sum_{i=1}^{k} n_i - k + k(|E^m| - 1).$$

Пусть n'_0 – количество вершин, инцидентных кратным ребрам или являющихся общим концом мультиребер. В дереве $T(X, E)$ между ними может быть не более чем $n'_0 - 1$ кратное ребро ($k(n'_0 - 1)$ связанных ребер). В противном случае мы получим цикл по кратным ребрам. Пусть n'_i – количество вершин в части T_i ($i \in \overline{1,k}$), инцидентных обычным ребрам или являющихся отдельными концами мультиребер. Между ними может быть не более чем $n'_i - 1$ обычное ребро. В противном случае мы получим цикл по обычным ребрам. Заметим, что $n'_0 + n'_i = n_i$. Тогда

$$c \le k(n'_0 - 1) + k|E^m| + \sum_{i=1}^{k}(n'_i - 1) = \sum_{i=1}^{k} n_i - k + k(|E^m| - 1),$$

что и требовалось доказать.

Теорема доказана.

Отметим, что случай $|E^m| = 0$, не рассмотренный в теореме, является вырожденным – кратное дерево, не содержащее мультиребер, по сути, эквивалентно обычному дереву.

Также отметим, что при $|E^m| = 1$ оценка для c обращается в равенство.

3. Остовные деревья в кратном графе. Критерий полноты остовного дерева в делимом графе

Пусть $G(X, E)$ – связный кратный граф.

Определение 11. Остовным деревом *в кратном графе* $G(X, E)$ *называется кратное дерево* $T(X, E')$, *для которого* $E' \subseteq E$.

Заметим, что в остовном дереве заведомо будут существовать кратные пути $S(x, y)$ для всех вершин x, y, инцидентных кратным ребрам или являющихся общим концом мультиребер. Однако, если хотя бы одна из вершин x, y инцидентна обычному ребру или отдельному связанному ребру мультиребра, существование пути $S(x, y)$ не гарантировано даже в том случае, когда такой путь существует в исходном графе $G(X, E)$.

Определение 12. *Остовное дерево* $T(X, E')$ *в кратном графе* $G(X, E)$ *является* полным, *если для любой пары вершин* $x \in X$, $y \in X$ *кратный путь* $S_T(x, y)$ *в дереве* $T(X, E')$ *существует тогда и только тогда, когда существует кратный путь* $S_G(x, y)$ *в исходном графе* $G(X, E)$.

Определение 13. Множеством достижимости по обычным ребрам *для некоторой вершины* x *назовем множество* R^o_x *всех вершин* y *таких, что существует путь из* x *в* y, *проходящий только по обычным ребрам.*

Множество достижимости по обычным ребрам определяется только для вершин, не инцидентных кратным ребрам и не являющихся общим концом мультиребра.

Теорема 3. *Пусть* $T(X, E')$ – *остовное дерево в делимом кратном графе* $G(X, E)$. *Остовное дерево* $T(X, E')$ *является полным тогда и только тогда, когда на каждом множестве достижимости по обычным ребрам* R^o_x *графа* $G(X, E)$ *построено обычное дерево* $T^o(R^o_x, E^o_x)$, *где* $E^o_x \subseteq E'$, $E^o_x \subseteq E^o$.

Доказательство. По теореме 3 статьи [1] (критерий существования кратного пути между двумя вершинами), учитывая связность графа $G(X, E)$, получаем, что в графе $G(X, E)$ могут существовать кратные пути $S_G(x, y)$ одного из двух видов:

1) вершины x и y инцидентны кратным ребрам или являются общим концом мультиребер;

2) вершины x и y инцидентны обычным ребрам, и путь $S_G(x, y)$ проходит только по обычным ребрам, то есть $y \in R^o_x$.

Отметим, что если для пары вершин x, y в графе $G(X, E)$ существует путь $S_G(x, y)$ первого типа, то существование пути $S_T(x, y)$ в остовном дереве $T(X, E')$ обеспечивается его связностью и не зависит от полноты дерева.

Пути второго типа существуют в исходном графе тогда и только тогда, когда вершина $y \in R^o_x$.

Поэтому для полноты дерева $T(X, E')$ необходимо и достаточно, чтобы в этом дереве вершины каждого множества достижимости R^o_x исходного графа могли быть связаны путями, проходящими только по обычным ребрам. Это возможно только в том случае, когда на множестве R^o_x построен связный обычный граф $T^o(R^o_x, E^o_x)$,

где $E_x^o \subseteq E'$ и $E_x^o \subseteq E^o$. При этом $T^o(R_x^o, E_x^o)$ обязательно является деревом (иначе возникнут циклы по обычным ребрам).

Теорема доказана.

Отметим, что условие теоремы справедливо только для делимых графов. Для произвольного кратного графа теорема 3 не будет верна. Рассмотрим соответствующие примеры.

Пример 3.

Пусть имеется кратный граф G кратности 2, изображенный на рис. 2.

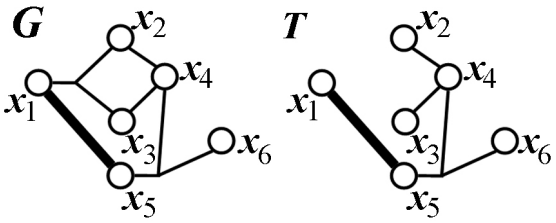

Рис. 2. Остовное дерево в кратном графе
Fig. 2. Spanning tree in a multiple graph

Заметим, что этот граф не является делимым, поскольку существует путь из вершины x_2 в вершину x_3, проходящий только по обычным ребрам, и при этом данные вершины являются концами одного мультиребра.

Нетрудно убедиться, что граф T является остовным деревом для G. Поскольку в графе G существует путь $S(x_1, x_5)$, а в графе T – нет, остовное дерево T не является полным. Однако условие теоремы 3 выполнено: в исходном графе G два множества достижимости по обычным ребрам – $R^o_{x_2} = \{x_2, x_3, x_4\}$ и $R_{x_6} = \{x_6\}$; на каждом из этих множеств в графе T построено обычное дерево. Таким образом, для произвольного кратного графа построение обычных деревьев на множествах достижимости R_x^o не является достаточным для полноты остовного дерева.

Отметим также, что в данном примере единственно возможное полное остовное дерево будет совпадать с графом G.

Пример 4.

Пусть теперь имеется кратный граф G, показанный на рис. 3.

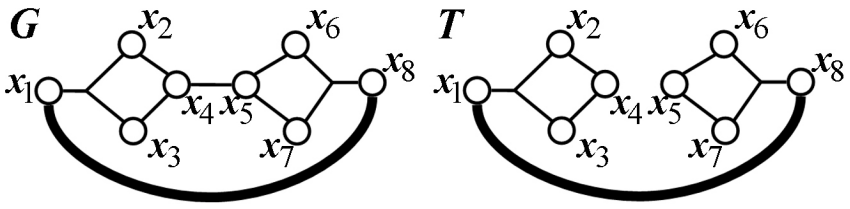

Рис. 3. Полное остовное дерево в кратном графе
Fig. 3. Complete spanning tree in a multiple graph

Граф G не является делимым, кроме того, в нем имеются кратные пути между каждой парой вершин.

Удаляя обычное ребро $\{x_4, x_5\}$, мы получим полное остовное дерево T. При этом в исходном графе было единственное множество достижимости по обычным ребрам $R_{x_2} = \{x_2, x_3, x_4, x_5, x_6, x_7\}$, на котором в остовном дереве T построен несвязный обычный граф. Следовательно, для произвольного кратного графа построение обычных деревьев на множествах достижимости R^o_x не является необходимым для полноты остовного дерева.

Отметим, что если бы мы в исходном графе G вместо обычного ребра $\{x_4, x_5\}$ удалили кратное ребро $\{x_1, x_8\}$ или любое из мультиребер, то мы все равно бы получили полное остовное дерево (при этом на множестве R_{x_2} было бы построено обычное дерево). Любое другое остовное дерево в данном графе не будет полным.

Независимо от того, делимый граф или нет, проверка полноты остовного дерева выполняется за полиномиальное число шагов.

Действительно, поиск множеств достижимости в кратном графе полиномиален (алгоритм 2 статьи [1]). Для делимого графа остается проверить, что в кратном дереве на каждом множестве достижимости построено обычное дерево, что может быть сделано с помощью стандартных полиномиальных алгоритмов для обычных графов.

Для произвольного кратного графа нужно проверить наличие путей между всеми возможными парами вершин. Затем для тех пар, где пути существуют, проверить наличие путей в остовном дереве. Количество пар вершин полиномиально, а каждую проверку можно осуществить с помощью полиномиальных алгоритмов из статьи [1].

4. Задача о минимальном остовном дереве в кратном графе. Алгоритм для делимых графов

Определение 14. *Целочисленная функция $l(e)$, определенная для всех ребер $e \in E$, является* длиной (весом) ребра *в кратном графе $G(X, E)$, если выполнено следующее:*

1) $l(e) > 0$ для любого ребра e;

2) если e является кратным или мультиребром, то $l(e_1) = l(e_2) = \ldots = l(e_k)$ и $l(e) = k \cdot l(e_1)$, где $e_1, \ldots, e_k$ – это связанные ребра данного ребра e.

Тогда *вес кратного графа $G(X, E)$* будет определяться по формуле

$$w(G(X, E)) = \sum_{e \in E} l(e).$$

Поставим две задачи о минимальном остовном дереве.

Задача 1 (минимальное остовное дерево)**.** *В кратном графе $G(X, E)$ требуется найти такое остовное дерево $T^{\min}(X, E')$, что для любого остовного дерева $T(X, E'')$ выполнено*

$$w(T^{\min}(X, E')) \leq w(T(X, E'')).$$

Задача 2 (минимальное полное остовное дерево)**.** *В кратном графе $G(X, E)$ требуется найти такое полное остовное дерево $T^{\min}_{complete}(X, E')$, что для любого полного остовного дерева $T_{complete}(X, E'')$ выполнено*

$$w(T^{\min}_{complete}(X, E')) \le w(T_{complete}(X, E'')).$$

Больший интерес здесь представляет задача 2 ввиду того, что в полном остовном дереве есть кратные пути между всеми парами вершин, для которых есть кратные пути в исходном графе. При этом очевидно, что

$$w(T^{\min}(X, E')) \le w(T^{\min}_{complete}(X, E'')).$$

Существование остовного дерева в связном кратном графе очевидно, поэтому задача 1 всегда разрешима. Разрешимость задачи 2 для делимых графов устанавливается следующей теоремой.

Теорема 4. *В связном делимом кратном графе $G(X, E)$ всегда существует полное остовное дерево $T(X, E')$.*

Доказательство. Если в графе $G(X, E)$ нет обычных ребер ($E^o = \varnothing$), то любое остовное дерево является полным (следует из определения связного кратного графа и остовного дерева).

Пусть в графе $G(X, E)$ множество $E^o \neq \varnothing$.

Для каждого множества R^o_x исходного графа, где вершина x инцидентна обычному ребру, построим обычное дерево $T^o_x(R^o_x, E^o_x)$. Здесь $E^o_x \subseteq E^o$. Будем считать каждое такое дерево единой *квазивершиной* T_i ($i \in \overline{1, q}$). Сформируем множество всех квазивершин $Q = \{T_1, \ldots, T_q\}$ и построим кратный граф $G_Q(X_Q, E_Q)$, где $X_Q = (X \cup Q) \setminus (\cup R^o_x)$, $E_Q = E^k \cup E^m_Q$. Множество E^m_Q получается из множества E^m заменой каждого мультиребра $\{a, \{a_1, \ldots, a_k\}\}$ на мультиребро $\{a, \{T_{i_1}, \ldots, T_{i_k}\}\}$ таким образом, что $a_j \in T_{i_j}$ для всех $j \in \overline{1, k}$. В силу делимости исходного графа $i_j \neq i_p$, если $j \neq p$.

Следовательно, мы получили связный делимый кратный граф $G_Q(X_Q, E_Q)$ без обычных ребер. Значит, в нем любое остовное дерево $T_Q(X_Q, E'_Q)$ является полным. Выполняя в дереве T_Q обратную замену всех квазивершин T_i на обычные деревья T^o_x, мы получим остовное дерево $T(X, E')$ для исходного графа $G(X, E)$. При этом для дерева $T(X, E')$ выполнено условие теоремы 3, следовательно, оно является полным.

Теорема доказана.

Пусть $G(X, E)$ – делимый кратный граф. Рассмотрим следующий эвристический алгоритм для задачи 2. Несмотря на то, что в нем обобщается известный алгоритм Краскала (см. [11]), решение не всегда будет точным. Более того, можно предположить, что задачи 1 и 2 для произвольных кратных графов NP-полны.

Алгоритм.

1. С помощью алгоритма Краскала строим минимальные остовные деревья $T^o_x(R^o_x, E^o_x)$ для каждого множества достижимости по обычным ребрам R^o_x исходного графа $G(X, E)$.

2. Будем считать каждое построенное дерево T^o_x *квазивершиной* T_i. Сформируем множество всех квазивершин $Q = \{T_1, \ldots, T_q\}$ и построим кратный граф

$G_Q(X_Q, E_Q)$, где $X_Q = (X \cup Q) \setminus (\cup R_x^o)$, $E_Q = E^k \cup E_Q^m$ (E_Q^m получается из E^m так же, как в доказательстве теоремы 4). Этот кратный граф не содержит обычных ребер.

3. Будем искать полное остовное дерево $T_Q(X_Q, E'_Q)$ в графе $G_Q(X_Q, E_Q)$ следующим образом.

3.1. Все ребра из E_Q сортируем по возрастанию весов и объявляем непросмотренными. Устанавливаем $E'_Q = \varnothing$.

3.2. Возьмем очередное непросмотренное ребро $e \in E_Q$. Если таких ребер нет, переходим на шаг 3.7.

3.3. Если e – кратное ребро и в графе $T_Q(X_Q, E'_Q)$ нет кратного пути из одного конца ребра e в другой, то добавляем e в E'_Q и переходим на шаг 3.5.

3.4. Если e – мультиребро вида $\{a, \{T_{i_1}, \ldots, T_{i_k}\}\}$ и в графе $T_Q(X_Q, E'_Q)$ нет мультиребра $\{b, \{T_{i_1}, \ldots, T_{i_k}\}\}$ такого, что существует кратный путь $S(a, b)$, то добавляем e в E'_Q.

3.5. Ребро e становится просмотренным.

3.6. Если $T_Q(X_Q, E'_Q)$ – связный граф, переходим на шаг 3.7. Иначе переходим на шаг 3.2.

3.7. Сортируем ребра из E'_Q по убыванию весов и объявляем их непросмотренными.

3.8. Если $|E'_Q| = \sum\limits_{i=1}^{k} n_i - k$ (n_i – количество вершин в частях $(G_Q)_i$ графа $G_Q(X_Q, E_Q)$), переходим на шаг 4.

3.9. Возьмем очередное непросмотренное ребро $e \in E'_Q$. Если таких ребер нет, переходим на шаг 4.

3.10. Исключаем ребро e из E'_Q, если это можно сделать, не нарушив связности дерева $T_Q(X_Q, E'_Q)$. Ребро e становится просмотренным. Переходим на шаг 3.8, если исключение проводилось, и на шаг 3.9, если нет.

4. Меняем в дереве $T_Q(X_Q, E'_Q)$ все квазивершины T_i на деревья, полученные на шаге 1. В итоге получается полное остовное дерево.

Отметим, что проверка связности кратного графа и проверка существования кратного пути между двумя вершинами выполняются с помощью полиномиальных алгоритмов, рассмотренных в статье [1]. Таким образом, каждый отдельный шаг алгоритма полиномиален. При этом на шаге 1 выполняется поиск $|Q|$ деревьев, шаги 3.2–3.6 выполняются не более $|E_Q|$ раз, а шаги 3.8–3.10 – не более $|E'_Q|$ раз. Остальные шаги выполняются однократно. Следовательно, алгоритм полиномиален.

Также стоит заметить, что алгоритм может быть применен и в случае, когда граф $G(X, E)$ не является делимым (шаг 3.8 в этом случае пропускается), однако тогда результирующее дерево не обязательно будет полным.

Заключение

Данная статья продолжает исследование кратных неориентированных графов, начатое в [1]. Мы ввели понятие кратного дерева и поставили задачи о минимальном и минимальном полном остовном дереве в кратном графе. Была обоснована оценка количества ребер в кратном делимом дереве и доказан критерий полноты остовного дерева в делимом кратном графе.

Кроме того, был предложен эвристический алгоритм поиска минимального полного остовного дерева в делимом графе. Этот алгоритм обобщает известный алгоритм Краскала и является полиномиальным.

Список литературы / References

[1] Смирнов А. В., "Задача о кратчайшем пути в кратном графе", *Моделирование и анализ информационных систем*, **24**:6 (2017), 466–478; [Smirnov A. V., "The Shortest Path Problem for a Multiple Graph", *Modeling and Analysis of Information Systems*, **24**:6 (2017), 788–801, (in Russian).]

[2] Cormen T. H., Leiserson C. E., Rivest R. L., Stein C., *Introduction to Algorithms*, 3rd ed., The MIT Press, McGraw-Hill Book Company, 2009.

[3] Berge C., *Graphs and Hypergraphs*, North-Holland Publishing Company, 1973.

[4] Basu A., Blanning R. W., "Metagraphs in workflow support systems", *Decision Support Systems*, **25**:3 (1999), 199–208.

[5] Basu A., Blanning R. W., *Metagraphs and Their Applications*, Integrated Series in Information Systems, **15**, Springer US, 2007.

[6] Рублев В. С., Смирнов А. В., "Потоки в кратных сетях", *Ярославский педагогический вестник*, **3**:2 (2011), 60–68; [Rublev V. S., Smirnov A. V., "Flows in Multiple Networks", *Yaroslavsky Pedagogichesky Vestnik*, **3**:2 (2011), 60–68, (in Russian).]

[7] Smirnov A. V., "The Problem of Finding the Maximum Multiple Flow in the Divisible Network and its Special Cases", *Automatic Control and Computer Sciences*, **50**:7 (2016), 527–535.

[8] Ford L. R., Fulkerson D. R., *Flows in Networks*, Princeton University Press, 1962.

[9] Рублев В. С., Смирнов А. В., "Задача целочисленного сбалансирования трехмерной матрицы и алгоритмы ее решения", *Моделирование и анализ информационных систем*, **17**:2 (2010), 72–98; [Roublev V. S., Smirnov A. V., "The Problem of Integer-Valued Balancing of a Three-Dimensional Matrix and Algorithms of Its Solution", *Modeling and Analysis of Information Systems*, **17**:2 (2010), 72–98, (in Russian).]

[10] Smirnov A. V., "Network Model for the Problem of Integer Balancing of a Four-Dimensional Matrix", *Automatic Control and Computer Sciences*, **51**:7 (2017), 558–566.

[11] Kruskal J. B., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", *Proceedings of the American Mathematical Society*, **7**:1 (1956), 48–50.

Smirnov A. V., "The Spanning Tree of a Divisible Multiple Graph", *Modeling and Analysis of Information Systems*, **25**:4 (2018), 388–401.

DOI: 10.18255/1818-1015-2018-4-388-401

Abstract. In this paper, we study undirected multiple graphs of any natural multiplicity $k > 1$. There are edges of three types: ordinary edges, multiple edges and multi-edges. Each edge of the last two types is a union of k linked edges, which connect 2 or $k + 1$ vertices, correspondingly. The linked edges should be used simultaneously. If a vertex is incident to a multiple edge, it can be also incident to other multiple edges, and it can be the common ending vertex to k linked edges of a multi-edge. If a vertex is the common end of some multi-edge, it cannot be the common end of any other multi-edge. Special attention is paid to the class of divisible multiple graphs. The main peculiarity of them is a possibility to divide the graph into k parts, which are adjusted on the linked edges and which have no common edges. Each part is an ordinary graph. The definition of a multiple tree is stated and the basic properties of such trees are studied. Unlike ordinary trees, the number of edges in a multiple tree is not fixed. In the article, the evaluation of the minimum and maximum number of edges in the

divisible tree is stated and proved. Next, the definitions of the spanning tree and the complete spanning tree of a multiple graph are given. The criterion of completeness of the spanning tree is proved for divisible graphs. It is also proved that a complete spanning tree exists in any divisible graph. If the multiple graph is weighted, the minimum spanning tree problem and the minimum complete spanning tree problem can be set. In the article, we suggest a heuristic algorithm for the minimum complete spanning tree problem for a divisible graph.

Keywords: multiple graph, multiple tree, divisible graph, spanning tree, complete spanning tree, minimum spanning tree

On the authors:
Alexander V. Smirnov, orcid.org/0000-0002-0980-2507, PhD, Associate Professor
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003, Russia, e-mail: alexander_sm@mail.ru

Acknowledgments:
This work was supported by the Russian Foundation for Basic Research under the Grant No 17-07-00823 A.

Моделирование и анализ информационных систем. Т. 25, №4 (2018), с. 402–410
Modeling and Analysis of Information Systems. Vol. 25, No 4 (2018), pp. 402–410

Технология блокчейн

DOI: 10.18255/1818-1015-2018-4-402-410

УДК 51-37

О некоторых подходах к решению задачи «Useful Proof-of-work for blockchains»

Дурнев В. Г., Мурин Д. М., Соколов В. А., Чалый Д. Ю.

получена 30 июля 2018

Аннотация. Технология блокчейн основана на принципе доказательства работой «Proof-of-work». Суть данного принципа состоит в том, что некоторое событие (например, перевод денежных средств с одного счета на другой) становится значимым только после того, как оно подтверждено определенным объемом вычислительной работы. Соответственно возникает потребность в вычислительных задачах, над которыми такую работу можно производить, причем на решение этих задач будет тратиться практически вся вычислительная мощность блокчейн-сети. На сегодня в качестве таких задач получили распространение «хэш-головоломки» – задачи поиска битовой строки с хэшем, удовлетворяющим определенным условиям. Существенным недостатком хэш-головоломок является отсутствие у них какого-либо полезного применения за пределами технологии блокчейн. В работе описываются подходы к решению задачи «Useful Proof-of-work for blockchains», а именно предлагается рассматривать в качестве вычислительных задач для доказательства работой возникающие на практике индивидуальные представители NP-полных задач, которые могут решаться, например, SAT- или LLL-решателями. Отдельной проработки требует вопрос об использовании FPT-задач. Предлагаемый подход позволяет обеспечить следующие свойства вычислительных задач для доказательства работой: полезность, управляемость сложностью задач (через изменение размерности, выбор задач определенного вида, указание точности необходимого решения), массовость. При этом допускается, что не каждая решенная задача может оказаться полезной, однако предоставляется возможность решать с помощью технологии блокчейн задачи, возникающие на практике. Кроме прочего, таким образом становится возможным сопоставить стоимость виртуальной криптовалюты через затраты электроэнергии при ее генерации с практическим результатом от решения вычислительных задач. Наиболее трудными вопросами в контексте рассматриваемого подхода являются реализация связи событий и задач, обеспечивающих эти события вычислительной работой, и реализация системы анализа сложности задач. Статью следует воспринимать как программу исследований, поскольку многие технические детали требуют отдельной проработки.

Ключевые слова: доказательство работой, блокчейн, выполнимость, SAT-решатель, NP-полнота, FPT, алгоритм

Для цитирования: Дурнев В. Г., Мурин Д. М., Соколов В. А., Чалый Д. Ю., "О некоторых подходах к решению задачи «Useful Proof-of-work for blockchains»", *Моделирование и анализ информационных систем*, **25**:4 (2018), 402–410.

Об авторах:
Дурнев Валерий Георгиевич, д-р физ.-мат. наук, профессор,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: Durnev@uniyar.ac.ru

Мурин Дмитрий Михайлович, orcid.org/0000-0002-8068-0784, канд. физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: nirum87@mail.ru

Соколов Валерий Анатольевич, orcid.org/0000-0003-1427-4937, д-р физ.-мат. наук, профессор,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: valery-sokolov@yandex.ru

Чалый Дмитрий Юрьевич, orcid.org/0000-0003-0553-7387, канд. физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: chaly@uniyar.ac.ru

Введение

С момента появления основополагающей статьи Сатоши Накомото [1], блокчейн является динамично развивающейся технологией [2], чему во многом способствовал взрывной рост популярности криптографических валют, созданных на основе этой технологии.

Непосредственно блокчейн представляет собой структуру криптографически связанных между собой данных, которые традиционно называются блоками. Блоки создаются на вычислительных устройствах, которые объединяются в блокчейн-сеть и содержат служебную информацию, в том числе ссылку на предыдущий блок, значение, получающееся в вершине дерева Меркла, в листах которого формируются значения на основе событий, привязываемых к этому блоку, а также специальное поле, предназначенное для размещения решения задачи специального вида.

Одним из основных примитивов при обеспечении безопасности в технологии блокчейн выступают криптографические хэш-функции. На их основе строятся так называемые «хэш-головоломки» – задачи поиска битовой строки с хэшем, удовлетворяющим определенным условиям. Решается хэш-головоломка следующим образом: для блока задаются значения всех полей за исключением специального, а значения специального поля перебираются (итерируются) до тех пор, пока значение хэш-функции от служебной информации блока не будет удовлетворять заданным условиям. Традиционное условие, которому должно удовлетворять значение хэш-функции, – это наличие определенного числа нулей в начале хэш-свертки. Задавая число нулей, можно управлять сложностью решаемой хэш-головоломки таким образом, чтобы число блоков, создаваемых в единицу времени, было в среднем одинаково.

Хэш-головоломки обладают следующими важными свойствами:

1) решение хэш-головоломки позволяет криптографически связать служебную информацию блока;

2) легкость создания;

3) управляемость сложностью.

К недостаткам хэш-головоломок относится отсутствие у них какого-либо полезного применения за пределами технологии блокчейн, скорее даже их полная бесполезность. Поэтому возникает потребность в создании технологии блокчейн, в которой для обеспечения работой используются полезные и интересные задачи.

Помимо хэш-головоломок существует большое число трудноразрешимых задач, представляющих с практической точки зрения существенно больший интерес. К таким задачам, безусловно, относятся NP-полные задачи, в том числе выполнимость

булевой формулы, на основе которой предложены наиболее эффективные на настоящий момент программные приложения для решения представителей NP-полных задач – CDSL SAT-решатели (SAT-solver) [3,4], задачи теории решеток, находящие свое применение в теории управления, радиосвязи и кристаллографии, задача о рюкзаке, находящая применение в логистике, и многие другие. Кроме того, есть целый класс FPT-задач с параметризованной сложностью.

Задача «Useful Proof-of-work for blockchains» сформулирована на международной студенческой олимпиаде по криптографии 2017 года [5] и на момент публикации статьи считается нерешенной [6]. Необходимо, однако, отметить, что в статье [7] предложен как минимум один из возможных вариантов решения задачи в поставленной на олимпиаде формулировке.

Итак, задача «Useful Proof-of-work for blockchains» состоит в создании задачи P, которая может быть использована для систем с доказательством работой таким образом, что информация, полученная в процессе решения, может быть использована за пределами системы. Более формально:

1. P – это семейство задач, параметризированных двумя переменными: I (входные данные, например, 256-битовая строка) и C (сложность, например, некоторое положительное целое число).

2. Для фиксированного входа и сложности $P(I, C)$ является проблемой, которая может быть решена с помощью некоторого алгоритма A. Невозможно найти доказуемое решение задачи $P(I, C)$, если I неизвестно.

3. Среднее время T (количество шагов или итераций вычислений), необходимое для нахождения решения $P(I, C)$ с использованием алгоритма A, известно (при условии, что входные данные I выбраны случайным образом и равномерно) и зависит от C, поэтому $T = T(C)$ и $T(C)$ могут быть сделаны очень маленькими, неимоверно большими или чем-то промежуточным, регулируя переменную сложности C.

4. Должно быть легко проверить, правильно ли выполнено любое предоставленное решение.

5. Является желательным любое доказательство того, что, вероятно, не существует значительно лучших алгоритмов для решения P, чем данный алгоритм A.

6. Необходимо описать, как информация, полученная в процессе решения P, может быть полезна вне системы доказательства работой.

Исходя из формулировки задачи, можно заключить, что приоритет при построении задачи P отдается все же тем свойствам, которые позволяют использовать эту задачу в качестве замены традиционных хэш-головоломок. При этом полезность предлагаемой задачи является косвенной, поскольку из существования I' и C', таких что $P(I', C')$ является практически полезной задачей, не следует, что доля таких параметров существенна, а также что выбор задач, обеспечивающих события вычислительной работой, будет в пользу этих параметров.

Мы предлагаем решать эту задачу, исходя из приоритета полезности решаемых задач. С этой точки зрения, ответы на вопросы 4–6 становятся очевидными. Если рассматривать в качестве P множество NP-полных задач, то частое появление их на практике дает ответ на вопрос 6. Для задач общего положения лучшие результаты в решении показывают на практике SAT-решатели, которые могут претендовать на роль алгоритма A. И, в силу принадлежности классу NP, решение задач легко

проверить на корректность. При этом технологии блокчейн, базирующиеся на NP-полных задачах, в доступной авторам литературе не обнаружены.

Остаются три первых вопроса, которые можно свести к двум задачам: первая состоит в том, чтобы связать решаемую задачу с блоком (в исходной формулировке за это отвечают входные данные I), вторая задача состоит в организации управления сложностью задач с целью обеспечения равномерной генерации блоков в условиях колебания вычислительной мощности блокчейн-сети (за это отвечает параметр сложности C, от которого зависит функция T).

1. Каким образом получать задачи, решение которых может оказаться полезным

Хорошо известен факт, что решение индивидуальных представителей NP-полных задач представляет существенную ценность как для практики, так и для развития некоторых теорий [8]. Ввиду того, что представители NP-полных задач достаточно часто возникают в различных областях знаний [9], мы считаем, что сбор задач, возникающих на практике, является более перспективным, чем попытка создания задач, решение которых только потенциально может найти практическое применение. Однако в случае нехватки гарантированно полезных задач допускается использование генераторов задач, «близких» к полезным, что, впрочем, приведет к вопросам о том, насколько много действительно полезных задач и, в связи с этим, насколько правомерна поставленная в работе цель.

Реализация сбора задач может быть организована по-разному. Мы видим два принципиальных варианта: либо задачи публикуются в самой цепочке блокчейна (в качестве смарт-контрактов или транзакций особого вида, связанных с одним из блоков), либо набор задач, имеющих практическое значение, создается вне цепочки блокчейна отдельным сервисом, создающим и поддерживающим базу данных задач. И в первом, и во втором случае должна формироваться база данных задач, позволяющая осуществлять выбор задач на основании оценки среднего времени для нахождения решения.

В первом случае, предполагая технологию блокчейн открытой, мы допускаем раскрытие задач, после чего они могут быть решены вне нашей блокчейн-сети, а решения впоследствии могут быть использованы для формирования блоков нашей блокчейн цепочки. Во втором случае такого раскрытия можно не допустить, но это приведет к вопросу о том, кто будет являться владельцем базы данных задач, и может привести к централизации управления, для устранения которого создавалась технология блокчейн. Мы считаем, что при большом числе задач, предназначенных для решения, их публикация не будет являться критичной по следующим причинам:

1) нецелесообразно тратить вычислительные ресурсы на решение всех задач подряд, в то время как их можно тратить на решение выбранных задач с целью получения премии за создание блока;

2) при большом числе задач вероятность выбрать для решения задачу, которая потом встретится в блокчейне, мала;

3) кроме решения задачи, необходимо также обеспечить привязку задачи к блоку и событиям, то есть нужно осуществить подбор событий таким образом, чтобы

значение хэш-функции от случайной информации блока являлось указателем на решенную задачу, что само по себе является трудной задачей.

Отметим, что в первом случае публикация задач в блокчейне может не преследовать цели обеспечения событий вычислительной работой. Например, за решение задач может предлагаться оплата.

Решенные задачи, после создания последующих блоков, могут удаляться из базы данных или помечаться как архивные, с публикацией задачи и решения и направлением решения поставившему задачу лицу или сервису.

Здесь необходимо сказать еще несколько слов о конфиденциальности. Многие полезные для практики представители NP-полных задач могут возникать в рамках коммерческих разработок. Сами задачи и их решения могут являться коммерческой тайной. Для закрытых (корпоративных) блокчейн проектов [10, 11] это не является существенной проблемой, однако невозможность использовать более широкие возможности открытых блокчейн проектов может вызывать некоторые неудобства. Поэтому мы хотим обратить внимание, что при нашем подходе возможно применение различных методов «маскирования» исходной задачи. К таким методам можно отнести, например, полиномиальное сведение одной задачи к другой, использование методов, близких к доказательствам с нулевым разглашением и другие. В качестве классического примера можно привести известную задачу о изоморфизме графа, используемую для иллюстрации доказательств с нулевым разглашением. Если у организации есть необходимость найти гамильтонов путь в большом графе, то она может обратиться к блокчейн-сети, предоставив ей запрос на решение этой задачи для графа, изоморфного тому, который представляет интерес. Изоморфный граф легко построить, осуществив перестановку вершин исходного графа. При этом исходный граф не будет опубликован.

2. Об управлении сложностью задач

По мере поступления задач и формирования базы данных необходимо обеспечить их классификацию, сведение к базовой задаче, например, выполнимости конъюнктивной нормальной формы, и в дальнейшем выбор задач для обеспечения событий вычислительной работой. Необходимость сведения к базовой задаче возникает для унификации решающего алгоритма A, в качестве которого мы в первую очередь рассматриваем SAT-решатели. Такое сведение позволяет применять один алгоритм ко всем индивидуальным представителям NP-полных задач. При этом естественно допустить другой подход: существование разных блокчейн проектов, которые при обеспечении событий вычислительной работой ориентируются на определенные задачи и не принимают к рассмотрению задачи другого вида. Этот подход потребует наличия источника полезных на практике задач определенного вида и применения к ним специализированного алгоритма, при этом подходе управление сложностью задач может быть более строгим.

Стоит обратить внимание на класс разрешимых с фиксированным параметром задач – класс FPT (Fixed-Parameter Tractable) [12, 13].

Определение 1. *Параметризированная задача* Π *принадлежит классу FPT, если*

она может быть решена некоторым параметризированным алгоритмом за время $t(n,k) = O(n^{O(1)} \cdot f(k))$ *для функции* f*, зависящей только от параметра* k.

Для FPT-задач естественно использовать параметр k для управления сложностью, при этом для фиксированного k FPT-задачи могут быть решены за полиномиальное время. Это позволяет рассматривать FPT-задачи в качестве хорошего примитива для обеспечения событий вычислительной работой.

Управление сложностью задач должно основываться на предварительной обработке множества доступных к решению задач. Полагая, что база данных задач сформирована и имеется определенная практика решения задач блокчейн-сетью, необходимо провести анализ задач по следующим направлениям:

1) классификация задачи (оптимизации, вычисления, распознавания; что собственно за задача представлена к решению, можем ли мы отнести задачу к какому-то особому виду);

2) размерности исходной задачи и базовой задачи, к которой была сведена исходная;

3) точность решения, которую следует обеспечить;

4) имеющаяся практика решения задачи блокчейн-сетью;

5) имеющаяся практика решения блокчейн-сетью близких по характеристикам задач.

Результатом анализа должно являться множество задач, которые блокчейн-сеть может решить за определенное в среднем время, зависящее от числа блоков, которые необходимо создавать за единицу времени.

Выбор задачи из данного множества может быть осуществлен, например, следующим образом: выбрав события, которые мы хотим обеспечить вычислительной работой, и сформировав блок, можно вычислить значение хэш-функции от служебной информации блока. Полученное таким образом значение хэш-функции может являться указателем на задачу, которую необходимо решить для обеспечения блока работой. Кроме прочего, таким образом может быть обеспечена связь между служебной информацией блока и информацией о задаче, обеспечивающей события, связанные с блоком, вычислительной работой.

Таким образом, мы переходим к следующему разделу, в котором обсудим вопрос связи событий и задач, обеспечивающих эти события вычислительной работой.

3. О связи событий и задач, обеспечивающих эти события вычислительной работой

В этом разделе мы опишем два способа, которыми можно связать служебную информацию блока, информацию о событиях и информацию о задаче, обеспечивающей события вычислительной работой.

Первый способ близок к традиционному блокчейну и использует хэш-головоломки. Мы можем использовать дерево Меркла событий и служебную информацию блока для выбора задачи из базы данных задач (например, как описано в предыдущем разделе). Далее можно включить выбранную задачу в дерево Меркла, а ее решение в служебную информацию блока, после чего в соответствии с традиционным блокчейном решить хэш-головоломку для блока в такой конфигурации.

Решение хэш-головоломки в этом случае решает традиционную задачу связывания служебной информации блока, информации о событиях и информации о задаче, обеспечивающей события вычислительной работой. Блок, полученный таким образом, можно назвать «тяжелым», поскольку на его формирование необходимо потратить больше вычислительных ресурсов за счет решения прикрепленной к блоку задачи, чем на обычный «легкий» блок. Поскольку на формирование тяжелого блока должно тратиться больше вычислительных ресурсов, то, естественно, возникает необходимость мотивировать участников блокчейн-сети к созданию таких блоков. Мы видим две основных возможности для этого: с одной стороны, можно увеличить вознаграждение за формирование тяжелых блоков, а с другой, можно, в отличие от традиционной блокчейн технологии, при которой участники всегда считают истинной самую длинную версию цепочки и работают над ее удлинением, считать истинной самую длинную версию цепочки, содержащей больше всего тяжелых блоков. Можно положить «вес» тяжелого блока равным трем «весам» легких блоков. Соответственно за генерацию тяжелого блока должно осуществляться трехкратное вознаграждение, и цепочка из одного тяжелого блока должна быть эквивалентна цепочке из трех легких блоков. Отметим, что на настоящий момент событие принимается как достоверное в блокчейне криптовалюты Биткойн после того, как будут созданы шесть блоков, следующие за блоком, к которому привязано событие. Таким образом, потенциальный отказ от трех легких блоков в пользу одного тяжелого не является, с нашей точки зрения, критичным для функционирования технологии блокчейн.

Соответственно тяжелый блок становится выгодно создавать, если затраты времени на его создание в среднем меньше времени, затрачиваемого на создание трех легких блоков. Следовательно, мы должны выбирать сложность задач таким образом, чтобы обеспечить их решение в течение времени, затрачиваемого в среднем на создание двух блоков. Кроме прочего, это дает дополнительный механизм для управления сложностью задач: при оценке сложности задач можно ввести дополнительный коэффициент, который указывал бы число раз, при которых задача выбиралась для решения, но решение которой не было получено из-за приращения альтернативной ветви блокчейна.

Второй способ отличен от традиционного блокчейна и предполагает, что при формировании блока в него должны быть записаны решение задачи, опубликованной в предыдущем блоке, и задача, решить которую требуется для формирования следующего блока. Выбор задачи, как и ранее, может осуществляться с помощью хэш-функции, значение которой на входе из служебной информации блока можно рассматривать как указатель на задачу (например, это может быть первая нерешенная задача в базе данных, k-битовый префикс которой совпадает с k-битовым префиксом значения хэш-функции). Следует обратить внимание, что во втором способе, как и в традиционном блокчейне, возможно формирование альтернативных цепочек по следующим причинам:

1) различный выбор задач участниками блокчейн-сети;

2) получение участниками блокчейн-сети различных решений одной и той же задачи.

В блоке может публиковаться не одна, а несколько задач для решения (например, первые пять нерешенных задач в базе данных, k-битовый префикс которых

совпадает с k-битовым префиксом значения хэш-функции), что породит еще одну возможность для создания альтернативной цепочки.

Как и в традиционном блокчейне, участники должны считать истинной самую длинную версию цепочки, «тяжелых» блоков в этом случае не возникает.

4. Заключение

В работе предложены подходы к решению задачи «Useful Proof-of-work for blockchains», которые базируются на полезности решаемой задачи. Данная постановка приводит к определенным трудностям в вопросе связывания в одном блоке служебной информации, информации о событиях и информации о задаче, обеспечивающей события вычислительной работой. В статье дается описание способов преодоления этих трудностей. В качестве вычислительных задач для доказательства работой предлагается рассматривать индивидуальные представители NP-полных задач (альтернативным вариантом может выступать рассмотрение FPT-задач). Задачи, полезность которых обеспечивается возникновением на практике, должны передаваться в специальную базу данных. Выбор задач для решения должен осуществляться в соответствии с описанными в статье подходами. В качестве алгоритмов их решения рассматриваются SAT-решатели. Управление сложностью задач предлагается реализовывать через изменение размерности, выбор задач определенного вида, указание точности необходимого решения, анализ опыта блокчейн-сети по решению задач со схожими параметрами. Многие вопросы, затронутые в статье, требуют проведения экспериментальных исследований, запланированных авторами.

Список литературы / References

[1] Nakamoto S., “Bitcoin: A Peer-to-Peer Electronic Cash System”, 2009, 1–9. https://bitcoin.org/bitcoin.pdf.

[2] Buterin V., “Ethereum White Paper: A next-generation smart contract and decentralized application platform”, 2014. https://github.com/ethereum/wiki/wiki/White-Paper.

[3] Marques-Silva J.Ṗ., Sakallah K.Ȧ., “GRASP: A new search algorithm for satisfiability”, *ICCAD ’96 Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, 1996, 220—227.

[4] Bayardo Jr.Ṙ., Schrag R., “Using CSP look-back techniques to solve real-world SAT instances”, *AAAI’97/IAAI’97 Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, 1997, 203—208.

[5] “International Students’ Olympiad in Cryptography NSUCRYPTO. Useful Proof-of-work for blockchains”, 2017, 12–13. https://nsucrypto.nsu.ru/archive/2017/round/2/section/0/task/11.

[6] “International Students’ Olympiad in Cryptography NSUCRYPTO. Unsolved problems”, 2018. https://nsucrypto.nsu.ru/unsolved-problems/.

[7] Ball M., Rosen A., Sabin M., Vasudevan P. N., *Proofs of Useful Work*, 2017. https://eprint.iacr.org/2017/203.pdf.

[8] Garey M. R., Johnson D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co, San Francisco, Calif., 1979.

[9] Cormen T. H., Leiserson C. E., Rivest R. L., Stein C., *Introduction to Algorithms (third ed.)*, MIT Press, 2009.

[10] BitFury Group, "Public versus Private Blockchains. Part 1: Permissioned Blockchains. White Paper", 2015, 1–23 (совм. с Garzik J.). https://bitfury.com/content/downloads/public-vs-private-pt1-1.pdf.

[11] BitFury Group, "Public versus Private Blockchains. Part 2: Permissionless Blockchains. White Paper", 2015, 1–20 (совм. с Garzik J.). https://bitfury.com/content/downloads/public-vs-private-pt2-1.pdf.

[12] Downey R., Fellows M., *Parameterized complexity*, Springer Verlag, New York, 1999.

[13] Flum J., Grohe M., *Parameterized complexity theory*, Springer Verlag, Berlin, Heidelberg, 2006.

Durnev V. G., Murin D. M., Sokolov V. A., Chalyy D. Ju. , "On Some Approaches to the Solution of the Problem «Useful Proof-of-work for Blockchains»", *Modeling and Analysis of Information Systems*, **25**:4 (2018), 402–410.

DOI: 10.18255/1818-1015-2018-4-402-410

Abstract. The blockchain technology is based on the "Proof-of-work" principles. The essence of this principle is that some event (for example the bill-to-bill money transaction) becomes significant after the confirmation by a certain computer work. So, a demand arose for such computational problems to work on, and we will spend on it about the whole blockchain system computing capacity. Now the main kind of such a problem is a hash-puzzle – the problem to find a bit string with a hash that satisfies some conditions. The important hash-puzzle weakness is the lack of the useful application outside of the blockchain technology. In this work, we offer some approaches to "Useful Proof-of-work for blockchains" problem, namely, consider some practical variants of the NP-complete problems that could be solved with the help of SAT or LLL-solvers as the Proof-of-Work computational problems. The use of the FPT-problems requires special study. The offered approach allows to provide the following characteristics of the proof-of-work computational problems: usefulness, problems complexity management (through the dimension change, choosing problems of certain kind, the indication of necessary solution precision), mass character. Herewith we admit that not every solved problem can be useful but we consider the opportunity to solve some practical problems with the help of the blockchain technology. Among other things it is also possible to compare the virtual crypto-currency value (through the energy costs spent) and the effective result of the practical problems solution. The most complicated points of the described approach are the realization of the events-problems (providing the computer work for these events) relations and the realization of the problems complexity analysis system. This issue should be viewed as the study program because of many technical details that must be worked out further.

Keywords: proof-of-work, blockchain, satisfiability, SAT-solver, NP-complete, FPT, algorithm

On the authors:
Valeriy G. Durnev, Doctor, Professor,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: Durnev@uniyar.ac.ru

Dmitry M. Murin, orcid.org/0000-0002-8068-0784, PhD, Docent,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: nirum87@mail.ru

Valery A. Sokolov, orcid.org/0000-0003-1427-4937, Doctor, Professor,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: valery-sokolov@yandex.ru

Dmitry Ju. Chalyy, orcid.org/0000-0003-0553-7387, PhD, Docent,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: chaly@uniyar.ac.ru

Моделирование и анализ информационных систем. Т. 25, № 4 (2018), с. 411–420
Modeling and Analysis of Information Systems. Vol. 25, No 4 (2018), pp. 411–420

Вопросно-ответные системы

DOI: 10.18255/1818-1015-2018-4-411-420

УДК 517.9

Вопросно-ответная система для поддержки абитуриентов с использованием современных мессенджеров

Филонов Д. Р., Чалый Д. Ю.[1], Мурин Д. М., Дурнев В. Г., Соколов В. А.[1]

получена 31 июля 2018

Аннотация. В настоящее время растет интерес пользователей к приложениям для мгновенного обмена сообщениями, мессенджерам. Они позволяют не только общаться с другими пользователями, но и включают в себя функционал, помогающий создавать автоматических собеседников, автоматизирующих отдельные бизнес-процессы либо удовлетворяющих информационные потребности пользователей. В статье рассматривается узкоспециализированная вопросно-ответная система, которая использует инфраструктуру, предоставляемую современными мессенджерами для обмена сообщениями и предназначенную для информационной поддержки абитуриентов, поступающих в университет. В процессе разработки системы был накоплен корпус характерных вопросов абитуриентов, а также разработана модель, которая позволяет осуществлять поиск близких вопросов по этому корпусу. При этом абитуриент может формулировать свои вопросы на естественном языке без изучения предварительно заданных шаблонов или специальных правил построения сообщений. Для получения ответов на вопросы, которых нет в корпусе системы, привлекаются специалисты приемной комиссии, имеющие свой интерфейс. Система была реализована с использованием современных облачных технологий, предоставляемых компанией Amazon. Среди них бессерверные (serverless) вычисления и NoSQL-базы данных. Для этого была разработана архитектура сервиса, удовлетворяющая этой модели вычислений. Поскольку система может оперировать с чувствительными данными, включая персональные данные, а также предоставлять персонализированный сервис, были проанализированы методы обеспечения безопасности системы, а также подходы к авторизации пользователей. В настоящее время система проходит тестирование и оценку используемых алгоритмов информационного поиска.

Ключевые слова: вопросно-ответная система, мессенджер, информационный поиск, облачные системы и сервисы, безопасность

Для цитирования: Филонов Д. Р., Чалый Д. Ю., Мурин Д. М., Дурнев В. Г., Соколов В. А., "Вопросно-ответная система для поддержки абитуриентов с использованием современных мессенджеров", *Моделирование и анализ информационных систем*, **25**:4 (2018), 411–420.

Об авторах: Филонов Дмитрий Русланович, orcid.org/0000-0002-6402-5114, магистр,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: frbdima@gmail.com

Чалый Дмитрий Юрьевич, orcid.org/0002-0007-1896-0876, канд. физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: chaly@uniyar.ac.ru

Мурин Дмитрий Михайлович, orcid.org/0000-0002-8068-0784, канд. физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: nirum87@mail.ru

Дурнев Валерий Георгиевич, д-р физ.-мат. наук, профессор,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: durnev@uniyar.ac.ru

Соколов Валерий Анатольевич, orcid.org/0000-0003-1427-4937, д-р физ.-мат. наук, профессор,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: sokolov@uniyar.ac.ru

Благодарности:
[1] Работа выполнена при поддержке проекта «Моделирование и анализ информационных систем» (инициативный проект ЯрГУ № АААА-А16-116070610022-6).

Введение

Поиск информации является одной из базовых потребностей человека в современном информационном обществе. Начиная с середины 60-х годов [2, 8] развивались системы, перед которыми ставились задачи обработки текстов на естественном языке с целью выделения в них смысловых единиц, которые потом использовались для удовлетворения информационных потребностей пользователей. Одним из видов таких систем являются вопросно-ответные системы, которые в настоящее время стали перспективным направлением исследований [3, 6, 7]. При этом выделяются системы [8], основанные на подходах, используемых в информационном поиске (IR, information retrieval) [1], а также системы, отвечающие на вопросы, используя знания о фактах, скажем, статистического характера. Первая парадигма, IR-системы, полагается на то, что существует большой корпус документов, в котором содержится ответ на вопрос пользователя. Поиск ответа во многом похож на то, как работают поисковые системы: вопрос пользователя преобразуется в запрос для поисковой системы с учетом контекста, а система выдает ранжированный список документов, из которых извлекается ответ. Вторая парадигма представляет вопрос в виде логического предиката, а ответ – значение свободной переменной этого предиката, которая делает его истинным.

В нашей работе мы применяем подходы, характерные для IR-парадигмы вопросно-ответных систем. Своей задачей мы ставим разработку узкоспециализированной системы для удовлетворения информационных потребностей абитуриентов, поступающих в университет. Поскольку большинство абитуриентов делают это первый раз, то у них возникает много вопросов, касающихся процесса поступления, формальных требований к абитуриентам, текущей конкурсной ситуации, информации о процессе обучения, культурной, спортивной и творческой жизни в университете. Для поиска ответов на эти вопросы можно использовать сайт университета, группы в соцсетях и другие информационные источники, включая общение с представителями приемной комиссии, а также выпускниками и студентами университета. Однако это требует существенных усилий и может приводить к получению недостоверной информации. Поэтому наличие ассистента, который способен находить ответы в достоверных источниках, могло бы существенно упростить процесс получения информации, включая персонализированную справку о текущей конкурсной ситуации. Это, в частности, поднимает вопросы безопасности, включая то, каким образом должен авторизоваться абитуриент, как хранить чувствительную информа-

цию в анонимизированном виде и как безопасно передавать ее по коммуникационным сетям. В нашей работе мы обобщаем и развиваем подход, который реализуется в работе [4]. При этом мы включаем членов приемной комиссии как экспертов, которые могут знать ответы на те вопросы, которые еще не внесены в систему. Это становится возможным благодаря интерфейсам, существующим в современных мессенджерах.

1. Модель вопросно-ответной системы

Целью вопросно-ответной системы, которая основывается на подходах, характерных для информационного поиска, является поиск кратких текстовых фрагментов в корпусе документов, которые являются ответами на вопрос пользователей. В контексте таких систем подобные ответы еще называют фактоидами. Наш опыт показывает, что существенная часть вопросов абитуриентов имеет именно такой характер. Согласно Тейлору [2] все информационные запросы можно разделить в зависимости от их четкости на следующие категории:

1. «Идеальный вопрос», который пользователь не может сформулировать реально, но присутствующий в его подсознании неоформленно; осознанное отношение к информационной потребности, но в неопределённой форме (при этой степени понимания индивид может обратиться к знающему человеку, который поможет ему оформить запрос).

2. Вопрос исследователя, осознающего, что он ищет и в каком виде.

3. Вопрос в таком виде, в каком он представлен в информационной системе.

Вопросно-ответные системы на первом шаге пытаются определить тип вопроса и вычленить смысловые сущности, которые должен содержать ответ. Наша система предназначена прежде всего для решения случая 3 и в какой-то степени случая 2, когда вопрос содержит термины, прямо сигнализирующие об информационной потребности пользователя. Для решения вопросов, которые относятся к категории 1, мы включаем во взаимодействие внутри информационной системы представителей приемной комиссии.

Важным компонентом рассматриваемого типа вопросно-ответных систем является корпус документов, по которым производится поиск ответов. В течение приемных кампаний 2016 и 2017 года в группе ВКонтакте для абитуриентов факультета ИВТ производилась активная работа по информированию абитуриентов и ответам на возникающие вопросы. Поскольку ответы публиковались в формате один пост – один ответ, это позволило накопить базу вопросов и ответов на них. Все такие публикации были скачаны автоматизированным образом, используя API ВКонтакте. В результате была сформирована база из вопросов (а также соответствующих им ответов) в количестве 141. Эти данные являются базой знаний нашей системы. Их отличает краткость — вопросы чаще всего содержат небольшое количество терминов. Наша экспертная оценка, которая основывается на повторяемости вопросов в каждый год приема, показывает, что данная база вопросов содержит существенную долю информационных потребностей, в которых ответы являются фактоидами.

Каждый вопрос рассматривается в виде «мешка слов» и при помощи модели векторного пространства [1] представляется в виде многомерного вектора. Это одна из фундаментальных моделей информационного поиска, которая позволяет осуществлять поиск документов по запросу, ранжирование, классификацию и кластеризацию документов [1]. Использование этой модели возможно и для вопросно-ответных систем [5]. Размерность векторного пространства определяется мощностью словаря терминов коллекции документов. Рассмотрим подход, который мы используем для преобразования вопросов на естественном языке в вектор.

Сначала из вопроса убираются стоп-слова, т.е. такие термины, которые являются общеупотребительными и не несут существенной смысловой нагрузки (например, предлоги). Координаты преобразованного в виде вектора документа можно представить несколькими способами. Так, i-я координата может быть равна:

1. Значению «истина», если i-й термин словаря присутствует в вопросе, и «ложь» в противном случае.

2. Частоте вхождения i-го термина в документ.

3. Частоте вхождения i-го термина в документ, умноженной на обратную документную частоту. Документная частота равна количеству документов, содержащих i-й термин.

Для данной коллекции документов наиболее рациональным методом взвешивания координат векторов является использование только частот терминов. Домножать документы на их обратную частоту не целесообразно, так как тексты достаточно короткие и в них отсутствуют широкоупотребительные слова (а те, что встречаются, фильтруются списком стоп-слов). Использование исключительно логических значений также не рационально, так как некоторые термины могут встречаться несколько раз, поэтому частота термина является оптимальным показателем релевантности. Для снижения размерности векторов можно использовать стемминг или лемматизацию [1]. В итоге мы получаем набор векторов, которые моделируют известные системе информационные потребности.

Вопрос пользователя при помощи аналогичного подхода преобразуется в вектор, и при помощи косинусной меры сходства выбирается наиболее близкий известный системе вопрос. В качестве результата система выдает известный ей ответ на этот вопрос.

2. Архитектура вопросно-ответной системы

Разработка системы велась на языке программирования Python 3.6 с использованием фреймворка Serverless для разворачивания на сервисе бессерверных вычислений AWS Lambda [9]. Для хранения данных и обновления индекса системы используется база данных Amazon DynamoDB [10]. Все это позволяет реализовать масштабируемый и отказоустойчивый сервис.

AWS Lambda — это управляемая событиями (event-driven), бессерверная вычислительная платформа, предоставляемая в составе Amazon Web Services. Это вычислительная служба, которая запускает программный код в ответ на события

и автоматически управляет вычислительными ресурсами, которые требуются для выполнения этого кода.

Serverless — это архитектурный стиль, в котором приложения в значительной степени разрабатываются с использованием подходов Backend-as-a-Service (BaaS) и Functions-as-a-Service (FaaS). BaaS — это когда в клиентской части приложений используются сторонние удаленные сервисы. А FaaS — это когда в серверной части приложений логика представляет собой функции, запускаемые в облаке.

После загрузки программного кода сервис AWS Lambda обеспечивает все ресурсы, необходимые для его выполнения и масштабирования, с высокой степенью доступности. Можно настроить автоматический запуск программного кода из других сервисов AWS или непосредственно из любого мобильного или веб-приложения. При бессерверных вычислениях приложение по-прежнему работает на серверах, но AWS полностью берет на себя управление этими серверами. Таким образом, бессерверные приложения обладают тремя ключевыми преимуществами:

- отсутствие необходимости управлять серверами;
- гибкость масштабирования;
- автоматическое обеспечение высокой доступности.

Для хранения базы знаний, извлечения из нее промежуточных данных, пула «горячих» вопросов (т.е. таких, которые адресованы экспертам приемной комиссии) абитуриентов и таблицы с позициями экспертов в этом пуле используется нереляционная база данных Amazon DynamoDB, которая представляет собой быстрый и гибкий сервис баз данных NoSQL. Взамен привычных реляционных баз данных DynamoDB предлагает концепцию таблиц «ключ – значение», которые качественно отличаются от реляционного способа организации данных, так как используют менее строгие схемы описания данных. Каждый элемент (объект, запись) в таблице может отличаться от других количеством и типом атрибутов, при этом поддерживаются первичный и вторичный механизмы индексации, что позволяет избежать потерь в производительности запросов. Значения атрибутов таблицы могут быть строковыми, числовыми, двоичными, а также наборами значений. Размер одной записи не должен превышать 64 Кб. Так как единого интерфейса к NoSQL-продуктам не существует, пользователям необходим оригинальный интерфейс взаимодействия с этой СУБД. Поэтому в архитектуре проекта были развернуты два взаимодействующих уровня Data Access Layer (для реализации функций CRUD: Create, Read, Update, Delete) и уровня сервисов для работы с различного рода данными для каждой из таблиц проекта.

DynamoDB позиционируется как простая в эксплуатации СУБД для онлайновых систем, которые могут масштабироваться до тысяч запросов в секунду с сохранением короткого времени отклика. При этом она, конечно, не сравнится по функциональности с реляционными СУБД и не умеет выполнять сложные запросы, хотя поддерживается ряд атомарных операций, которые, например, изменяют значение конкретного числового поля в записи. Главное, DynamoDB позволяет максимально быстро развернуть онлайновую систему, которой требуется база с достаточно простой организацией данных.

Некоторые преимущества использования DynamoDB:

- быстрая и стабильная работа. Решение Amazon DynamoDB работает стабильно и быстро в системах любого масштаба в любой области применения. Среднее время обработки запроса на сервере составляет несколько миллисекунд. По мере роста объемов данных и повышения необходимой производительности система Amazon DynamoDB обеспечивает соответствие требованиям к пропускной способности и времени обработки запроса с помощью технологий автоматического разбиения на разделы и SSD в системах любого масштаба;

- высокая масштабируемость. При создании таблицы нужно просто указать требуемый объем запросов в единицу времени. Amazon DynamoDB выполняет все операции по масштабированию в скрытом режиме и в ходе их выполнения продолжает обеспечивать соответствие установленным требованиям к пропускной способности;

- гибкость. Amazon DynamoDB поддерживает работу со структурами данных на основе как документов, так и пар «ключ-значение», благодаря чему возможно выбрать оптимальную структуру базы данных с учетом особенностей разрабатываемой системы;

- событийно-ориентированное программирование: Amazon DynamoDB интегрируется с сервисом AWS Lambda для создания триггеров, которые позволяют разрабатывать приложения, автоматически реагирующие на изменение данных.

Общая архитектура сервиса изображена на рис. 1, она состоит из нескольких взаимодействующих уровней:

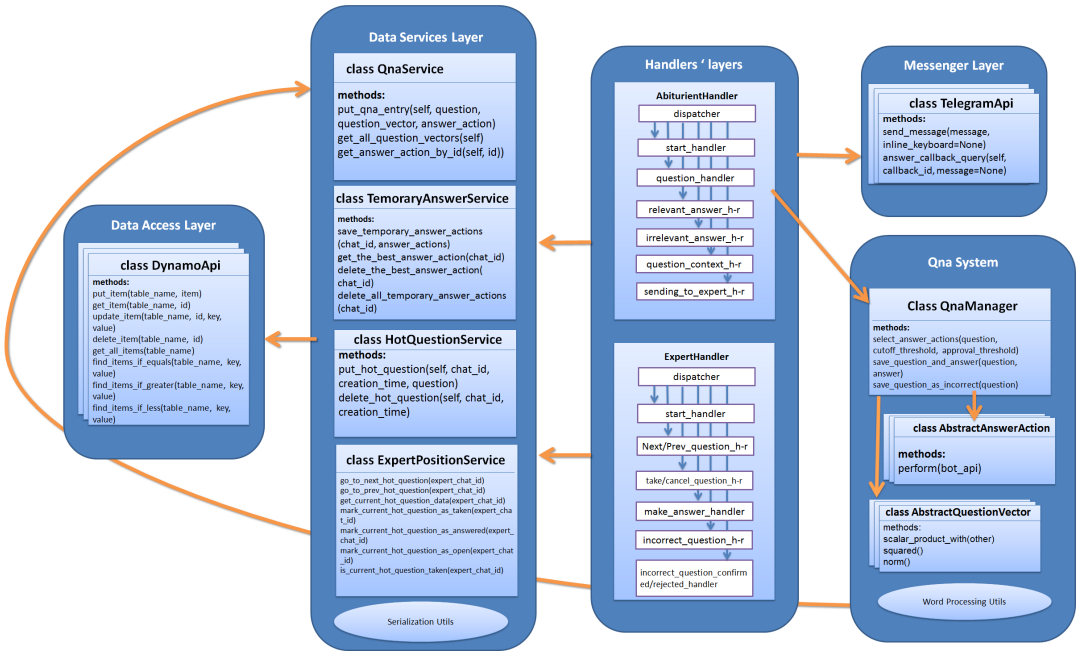

Рис. 1. Схема классов вопросно-ответной системы

Fig. 1. Question answering system class scheme

- уровень вопросно-ответной системы (*Qna System*). Здесь реализуется модель векторного пространства и поиск наиболее релевантных документов в ней. Для обработки естественного языка используются библиотеки NLTK [11] и pymorhy 2 [12]. Класс QnaManager реализует внешний интерфейс взаимодействия с базой знаний;

- уровень доступа к данным (*Data Access Layer*). Слой доступа к данным представлен набором классов, реализующих общий интерфейс взаимодействия с хранилищем данных. В настоящее время реализован класс DynamoAPI, взаимодействующий с нереляционной базой данных Amazon Dynamo DB;

- уровень сервисов для работы с данными (*Data Services Layer*) предназначен для работы с конкретными объектами вопросно-ответной системы: базой вопросов и ответов, очередью релевантных ответов, пулом вопросов пользователей, требующих ответа, а также навигации эксперта приемной комиссии по пулу вопросов абитуриентов, требующих ответа;

- уровень для работы с мессенджерами (*Messenger Layer*), который реализует функционал, требуемый для взаимодействия с конкретным мессенджером;

- уровень обработчиков событий системы (*Handlers' Layer*). Этот уровень реализован с использованием AWS Lambda и включает реакцию системы на различные события, например, получение вопроса от абитуриента, действия экспертов приемной комиссии и т.п.

3. Вопросы безопасности

Каждая система или сервис должны удовлетворять требованиям безопасности, исключающим утечку чувствительных данных. Проанализируем архитектуру разработанной системы с точки зрения безопасности. Система построена с помощью архитектуры «клиент – сервер», в которой клиентом является приложение-мессенджер, а сервер реализован при помощи бессерверного подхода с использованием сервисов Amazon. Безопасность клиентского приложения обеспечивается компанией-разработчиком мессенджера, которая должна своевременно проводить обновления, решающие проблемы безопасности. Серверная часть должна быть реализована с использованием принципов разделения ролей, авторизующих каждый используемый сервис. Это можно реализовать при помощи сервиса Amazon IAM. В нашей системе такой подход пока не реализован, и это является одним из направлений дальнейших исследований. Клиентская и серверная части взаимодействуют между собой через сеть Интернет по специализированному протоколу. Обеспечение корректности и безопасности работы такого протокола также является ответственностью разработчика мессенджера.

Разработанная система может предоставлять персонализированный сервис абитуриентам, однако вопросы авторизации и аутентификации не были нами рассмотрены при обсуждении архитектуры системы. Отсутствие таких возможностей позволяет пользователям получать доступ к закрытой информации, которая должна быть доступна только определенным адресатам. Кроме того, система не должна

хранить в явном виде персональные данные абитуриентов, поскольку ее работа происходит в облаке, принадлежащем сторонней организации. К таким персональным данным относятся, например, номер телефона или адрес электронной почты, которые сообщаются при подаче документов. Очевидно, что такие данные легко позволяют аутентифицировать пользователей. В самом мессенджере пользователи идентифицируются по присвоенным во время регистрации уникальным идентификаторам. Таким образом, процесс аутентификации должен сопоставлять этим идентификаторам реальных абитуриентов, причем без раскрытия их персональных данных.

Мы предлагаем использовать следующую схему аутентификации пользователей, которая исключает хранение в явном виде персональных данных пользователей. При загрузке информации о пользователях в облако рассчитывается значение криптографической хеш-функции (например, с помощью алгоритма MD5) от тех данных, которые известны только пользователю и приемной комиссии. Такими данными могут быть фамилия, имя, отчество, номер телефона, персональный идентификатор абитуриента, который назначается ему при подаче документов. Пользователь аутентифицируется, ответив на несколько вопросов, которые касаются этих данных. При помощи той же самой хеш-функции ответы пользователя преобразуются в его идентификатор, который потом сравнивается с базой загруженных идентификаторов. При совпадении с одним из существующих идентификаторов пользователь аутентифицируется.

Архитектура нашей системы содержит часть, которая должна быть доступна только специалистам приемной комиссии. Мы предлагаем изолировать эту часть в виде отдельного канала используемого мессенджера, к которому дается доступ только для тех учетных записей, которые принадлежат данной категории пользователей.

Рассмотренные подходы позволяют обеспечить безопасность чувствительных данных и предложить безопасное взаимодействие между системой и всеми категориями пользователей.

Заключение

В результате выполненных работ была создана автоматизированная вопросно-ответная система, которая позволяет отвечать на вопросы абитуриентов, сформулированные на естественном языке, а в отдельных случаях привлекать специалистов приемной комиссии. Архитектура системы обеспечивает безопасное взаимодействие, а также является масштабируемой, поскольку реализована при помощи облачных сервисов Amazon. Система включает в себя дополнительный уровень абстракции, который позволяет использовать различные мессенджеры, базы данных или методы извлечения знаний из неструктурированных данных, представленных на естественном языке.

Дальнейшее развитие системы предполагает всестороннее тестирование, проведение оценок качества информационного поиска, подключение новых мессенджеров.

Список литературы / References

[1] Маннинг К.Д., Рагхаван П., Шютце Х., *Введение в информационный поиск*, Вильямс, М., 2011; In English: Manning C.D., Raghavan P., Schutze H., *Introduction to Information Retrieval*, Cambridge University Press New York, NY, USA, 2008.

[2] Taylor R.S., "The Process of Asking Questions", *American Documentation*, **13**:4 (1962), 391–396.

[3] Radev D.R., Prager J., Samn V., "Ranking suspected answers to natural language questions using predictive annotation", *Proceedings of the sixth conference on Applied natural language processing*, 2000.

[4] Филонов Д.Р., Тупикин В.И., "Чат-бот для Telegram для помощи абитуриентам", *Заметки по математике и информатике*, 2017, 152–156; [Filonov D.R., Tupikin V.I., "Chat-bot dlya Telegram dlya pomoschi abiturientam", *Zametki po matematike i informatike*, 2017, 152–156, (in Russian).]

[5] Jovita L., Hartawan A., Suhartono D., "Using Vector Space Model in Question Answering System", *Procedia Computer Science*, **59** (2015), 305—311.

[6] Mishra A., Jain S.K., "A Survey on Question Answering Systems with Classification", *Journal of King Saud University – Computer and Information Sciences*, **28**:3 (2016), 345–361.

[7] Bouziane A., Bouchiha D., Doumi N., Malki M., "Question Answering Systems: Survey and Trends", *Procedia Computer Science*, **73** (2015), 366–375.

[8] Jurafsky D., Martin J.H., *Speech and Language Processing*, Prentice-Hall, NJ, USA, 2009.

[9] Amazon Web Services, *Serverless Computing and Applications*. https://aws.amazon.com/serverless/.

[10] Amazon Web Services, *Amazon DynamoDB*, https://aws.amazon.com/dynamodb/.

[11] Perkins J., *Python Text Processing with NLTK 2.0 Cookbook*, Packt Publishing, 2010.

[12] Korobov M., "Morphological Analyzer and Generator for Russian and Ukrainian Languages", *Analysis of Images, Social Networks and Texts*, 2015, 320–332.

Filonov D. R., Chalyy D. Ju., Murin D. M., Durnev V. G., Sokolov V. A., "Question Answering System for Applicant Support by Using Modern Messengers", *Modeling and Analysis of Information Systems*, **25**:4 (2018), 411–420.

DOI: 10.18255/1818-1015-2018-4-411-420

Abstract. There is an increasing interest to the instant messaging applications, messengers. These applications allow us to interact with other users and include a functionality that can help us to implement bots that automate various business processes or provide information services. In this paper, we consider a specialized question answering system that uses today's messaging services infrastructure to support university applicants. We gathered a corpus of applicants questions throughout two years and developed an information retrieval model that helps us to find similar questions in the corpus. Applicants can type their questions using a natural language without any formal requirements to phrase construction or using special templates. If the system is unable to find a relevant answer, the user can directly address the question to representatives of the university. The system was implemented with the use of modern cloud services that are provided by Amazon. We used serverless computations and NoSQL data bases, so we had to develop an architecture of the system in that way. Since the system contains sensitive personal data and provide personalized service, we must focus our attention on security. We proposed the means that must improve the safety of the system, more specifically, authentification process that can be used without the explicit use of personal data, however, this is a future work. At present we test our system and evaluate its quality of information retrieval.

Keywords: question answering system, messenger, information retrieval, cloud systems and services, security

On the authors:
Dmitry R. Filonov, orcid.org/0000-0002-6402-5114, Master,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: frbdima@gmail.com

Dmitry Ju. Chalyy, orcid.org/0002-0007-1896-0876, PhD,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: chaly@uniyar.ac.ru

Dmitry M. Murin, orcid.org/0000-0002-8068-0784, PhD,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: nirum87@mail.ru

Valery G. Durnev, Prof.,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: durnev@uniyar.ac.ru

Valery A. Sokolov, orcid.org/0000-0003-1427-4937, Prof.,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: sokolov@uniyar.ac.ru

Acknowledgments:
[1] This work was supported by initiative project of P.G. Demidov Yaroslavl State University «Modeling and Analysis of Information Systems» No AAAA-A16-116070610022-6.

Моделирование и анализ информационных систем. Т. 25, № 4 (2018), с. 421–434
Modeling and Analysis of Information Systems. Vol. 25, No 4 (2018), pp. 421–434

Динамические модели процессов

DOI: 10.18255/1818-1015-2018-4-421-434

УДК 004.94

Динамическая модель процессов информационных обменов в пиринговой сети

Кононова А. И.

получена 16 июля 2018

Аннотация. Рассматривается модель распространения файла в пиринговой файлообменной сети, построенная на основе обыкновенных дифференциальных уравнений. Определены фазовые переменные, описывающие состояние раздачи файла (в первом приближении — это количество пользователей — сидеров и личеров на раздаче), проанализированы факторы, влияющие на распространение файла и изменение количества пользователей, участвующих в обмене. На основе анализа записана система дифференциальных уравнений, описывающая эволюцию раздачи — динамическая модель эволюции раздачи. Рассмотрен жизненный цикл раздачи в файлообменной сети, состоящий из четырёх стадий — создание раздачи, быстрый набор личеров, стабилизация и (для раздач файлов, утрачивающих со временем актуальность) угасание. Каждой стадии соответствует своё соотношение параметров модели, причём со временем параметры изменяются. Описан процесс измерения состояния реальных раздач. Показан пример траектории, соответствующей эволюции реальной раздачи на крупном торрент-трекере. Далее рассматривается этап стабилизации раздачи, характеризующийся постоянными в первом приближении параметрами. Исследованы особые точки динамической модели эволюции раздачи, описано их возможное количество и тип. Описаны все конфигурации общего положения, возможные в модели эволюции раздачи в файлообменной пиринговой сети. Изображены фазовые портреты каждой конфигурации. Проанализировано влияние различных административных мер на запас устойчивости раздачи. Показана неоднозначность влияния системы учёта рейтинга на устойчивость раздач. Также показано положительное влияние системы таймбонусов, обратной связи и поглощения раздач.

Ключевые слова: файлообменные сети, обыкновенные дифференциальные уравнения, динамические системы, устойчивость

Для цитирования: Кононова А. И., "Динамическая модель процессов информационных обменов в пиринговой сети", *Моделирование и анализ информационных систем*, **25**:4 (2018), 421–434.

Об авторах:
Кононова Александра Игоревна, orcid.org/0000-0002-4178-3828, канд. техн. наук, доцент,
Национальный исследовательский университет «Московский институт электронной техники»,
пл. Шокина, 1, г. Москва, г. Зеленоград, 124498 Россия, e-mail: illinc@bk.ru

Введение

В настоящее время невозможно представить себе компьютер общего назначения, не подключённый к сети. Сеть необходима как для устойчивого функционирова-

ния самого компьютера (в частности, регулярного получения обновлений безопасности), так и для доступа пользователя к необходимому ему контенту. Причём любая достаточно крупная сеть характеризуется нерегулируемым ростом, описать который можно только статистически. В частности, широкое распространение получили файлообменные пиринговые сети, позволяющие распространять контент между пользователями, не создавая большую нагрузку на сервер [1].

Наиболее часто используемая характеристика любой сети — количество составляющих её узлов. Изменение этой характеристики может быть промоделировано при помощи динамических систем.

1. Моделирование раздачи

Рассмотрим распространение (раздачу) одного файла в файлообменной сети.

Узлы сети и, соответственно, пользователи, участвующие в пересылке этого файла, делятся на два вида с качественно разным поведением: обладающие полной копией файла и раздающие его (сидеры) и скачивающие файл (личеры).

Обозначим:

s — количество сидеров, раздающих файл *(seeders);*

l — количество личеров, скачивающих его *(leechers).*

Это и есть минимальный набор фазовых переменных [2, 3], необходимый для моделирования процессов, происходящих в подобной сети.

Нецелые значения s и l могут возникнуть в следующих ситуациях: либо ширина канала какого-либо пользователя меньше усреднённой, либо пользователь присутствует на раздаче не круглосуточно, но регулярно появляется по некоторому расписанию. И то и другое встречается довольно часто; во многих торрент-клиентах присутствует как функция расписания, так и ограничение скорости.

Обе величины s и l непрерывно изменяются во времени: личеры качают файл и становятся сидерами, к раздаче подключаются новые личеры, кроме того, любой пользователь может в любой момент уйти с раздачи.

1.1. Раздача и скачивание файла

Основные процессы в файлообменной сети — раздача файла сидерами и, соответственно, скачивание его личерами. При этом общее количество пользователей $s + l$ не изменяется. Личер, полностью скачавший файл, переходит в категорию сидеров.

Уравнения, описывающие этот процесс, выглядят следующим образом:

$$\begin{cases} \dot{s} &= C(s,l) \\ \dot{l} &= -C(s,l) \end{cases}, \tag{1}$$

где точка обозначает производную по времени, а $C(s,l)$ — количество докачавших в единицу времени при исходных s сидерах и l личерах.

Пусть ширина канала связи у различных личеров и различных сидеров одинакова. Тогда, очевидно, $C(s,l)$ пропорционально s — чем больше суммарный канал

раздачи, тем быстрее идёт скачивание. Также $C(s,l)$ пропорционально l — чем больше личеров, тем большее их количество станет сидерами. Казалось бы, они должны конкурировать за раздачу, но файлообменные протоколы построены так, что любой личер раздаёт уже скачанную часть файла и, соответственно, не только создаёт нагрузку на сидеров, но и берёт её часть на себя.

Таким образом, в первом приближении эту величину можно представить как

$$C(s,l) = \gamma \cdot s \cdot l, \tag{2}$$

где $\gamma > 0$ характеризует пропускную способность каналов раздачи-скачивания.

1.2. Подключение к раздаче новых пользователей

Новые пользователи, только включающиеся в раздачу, ещё не имеют копии файла и становятся личерами. Их количество пропорционально количеству пользователей ресурса, заинтересованных в получении этого файла, но ещё не участвующих в раздаче, а также привлекательности самой раздачи.

Привлекательность раздачи определяется содержанием раздаваемого файла (обозначим эту зависимость коэффициентом α), а также количеством сидеров s на ней (из нескольких раздач одного файла в сопоставимом качестве выбирается та, где больше пользователей уже раздают файл).

Если обозначить общее количество пользователей, заинтересованных в данный момент в получении этого файла, как N (в первом приближении эту величину можно считать постоянной), то количество новых заинтересованных пользователей равно $N-l-s$, а подключение новых пользователей к раздаче можно выразить уравнением:

$$\dot{l} \;=\; \alpha \cdot s \cdot (N - l - s)\ , \tag{3}$$

где $\alpha > 0$ характеризует привлекательность данной раздачи, $N > 0$ — общее количество заинтересованных в ней пользователей. Необходимо отметить, что данная оценка притока новых личеров корректна только при $s < N$. Действительно, при $l + s > N$ правая часть (3) отрицательна. При большом значении l и $s < N$ отрицательное значение притока личеров можно трактовать как уход с раздачи тех, кто не входит в целевую аудиторию раздачи и начал скачивание по ошибке, но при $s > N$ и малом l это не так (отрицательная величина добавляется к l, хотя превышает размер аудитории s).

1.3. Отток пользователей с раздачи

Кроме подключения к раздаче новых пользователей присутствует и отток уже участвующих в раздаче. Сидеры уходят с раздачи, если считают, что её уже не обязательно поддерживать, то есть если личеров мало, а сидеров достаточно много. Личеры отключаются от раздачи, если не могут скачать файл, то есть в раздаче участвует мало пользователей (здесь рассматривается общее количество пользователей $s + l$, а не только сидеров; множество личеров при малом количестве сидеров нормально для новых раздач, так что такая ситуация не приводит к оттоку личеров).

Так как каждый пользователь принимает решение об уходе с раздачи независимо от других, общий отток пользователей пропорционален их количеству (s и l соответственно). Введём коэффициенты β_1, μ_1 и β_2, μ_2, характеризующие скорости оттока сидеров и личеров с раздачи, тогда отток может быть описан уравнениями:

$$\begin{cases} \dot{s} &= -\beta_1 \cdot s \cdot e^{-\mu_1 l} \\ \dot{l} &= -\beta_2 \cdot l \cdot e^{-\mu_2 (s+l)} \end{cases}, \tag{4}$$

где $\mu_{1,2} > 0$ характеризует то, при насколько малом количестве соответственно личеров и всех пользователей раздача признаётся неактивной; $\beta_{1,2} > 0$ — скорость оттока с неактивной раздачи соответственно сидеров и личеров.

Таким образом, модель оттока пользователей эквивалентна мягкой [4] модели радиоактивного распада, где период полураспада определяется активностью раздачи.

1.4. Модель, учитывающая все рассмотренные процессы

В реальной файлообменной сети одновременно происходит как скачивание (1), так и приток новых пользователей (3) и отток старых (4), так что её динамика описывается уравнениями

$$\begin{cases} \dot{s} &= C(s,l) - \beta_1 \cdot s \cdot e^{-\mu_1 l} \\ \dot{l} &= \alpha \cdot s \cdot (N - l - s) - C(s,l) - \beta_2 \cdot l \cdot e^{-\mu_2 (s+l)} \end{cases}, \tag{5}$$

где

$$C(s,l) = \gamma \cdot s \cdot l.$$

Параметры системы (5):

$\gamma > 0$ характеризует пропускную способность каналов раздачи-скачивания;

$\alpha > 0$ и $N > 0$ — параметры притока новых личеров (N — общее число пользователей ресурса, заинтересованных в файле, α характеризует привлекательность раздачи);

β_1, μ_1 и β_2, μ_2 — параметры оттока сидеров и личеров соответственно.

Выбором единиц измерения s и l можно любой из этих параметров сделать единицей, поэтому примем $\gamma = 1$ и далее будем рассматривать систему в виде

$$\begin{cases} \dot{s} &= s \cdot l - \beta_1 \cdot s \cdot e^{-\mu_1 l} \\ \dot{l} &= \alpha \cdot s \cdot (N - l - s) - s \cdot l - \beta_2 \cdot l \cdot e^{-\mu_2 (s+l)} \end{cases}, \tag{6}$$

где

$$\alpha > 0, N > 0, \beta_{1,2} > 0, \mu_{1,2} > 0.$$

Эта модель в некотором приближении описывает распространение файла в пиринговой сети. Она может быть заменена на «мягкую» модель, учитывающую изменение во времени её коэффициентов $\alpha, N, \beta_1, \beta_2, \mu_1, \mu_2$.

Необходимо помнить, что, кроме детерминированных воздействий, на реальную раздачу действуют также случайные возмущения, которые влияют как на смещение текущего состояния раздачи на фазовой плоскости, так и на мгновенные значения параметров $\alpha, N, \beta_1, \beta_2, \mu_1, \mu_2$.

2. Жизненный цикл раздачи

Жизненный цикл раздачи можно разделить на четыре основных этапа:

- создание;
- быстрый набор личеров;
- стабилизация;
- угасание.

Первые три присутствуют в жизненном цикле любой раздачи, последний — в раздаче файлов, актуальность которых падает со временем (новости, текущие номера периодических изданий, анонсы и т. п.).

Рассмотрим их подробнее.

1. При создании раздачи её сидирует только автор, а личеров нет. После создания ссылка на раздачу попадает в новости трекера, и начинается этап набора личеров.

2. После анонсирования раздачи в новостях к ней активно подключаются личеры, заинтересованные в раздаваемом файле и следящие за новостями. Этот процесс идёт быстрее раздачи-скачивания, особенно при большом размере файла и относительно узком канале автора. Отток пользователей на этом этапе вообще отсутствует.

   Таким образом, этап быстрого набора личеров характеризуется $\beta_1 = \beta_2 = 0$ (новая раздача, постоянно сидируемая автором, активна независимо от количества других пользователей) и высоким α (новости трекера привлекают повышенное внимание).

3. По мере того как ссылка на раздачу вытесняется с верхних строк новостей, приток новых личеров плавно снижается. Через какое-то время личеры узнают о раздаче из поиска, а не из новостей.

   Возникает отток. Через какое-то время, если сидеров достаточно, раздачу может покинуть автор.

   Характеристики притока и оттока определяются объёмом и настроением пользователей и в первом приближении могут считаться постоянными. Начинается этап стабилизации.

   Именно этот этап описывается моделью (6).

4. Если раздаваемый файл теряет актуальность, приток личеров (α и N) снижается до нуля, так что раздача под действием постоянного или усиливающегося оттока постепенно приходит к $s = l = 0$, даже если ранее было достигнуто стабильное ненулевое состояние (6).

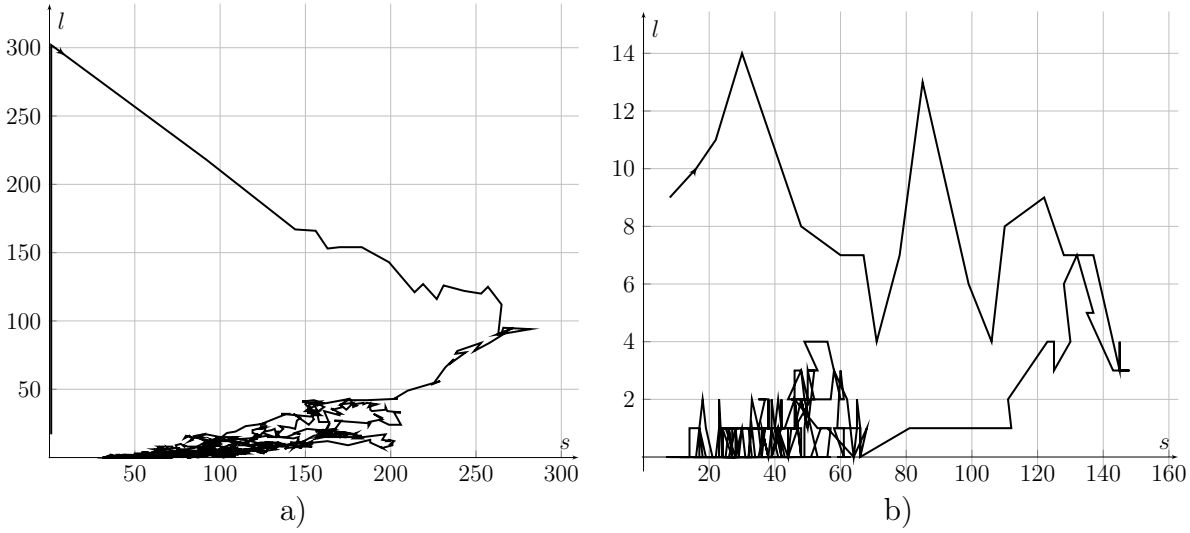

Рис. 1. Траектории раздач
Fig. 1. Experimental trajectories

Это хорошо прослеживается для экспериментальных данных. В частности, типичная траектория эволюции популярной раздачи на фазовой плоскости sl показана на рис. 1, a). Пример эволюции менее популярной — на рис. 1, b) (траектории таких раздач более разнообразны по форме, так как на них большее влияние оказывают случайные возмущения).

Измерения были начаты вскоре после создания раздачи, для рис. 1, a) — в точке $(1, 17)$. Отчётливо виден этап быстрого набора личеров (движение вдоль оси $0l$ до точки $(1, 302)$, после чего следует быстрый переход в $(92, 218)$, когда личеры, подключившиеся в начале раздачи, почти одновременно докачали файл и превратились в сидеров). После этого начинается этап стабилизации. Вначале преобладают процессы раздачи, на которые накладываются случайные возмущения и суточные колебания (многие пользователи появляются на раздаче по расписанию, что в модели отражено дробными значениями, а при измерениях — колебаниями). В точке $(260, 100)$ начинает активно проявляться отток сидеров. Направление траектории меняется. Движение замедляется, поэтому влияние суточных колебаний заметнее (на рис. 1, a) эти колебания приводят к тому, что измеренная траектория идёт плотным зигзагом).

В конце состояние раздачи колеблется от около $(30, 1)$ днём до около $(70, 3)$ ночью, то есть среднесуточное количество составляет около $(50, 2)$ (привести точное значение невозможно из-за постоянных случайных возмущений).

Набор личеров на рис. 1, b) происходит более плавно. Кроме того, на рис. 1, b) в самого начала чётко видны суточные колебания численности.

Некоторые раздачи через какое-то время после создания теряют всех пользователей и приходят в состояние $(0, 0)$, после чего уже не покидают этого состояния (вымирают). Иногда администрации трекера удаётся вернуть сидеров на вымершую раздачу; чаще через какое-то время подобная раздача удаляется из базы торрент-трекера (закрывается).

Далее рассматривается жёсткая модель (6), описывающая этап стабилизации раздачи. Соответственно начальное состояние раздачи имеет вид (s_0, l_0), где $s_0 \geqslant 0, l_0 \geqslant 0$ — значения, сформировавшиеся на этапе быстрого набора личеров.

3. Особые точки модели

Анализ особых точек динамической системы — ключевой момент её исследования [2, 5]. Особые точки системы (6) можно найти из системы уравнений

$$\begin{cases} s \cdot l - \beta_1 \cdot s \cdot e^{-\mu_1 l} & = & 0 \\ \alpha \cdot s \cdot (N - l - s) - s \cdot l - \beta_2 \cdot l \cdot e^{-\mu_2 (s+l)} & = & 0. \end{cases} \tag{7}$$

При любых значениях параметров решением (7) и, соответственно, особой точкой системы (6), очевидно, будет точка $s = l = 0$. Других вещественных решений при $s = 0$ система (7) не имеет (при $s = 0$ и $l \neq 0$ второе уравнение не обращается в верное равенство).

Если $s \neq 0$, из первого уравнения получаем $l - \beta_1 \cdot e^{-\mu_1 l} = 0$, или

$$\frac{l}{\beta_1} = e^{-\mu_1 l}. \tag{8}$$

Это трансцендентное уравнение при любых положительных параметрах β_1 и μ_1 имеет единственное решение l^*, лежащее в интервале $(0, \beta_1)$.

Подставляя l^* во второе уравнение системы (7), получаем

$$\alpha \cdot s \cdot (N - l^* - s) - s \cdot l^* - \beta_2 \cdot l^* \cdot e^{-\mu_2 (s+l^*)} = 0, \tag{9}$$

что можно преобразовать в

$$-\frac{\alpha}{\beta_2 l^* e^{-\mu_2 l^*}} \cdot s \left(s - \left(N - l^* \frac{\alpha + 1}{\alpha} \right) \right) = e^{-\mu_2 s}. \tag{10}$$

Обозначим для краткости $q = \frac{N - l^* \frac{\alpha+1}{\alpha}}{2}$ и $\kappa = \frac{\alpha}{l^* e^{-\mu_2 l^*}}$. Ни одна из величин q, κ не зависит от β_2. Тогда уравнение (10) примет вид:

$$-\frac{\kappa}{\beta_2} \cdot s(s - 2q) = e^{-\mu_2 s}. \tag{11}$$

При $q = 0$ уравнение (11) не имеет корней, а система (6), соответственно, не имеет ненулевых особых точек.

При $q \neq 0$ левая часть этого уравнения представляет собой параболу ветвями вниз (при положительных параметрах $\kappa > 0$), которая, в зависимости от соотношения β_2 и других параметров, может не пересекать график правой части, касаться его или пересекать в двух местах. Можно показать, что касание происходит при $\beta_2 = \beta_2^*$:

$$\beta_2^* = \frac{2\alpha}{l^*} \cdot \frac{\left(\sqrt{\mu_2^2 q^2 + 1} - 1 \right) \cdot e^{\mu_2 q + \sqrt{\mu_2^2 q^2 + 1}}}{\mu_2^2} \cdot e^{\mu_2 l^* - 1}. \tag{12}$$

Соответственно, при $\beta_2 < \beta_2^*$ уравнение (11) имеет два решения s_1^* и s_2^*, причём они оба лежат в интервале от 0 до $2q < N$, при $\beta_2 = \beta_2^*$ — одно решение $s_1^* = s_2^* = s^*$, при $\beta_2 > \beta_2^*$ оно не имеет ни одного вещественного решения.

Таким образом, система (6) может иметь от одной до трёх особых точек. Философия общего положения [6] описывает устойчивые к малому изменению параметров случаи одной особой точки $(0, 0)$ и трёх особых точек $(0, 0), (s_1^*, l^*), (s_2^*, l^*)$. Переходный случай с двумя особыми точками $(0, 0), (s^*, l^*)$ неустойчив и при малом изменении параметров переходит в один из случаев общего положения. Соответственно, сама точка (s^*, l^*) в таком случае вырождена.

Можно показать, что начало координат $(0, 0)$ всегда является притягивающим узлом. При наличии ещё двух особых точек $(s_1^*, l^*), (s_2^*, l^*), s_1^* < s_2^*$ и расположении их в первой четверти координатной плоскости (это возможно при $q > 0$) точка (s_1^*, l^*) — седло, (s_2^*, l^*) — притягивающий обобщённый узел (классический узел или фокус). При $q < 0$, то есть расположении ненулевых особых точек во второй четверти (s_1^*, l^*) — отталкивающий обобщённый узел, (s_2^*, l^*) — седло (то есть ближайшая к началу координат ненулевая особая точка — всегда седло).

4. Фазовый портрет модели

Рассмотрим фазовый портрет системы уравнения (6).

Хотя физически реализуемы только неотрицательные значения s и l (а модель адекватна при $s < N$), уравнения (6) определены для любых вещественных значений. Соответственно, для полноты анализа рассчитано и отображено множество траекторий, включающих как положительные, так и отрицательные s и l.

На рис. 2 показаны траектории системы (6), которые качественно не различаются для разных значений параметров. Это траектории в значительном отдалении от особых точек, а также траектории, целиком лежащие на оси $0l$.

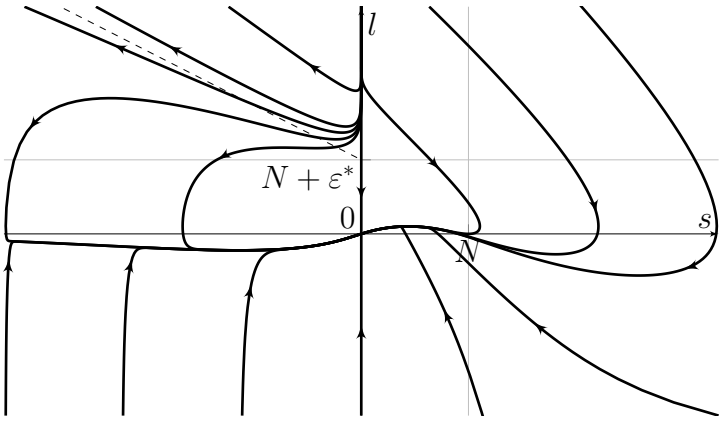

Рис. 2. Фазовый портрет модели вдали от особых точек
Fig. 2. Phase portrait of model far from equilibrium points

Так, если начальное состояние системы (6) находится на оси $0l$, когда нет сидеров, а есть только личеры, отсутствуют как раздача-скачивание, так и приток новых пользователей, так что на систему влияют только процессы оттока и траектория лежит строго на оси $0l$ и направлена в начало координат.

Для $s >> 0$, $l >> 0$ (первая четверть координатной плоскости, физически реализуемый случай) отток относительно мал. Траектории начинаются вблизи оси $0l$ (много личеров, а сидеров мало, но они есть). Вначале преобладает раздача-скачивание, и траектория отходит от оси $0l$ почти по прямой и подходит так к оси $0s$. Вблизи оси $0s$, когда большинство пользователей уже скачали файл, существенную роль начинает играть отток сидеров, и траектория искривляется, а движение по ней замедляется. Большинство траекторий развития реальных раздач имеют именно такую (с поправкой на случайные возмущения и суточные колебания) форму.

В случае $s << 0$ и $l << 0$ (третья четверть координатной плоскости) преобладает экспоненциальный отток, так что траектории вначале прижимаются к оси $0s$ (отток личеров пропорционален $e^{|s|+|l|}$ и происходит активнее), а затем двигаются вправо к началу координат.

В случае, когда $s + l << 0$ и $l << 0$, но $s > 0$ (четвёртая четверть) ситуация схожа с предыдущим случаем (траектории направлены к началу координат), но форма траекторий разнообразнее, так как на динамику влияют и другие процессы.

В случае, когда $s + l << 0$ при $l >> 0$ и $s << 0$ (часть второй четверти, прилежащая к оси $0s$) для l преобладает экспоненциальный отток, прижимающий траекторию к оси $0s$, так что траектории, отойдя от оси $0l$, описывают петли и возвращаются к началу координат.

Если же при $l >> 0$ и $s << 0$ получается $s + l >> 0$ (верхне-правая часть второй четверти, прилежащая к оси $0l$), отток по обеим переменным мал, так что траектории, удалившись от оси $0l$, делаются почти прямыми и расходятся веером. Ближайшие к оси $0l$ части этого веера со временем приближаются к оси $0l$ ещё сильнее и продолжаются в бесконечность; ближайшие к линии $l = -s$ рано или поздно описывают большие петли, прижимаются к оси $0s$ и попадают в начало координат. Можно показать, что сепаратриса, разделяющая бассейны бесконечности и начала координат, при увеличении l и $|s|$ асимптотически приближается к линии $l = -s + N + \varepsilon^*$, где $\varepsilon^* > 0$ — решение трансцендентного уравнения

$$\frac{\alpha\varepsilon}{\beta_2 e^{-\mu_2(N+\varepsilon)}} = 1. \tag{13}$$

На рис. 2 предельная линия $l = -s + N + \varepsilon^*$ показана пунктиром.

Поведение траекторий вблизи начала координат различается в зависимости от наличия и типа ненулевых особых точек, то есть значений величины q и параметра β_2. Это можно увидеть на рис. 3, a), b), c) и d), где представлен фазовый портрет (6) вблизи начала координат при различных соотношениях параметров (параметры рис. 2 соответствуют рис. 3, b).

Если ненулевых особых точек нет, любая траектория не из бассейна бесконечности рано или поздно придёт в начало координат $(0, 0)$ (рис. 3, b) и d).

При наличии седла (s_1^*, l^*) и притягивающего обобщённого узла (s_2^*, l^*) в первой четверти координатной плоскости (рис. 3, a), траектории, подходящие к началу координат справа, могут закончиться в (s_2^*, l^*). Бассейны притяжения начала координат $(0, 0)$ и ненулевого притягивающего обобщённого узла (s_2^*, l^*) разделяют сепаратрисы седла (s_1^*, l^*).

При наличии отталкивающего обобщённого узла (s_1^*, l^*) и седла (s_2^*, l^*) во второй четверти (рис. 3, c) любая траектория не из бассейна бесконечности рано или

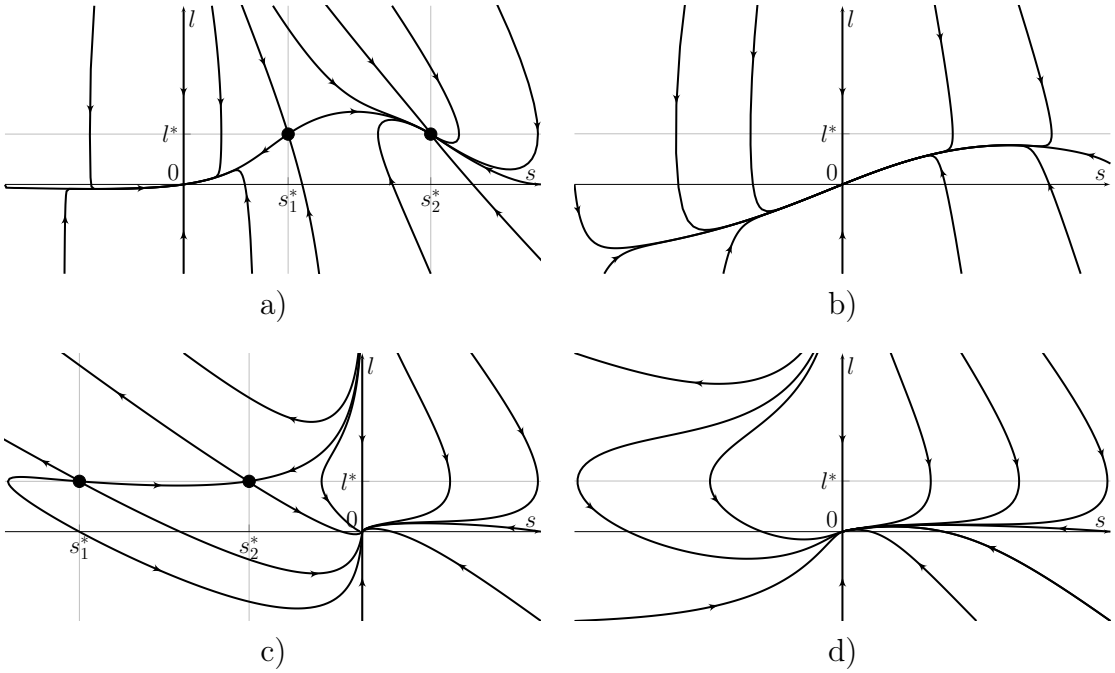

Рис. 3. Фазовый портрет модели вблизи особых точек: a) при $q > 0$, $\beta_2 < \beta_2^*$, b) при $q > 0$, $\beta_2 > \beta_2^*$, c) при $q < 0$, $\beta_2 < \beta_2^*$, d) при $q < 0$, $\beta_2 > \beta_2^*$
Fig. 3. Phase portrait of model far from equilibrium points: a) at $q > 0$, $\beta_2 < \beta_2^*$, b) at $q > 0$, $\beta_2 > \beta_2^*$, c) at $q < 0$, $\beta_2 < \beta_2^*$, d) at $q < 0$, $\beta_2 > \beta_2^*$

поздно придёт в начало координат $(0, 0)$, так же, как и при отсутствии ненулевых особых точек. Сепаратрисы седла отделяют относительно короткие траектории от описывающих большие петли вокруг обеих ненулевых особых точек.

Таким образом, стабилизация системы (6) в конечной ненулевой точке (s_2^*, l^*) возможна только при $q > 0$ и $\beta_2 < \beta_2^*$ (рис. 3, a). Во всех прочих случаях раздача из всех физически реализуемых начальных состояний достаточно быстро приходит в устойчивое нулевое состояние — вымирает (рис. 3, b), c), d).

5. Управление раздачей

Так как раздача всегда подвержена случайным внешним воздействиям, даже в случае, когда система (6) имеет две ненулевые особые точки (s_1^*, l^*), (s_2^*, l^*) в первом квадранте (то есть $q > 0$, $\beta_2 < \beta_2^*$), но расстояние между ними невелико, всегда есть возможность перескока из устойчивого состояния (s_2^*, l^*) в бассейн начала координат и дальнейшее исчезновение всех пользователей с раздачи.

Запасом устойчивости системы (6) будет расстояние h_{21} от устойчивого ненулевого состояния (s_2^*, l^*) до сепаратрисы, разделяющей бассейны ненулевого устойчивого состояния и начала координат [7, 8]. Эта сепаратриса соответствует притягивающему собственному направлению седла (s_1^*, l^*), а расстояние h_{21} тем больше, чем больше расстояние между ненулевыми особыми точками $s_2^* - s_1^*$.

Таким образом, для увеличения бассейна ненулевого притягивающего узла (s_2^*, l^*), и, соответственно, повышения устойчивости к внешним сбоям, необходимо как мож-

но больше увеличить расстояние $s_2^* - s_1^*$ между особыми точками системы, а для этого — увеличить как q, так и разницу $\beta_2^* - \beta_2$ (для краткости обозначим её b).

Можно показать, что повышения запаса устойчивости h_{21} раздачи (увеличения q и b) можно добиться, увеличивая μ_2, α и N либо снижая β_2. Уменьшение μ_2, α и N, как и рост β_2, приводят к снижению устойчивости.

Рассмотрим различные меры, принимаемые администрацией торрент-трекера, и их влияние на параметры системы (6) [9].

5.1. Рейтинг

На многих трекерах используется система рейтинга. Рейтинг пользователя равен отношению объёмов розданных и скачанных данных, причём пользователи с рейтингом больше единицы поощряются, с меньшим — штрафуются. Такая система двояко влияет на поведение сидеров — она уменьшает отток на популярных раздачах, сидирование которых позволяет быстро увеличить объём розданных данных и, соответственно, поднять рейтинг, но увеличивает отток сидеров с неактивных в данный момент раздач. Кроме того, так как скачивание файла уменьшает рейтинг, уменьшается и количество личеров, особенно на непопулярных раздачах.

Таким образом, учёт рейтинга приводит к увеличению N и α и снижению β_1 и β_2 для наиболее активных раздач (что сильно повышает их устойчивость), но при этом для малоактивных раздач эти параметры снижаются по сравнению с неконтролируемым файлообменом. Кроме того, снижаются параметры μ_1 и μ_2.

Всё это приводит к существенному повышению q и b для немногих популярных раздач и к снижению их для всех остальных.

Также недобросовестные пользователи могут искусственно увеличивать (накручивать) свой рейтинг с помощью различных манипуляций.

Таким образом, хотя система рейтинга призвана поощрять сидирование, она уменьшает запас устойчивости непопулярных раздач по сравнению с отсутствием регулирования.

При этом раздача может быть непопулярной не только из-за плохого качества раздаваемого файла, но и из-за того, что целевая аудитория данного конкретного файла мала (в частности, узкоспециализированная техническая и научная литература, оцифровки старых фильмов и радиопередач) — так называемые редкие раздачи. При этом, хотя на каждой отдельной раздаче пользователей очень мало, общее количество людей, интересующихся редкими файлами, как правило, огромно; соответственно, вымирание редких раздач может существенно снизить привлекательность трекера в целом.

5.2. Таймбонус и обратная связь

Администрация крупных современных торрент-трекеров поощряет как увеличение времени непрерывного сидирования в принципе (таймбонус), что соответствует уменьшению β_1, так и сидирование редких раздач, что эквивалентно увеличению μ_1. Всё это, как показано выше, приводит к уменьшению l^* и увеличению q и b.

Таймбонус, так же как и рейтинг, может быть накручен. В частности, если недобросовестный сидер сильно ограничивает канал отдачи, он может заработать тайм-

бонус практически без затрат исходящего трафика. Тем не менее, в этих условиях личер всё же может докачать файл, так что раздача остаётся активной.

Кроме того, в случае наличия на раздаче только личеров без сидеров администрация часто обращается с просьбой вернуться на раздачу к тем пользователям, которые ранее скачали файл и ушли с раздачи. С одной стороны, это смещает начальное состояние системы дальше от начала координат, с другой — знание о подобных мерах заставляет личеров дольше ждать и, соответственно, уменьшает β_2, что, в свою очередь, ещё больше увеличивает величину b и, следовательно, запас устойчивости системы.

Таким образом, система таймбонусов и обратной связи с пользователями увеличивает запас устойчивости раздач, в том числе непопулярных, по сравнению с отсутствием регулирования.

5.3. Поглощение раздач

Для популярных файлов зачастую возникает две или более раздачи, где раздаётся один и тот же материал в сопоставимом качестве и схожего объёма. В этом случае множество заинтересованных в этом файле пользователей дробится соответственно на несколько меньших групп объёмами N_1, N_2 и т. д., каждая из которых предпочитает одну из раздач.

Чем меньше отдельные N_i, тем менее устойчивы соответствующие раздачи, так что возможна ситуация, когда даже при большом суммарном количестве заинтересованных пользователей все раздачи файла придут в нулевое состояние.

Чтобы этого избежать, в подобных случаях администрация трекера закрывает все эквивалентные раздачи, кроме одной (лучшая раздача поглощает худшие). В этом случае количество заинтересованных пользователей оставшейся раздачи увеличивается до $N = \sum N_i$, вследствие чего её запас устойчивости возрастает.

Заключение

На основе анализа факторов, влияющих на количество пользователей раздачи в файлообменной сети, была построена динамическая модель (6), описывающая этап стабилизации раздачи после быстрого набора личеров.

Фазовые траектории системы (6) с учётом случайных возмущений и суточных колебаний хорошо соответствуют экспериментально полученным. Таким образом, выводы, полученные при анализе данной модели, могут быть использованы при администрировании торрент-трекера.

Исследование модели показало, что запас устойчивости каждой отдельной раздачи файлообменной сети можно повысить, используя систему таймбонусов. Также эффективна связь администрации с ранее скачивавшими файл пользователями и поглощение раздач-дубликатов. Система рейтинга повышает устойчивость некоторого количества наиболее популярных раздач, но отрицательно сказывается на всех остальных.

Список литературы / References

[1] Семенов Ю. А., *Телекоммуникационные технологии*, ИТЭФ-МФТИ, М., 2014; [Semenov Yu., *Telecommunication technologies*, ITEP-MIPT, M., 2014, (in Russian).]

[2] Андронов А. А. и др., *Качественная теория динамических систем второго порядка*, Наука, М., 1966; [Andronov A. A. et al., *Kachestvennaya teoriya dinamicheskih sistem vtorogo poryadka*, Nauka, M., 1966, (in Russian).]

[3] Баутин Н. Н., Леонтович Е. А., *Методы и приёмы качественного исследования динамических систем на плоскости*, Наука, М., 1990; [Bautin N. N., Leontovich E. A., *Metody i priyomy kachestvennogo issledovaniya dinamicheskih sistem na ploskosti*, Nauka, M., 1990, (in Russian).]

[4] Арнольд В. И., *«Жёсткие» и «мягкие» математические модели*, МЦНМО, М., 2008; In English: Arnold V.I., ""Hard" and "soft" mathematical models", *Butl. Soc. Catalana Mat.*, **13**:1 (1998), 7–26.

[5] Кононова А. И., "Исследование эволюции нелинейных динамических систем", *Сборник научных трудов победителей всероссийского конкурса научно-исследовательских работ студентов и аспирантов в области математических наук в рамках Всероссийского фестиваля науки*, 2011, 111–127; [Kononova A. I., "Issledovanie evolyucii nelinejnyh dinamicheskih sistem", *Sbornik nauchnyh trudov pobeditelej vserossijskogo konkursa nauchno-issledovatelskih rabot studentov i aspirantov v oblasti matematicheskih nauk v ramkah Vserossijskogo festivalya nauki*, 2011, 111–127, (in Russian).]

[6] Ильяшенко Ю. С., "Аттракторы динамических систем и философия общего положения", Матем. просв., сер. 3, **12**, Изд-во МЦНМО, М., 2008, 13–22; [Ilyashenko Yu.I., "Attractors of dynamical systems and philosophy of generic position", *Images de Mathematique*, **18**, 2006, 58–63, France.]

[7] Макаров И. М., Менский Б. М., *Линейные автоматические системы (элементы теории, методы расчета и справочный материал)*, Машиностроение, М., 1982; [Makarov I. M., Menskij B. M., *Linejnye avtomaticheskie sistemy (elementy teorii, metody rascheta i spravochnyj material)*, Mashinostroenie, M., 1982, (in Russian).]

[8] Gagarina L. G. et al., "Method for increasing reliability for transmission state of power equipment energy", *2015 IEEE global conference on signal and information processing (GLOBALSIP 2015)*, 2015, 433–437.

[9] Гагарина Л. Г. и др., "Моделирование процесса принятия управленческих решений", *Твердые бытовые отходы*, 2013, №1(79), 30–33; [Gagarina L. G. et al., "Modelirovanie processa prinyatiya upravlencheskih reshenij", *Tverdye bytovye othody*, 2013, №1(79), 30–33, (in Russian).]

Kononova A. I., "Dynamic Model of Single Torrent with File-Sharing P2P Network", *Modeling and Analysis of Information Systems*, **25**:4 (2018), 421–434.

DOI: 10.18255/1818-1015-2018-4-421-434

Abstract. In this work, a model of distribution of the file in P2P file-sharing network constructed on the basis of ordinary differential equations is considered. The phase variables which describe a condition of distribution of the file (as a first approximation is the number of users – seeder and leecher on distribution) are defined, the factors which influence the file distribution and the change of the number of users participating in exchange are analysed. On the basis of the analysis the system of the differential equations describing distribution evolution – dynamic model of evolution of distribution is written down. The life cycle of distribution in file-sharing network consisting of four stages – distribution creation, a fast gain leechers, stabilization and (for distributions of the files losing over time relevance) fading is considered. To each stage there corresponds the ratio of model parameters, and parameters change over time. The process of measuring the condition of real distributions is described. An example of the trajectory corresponding to the evolution of real distribution at a large torrent tracker is shown.

Further, the distribution stabilization stage which is characterized by constants as a first approximation parameters is considered. Equilibrium points of the dynamic model of distribution evolution are investigated, their possible quantity and type are described. All configurations of the general position, possible in the model of distribution evolution in a file-sharing P2P network are described. Phase portraits of each configuration are represented. The influence of various administrative measures on a stock of distribution stability is analysed. Ambiguity of the influence of a rating accounting system on the stability of distributions is shown. The positive influence of a system of timebonus, feedback and absorption of distributions is also shown.

Keywords: file-sharing network, ODEs, dynamical systems, stability

On the authors:
Alexandra I. Kononova, orcid.org/0000-0002-4178-3828, PhD,
National Research University of Electronic Technology,
1 Shokin sq., Moscow, Zelenograd, 124498, Russia, e-mail: illinc@bk.ru

Моделирование и анализ информационных систем. Т. 25, № 4 (2018), с. 435–458
Modeling and Analysis of Information Systems. Vol. 25, No 4 (2018), pp. 435–458

Тезаурусы

DOI: 10.18255/1818-1015-2018-4-435-458

УДК 004.912

Русскоязычные тезаурусы: автоматизированное построение и применение в задачах обработки текстов на естественном языке

Лагутина Н. С., Лагутина К. В., Адрианов А. С., Парамонов И. В.

получена 1 августа 2018

Аннотация. В работе выполнен обзор существующих электронных русскоязычных тезаурусов и методов их автоматического построения и применения. Авторы провели анализ основных характеристик тезаурусов, находящихся в открытом доступе, для научных исследований, оценили динамику их развития и эффективность в решении задач по обработке естественного языка. Были исследованы статистические и лингвистические методы построения тезаурусов, которые позволяют автоматизировать разработку и уменьшить затраты на труд экспертов-лингвистов. В частности, рассматривались алгоритмы выделения ключевых терминов из текстов и семантических тезаурусных связей всех типов, а также качество применения получившихся в результате их работы тезаурусов. Для наглядной иллюстрации особенностей различных методов построения тезаурусных связей был разработан комбинированный метод, генерирующий специализированный тезаурус полностью автоматически на основе корпуса текстов предметной области и нескольких существующих лингвистических ресурсов. С использованием предложенного метода были проведены эксперименты с русскоязычными корпусами текстов из двух предметных областей: статьи о мигрантах и твиты. Для анализа полученных тезаурусов использовалась комплексная оценка, разработанная авторами в предыдущем исследовании, которая позволяет определить различные аспекты тезауруса и качество методов его генерации. Проведённый анализ выявил основные достоинства и недостатки различных подходов к построению тезаурусов и выделению семантических связей различных типов, а также позволил определить потенциальные направления будущих исследований.

Ключевые слова: тезаурус, семантические отношения, автоматическое построение тезауруса, автоматическое выделение связей, выделение ключевых слов

Для цитирования: Лагутина Н. С., Лагутина К. В., Адрианов А. С., Парамонов И. В., "Русскоязычные тезаурусы: автоматизированное построение и применение в задачах обработки текстов на естественном языке", *Моделирование и анализ информационных систем*, **25**:4 (2018), 435–458.

Об авторах:

Лагутина Надежда Станиславовна, orcid.org/0000-0002-6137-8643, канд. физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: lagutinans@rambler.ru

Лагутина Ксения Владимировна, orcid.org/0000-0002-1742-3240, студентка,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: lagutinakv@mail.ru

Адрианов Алексей Сергеевич, orcid.org/0000-0002-3073-0982, студент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: alex.a4.25@yandex.ru

Парамонов Илья Вячеславович, orcid.org/0000-0003-3984-8423, канд. физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: Ilya.Paramonov@fruct.org

Введение

Тезаурус — это словарь терминов естественного языка, включающий в себя систему связей этих терминов [1]. Тезаурус может использоваться и как информационно-поисковый ресурс и как источник или эталон терминологии. Связи между словами являются материалом для построения лексико-семантических сетей для извлечения знаний, для определения семантической близости слов. Формализованность тезауруса позволяет легко автоматизировать его применение. Многие исследователи подчеркивают важность построения электронных тезаурусов и перспективу их использования в автоматических системах обработки текстов [2, 3].

Решая задачи в области автоматической обработки текстов на естественном языке, авторы обнаружили тот факт, что тезаурус является удобной моделью предметной области. Авторы успешно использовали автоматически сгенерированный тезаурус для построения альтернативной системы навигации для электронного туристического ресурса Open Karelia [4], а также для анализа тональности журнальных статей [5]. Следует отметить, что эти исследования проводились с использованием англоязычных текстов и применением соответствующих методов, алгоритмов и лингвистических ресурсов.

Попытка напрямую использовать разработанные подходы к анализу текстов на русском языке не дала столь же успешного результата. Самый поверхностный анализ ситуации показал низкое качество получаемых тезаурусов, в частности малое количество выделяемых связей между словами. Кроме того, применение в использованных алгоритмах внешнего электронного тезауруса РуТез как альтернативы англоязычного тезауруса WordNet существенно не повлияло на качество решения. Это побудило авторов провести анализ методов построения русскоязычных тезаурусов, а также более внимательно рассмотреть существующие электронные русскоязычные тезаурусы.

В 2015 году в работах [6, 7] был проведен краткий анализ русскоязычных тезаурусов. Авторы соответствующих обзоров отмечают недостаточный объём этих ресурсов, в частности по сравнению с WordNet, трудность интеграции с существующими алгоритмами и системами автоматической обработки текстов, сложность или невозможность использовать их, в том числе в коммерческих целях. Однако цифровые ресурсы могут достаточно быстро меняться, поэтому было бы интересно определить динамику и тенденции развития русскоязычных тезаурусов.

Существующие тезаурусы не могут полностью охватить весь спектр вопросов автоматической обработки текстов, особенно в предметных областях, поэтому задача построения новых тезаурусов и как самостоятельных ресурсов, и как вспомогательных инструментов является актуальной. К сожалению, авторам данной работы не удалось найти обзоры методов автоматизированного построения русскоязычных лексических ресурсов. Таким образом, одной из целей данной статьи стало описание

методов, применяемых при автоматизации построения тезаурусов, и определение направлений развития этих методов.

Представленная работа состоит из трех частей. В первой части проанализированы характеристики электронных тезаурусов русского языка, определена динамика развития этих ресурсов и оценены возможности их применения к задачам автоматической обработки текстов. Во второй части авторы рассматривают методы выделения терминов и связей между ними, применяемые современными исследователями при построении лексических ресурсов, чтобы определить основные тенденции и возможные направления развития в этой сфере, в первую очередь для построения тезаурусов. Третья часть описывает собственный опыт построения русскоязычных тезаурусов предметных областей. В заключении подведены итоги работы.

1. Обзор существующих электронных русскоязычных тезаурусов

Для английского языка существует бесплатный лингвистический ресурс WordNet (https://wordnet.princeton.edu), который можно назвать эталонным тезаурусом. Этот проект разрабатывается с начала 80-х годов XX века и активно используется в исследованиях. Для русского языка существует несколько проектов подобного рода.

1.1. Свойства и характеристики тезаурусов

Самым большим по объему является тезаурус RuTez. Этот проект разрабатывается Лабораторией информационных исследований во главе с Н. В. Лукашевич [8]. В его основу положены принципы построения WordNet, но модель описания сущностей отличается. Единицей тезауруса является понятие, снабженное набором терминов, значения которых соответствуют данному понятию. В качестве терминов могут выступать слова и словосочетания, количество которых может быть достаточно велико, например 20 и более. Слова и словосочетания, относящиеся к одному понятию, называются онтологическими синонимами.

Понятия связываются системой отношений. В RuTezе используются четыре типа связей: два вида иерархических отношений — «выше—ниже» и «часть—целое»; два вида ассоциативных отношений — симметричная ассоциация, связывающая понятия, очень близкие по смыслу, но не соединенные в одно понятие; и несимметричная ассоциация, связывающая два понятия, которые не могут существовать друг без друга, но не находятся в других отношениях.

На текущий момент тезаурус RuTez содержит 55 тысяч понятий, 158 тысяч слов и выражений, 210 тысяч отношений между этими понятиями. Как видно из таблицы 1, этот объем информации сопоставим с объемом WordNet по всем параметрам, кроме количества сущностей. Однако модель понятия RuTeza и модель синсета WordNet отличаются по своему смыслу, поэтому количественное сравнение в этом случае не вполне корректно.

Для некоммерческого использования по запросу можно получить тезаурус в формате XML. Кроме того, существует бесплатная версия RuTez-lite, содержащая 115

Таблица 1: Характеристики русскоязычных тезаурусов в сравнении с англоязычным тезаурусом Wordnet

Table 1: Characteristics of Russian-language thesauri in comparison with the English-speaking thesaurus Wordnet

Характеристика	РуТез	YARN	Wordnet.ru	RussNet	Wordnet
Основная единица	понятие	синсет	синсет	синсет	синсет
Количество единиц	55 тыс.	70 тыс.	20 тыс.	5,5 тыс.	117,7 тыс.
Количество слов, словосочетаний	158 тыс.	143,5 тыс.	100 тыс.	15 тыс.	155 тыс.
Виды отношений	иерархич., ассоц.	иерархич., антонимия	иерархич.	иерархич., антонимия, ассоц.	иерархич.
Число пар отношений	210 тыс.	134 тыс.	—	—	207 тыс.
Формат для использования	XML	XML	текстовые файлы	—	API
Последнее обновление	постоянно обновляется	постоянно обновляется	28.08.2008	14.06.2005	постоянно обновляется
Головная организация	Лаборатория информац. исследований	УрФУ, СПбГУ	авторский проект	СПбГУ	Принстонский университет

тысяч слов и выражений, которую можно изменять и копировать также в некоммерческих целях [9].

Разрабатывается и обновляется РуТез в первую очередь экспертами Лаборатории информационных исследований. Этот подход гарантирует качество тезауруса, но требует значительного времени для пополнения и изменения данных. Авторы обращают внимание на отзывы о своем проекте и учитывают существенные замечания. В целях интеграции с существующими мировыми онтологиями и тезаурусами на других языках в 2017 году РуТез был автоматически преобразован в формат WordNet. В результате создан тезаурус RuWordNet, содержащий 111,5 тысяч слов и выражений русского языка [10].

Еще один активный проект по созданию тезауруса русского языка — Yet Another RussNet (YARN) [11]. Разработчики используют модель полностью соответствующую WordNet, в основе которой лежат синсеты — группы синонимов и квазисинонимов, объединённые общим лексическим значением. Синсеты связываются между собой иерархическими отношениями и отношениям антонимии, устанавливаемым между противоположными по значению терминами. Дополнительно есть отношения между словами, а также межъязыковые связи между синсетами YARN и WordNet. В качестве начального наполнения была использована информация из Викисловаря.

Отличительной чертой данного проекта является организация дополнения тезауруса на основе краудсорсингового подхода: любой желающий после регистрации

на сайте YARN может участвовать в добавлении и редактировании данных. Авторы проекта декларируют контроль качества с помощью выделенного среди пользователей ядра редакторов, которые могут утверждать или отменять изменения, а также запрещать дальнейшее редактирование отдельных элементов тезауруса. Таким образом качество проекта напрямую зависит от квалификации выбранных редакторов.

В настоящий момент YARN содержит 143 508 слов, 69 799 синсетов, 104 906 неотредактированных пар синонимов, 29 764 неотредактированных родо-видовых отношений. Это меньше, чем в РуТез и WordNet, но если сравнить объем YARN в 2015 [7] и 2018 годах, то очевидна динамика роста. Этот факт, по-видимому, обусловлен применением автоматизации при добавлении данных из ресурсов, аналогичных Викисловарю, и краудсорсингового подхода.

Неоспоримым преимуществом проекта является то, что YARN распространяется по лицензии, разрешающей использовать материалы в исследовательских и коммерческих приложениях, копировать и изменять данные с условием ссылки на источник. Содержимое тезауруса предоставляется в XML-формате.

В открытом доступе находится еще один тезаурус, являющийся попыткой перевода WordNet — Wordnet.ru или Русский Wordnet [12]. Система построения ресурса полностью автоматическая и не предусматривает экспертной оценки. Авторы использовали эвристический алгоритм проверки соответствия синсетов английского языка синсетам русского, полученного в результате прямого перевода. Таким образом им удалось сгенерировать около 25 тысяч синсетов, 100 тыс. слов и словосочетаний.

Авторы разработали программу, визуализирующую Русский Wordnet, а текстовые файлы с данными к ней можно скачать в виде отдельного архива. К сожалению, дата модификации файлов в архиве последней версии — 28 августа 2008 года. Малый объем и отсутствие развития данного проекта показывают, во-первых, сложность построения полностью автоматического инструмента создания тезауруса, во-вторых, невозможность построения полноценных лексических ресурсов языка только на основе перевода без учета национальных особенностей.

Можно упомянуть ещё один электронный тезаурус, разработанный на кафедре математической лингвистики Санкт-Петербургского государственного университета — RussNet [13]. Модель ресурса полностью соответствует WordNet. Авторы создавали тезаурус, максимально совместимый с европейским многоязычным тезаурусом EuroWordNet, в рамках которого предложена система межъязыковых отсылок (Inter-Lingual-Index, ILI), дающих возможность переходить от терминов одного языка к сходным, но не обязательно тождественным словам другого. Следование этой системе позволяет использовать тезаурус для многоязычного поиска. Следует также отметить, что этот ресурс содержит самое большое количество разных типов связей, что обусловлено работой высококвалифицированных экспертов-лингвистов. К сожалению, закрытость данного проекта и отсутствие изменений с 2005 года сводит на нет все его преимущества. Однако в настоящий момент осуществляется интеграция RussNet и YARN [14]. Это позволит не только увеличить объем тезауруса, но и повысить его качество, так как RussNet создавался группой высококвалифицированных экспертов.

Среди открытых тезаурусов предметных областей авторам удалось обнаружить

только русскоязычную версию многоязычного тезауруса сельскохозяйственной терминологии AGROVOC [15]. Перевод AGROVOC на русский язык был выполнен специалистами Центральной научной сельскохозяйственной библиотеки в 2011 году.

Таким образом, в настоящий момент лингвистическими ресурсами, аналогичными англоязычному WordNet, для русского языка можно считать два тезауруса — РуТез и YARN. Все три ресурса сопоставимы по объему данных, как по количеству слов и словосочетаний, так и по количеству связей между ними. Преимуществом РуТеза, по мнению авторов, является качество, обеспеченное коллективом экспертов, а также разумный подход к развитию и преобразованию. Преимуществом YARN является его совместимость с международными лексическими ресурсами. Справедливости ради следует отметить, что разработчики РуТеза создали такую же совместимую версию — RuWordNet, но она пока меньше источника по объему. Также в качестве положительной стороны YARN нужно указать возможность его коммерческого использования.

1.2. Применение существующих тезаурусов к решению задач

Анализ электронных русскоязычных тезаурусов был бы неполным без обзора работ, использующих эти ресурсы для решения конкретных задач.

Существующие тезаурусы используются как источник терминологии. Авторы работы [16] используют РуТез как основу для построения общественно-политического тезауруса национального языка (татарского). В этом примере РуТез выступает в первую очередь как обычный филологический словарь тезаурусного типа. Авторы подчеркивают возможность использования такого рода лексических ресурсов в различных приложениях для автоматической обработки новостных документов, правовых актов или сообщений в социальных сетях. Поэтому полученный тезаурус преобразован и опубликован в облаке лингвистических открытых связанных данных — Linguistic Linked Open Data (LLOD) [17].

Следует отметить, что с целью передачи специфики татарской лексико-семантической системы авторы активно используют большое количество внешних источников. Такими источниками являются, во-первых, существующие двуязычные татарско-русские словари, в том числе специализированные общественно-политические, во-вторых, большое количество медиатекстов и текстов официальных документов. Текстовые документы предметной области необходимы, чтобы найти подходящие татарские термины для замены устаревших слов или добавления отсутствующих понятий.

Как источник терминов РуТез используется в работе [18]. Авторы решают задачу автоматической рубрикации текстов. Каждому понятию тезауруса ставится в соответствие рубрика текста. В исследуемых текстах ищутся слова, которые есть в РуТезе. На основе полученных данных строится модель тематического представления каждого текста и определяется рубрика. Эксперименты показали высокое качество результатов. Однако наличие узкопредметной рубрики потребовало дополнения тезауруса новыми терминами.

Структура тезауруса может быть использована как образец для построения удобного в использовании лингвистического ресурса. В полном соответствии со

структурой РуТез построен тезаурус по безопасности [19]. Этот тезаурус используется в специализированной информационно-аналитической системе, в том числе для автоматической текстовой классификации документов. Наполнение данными осуществляется из других источников: нормативно-справочная литература, специализированные текстовые коллекции, новости из средств массовой информации по тематике безопасности.

Связи между словами в тезаурусах являются ключевой информацией в методах расчета семантической близости терминов. В работе [20] сделана попытка автоматизировать оценку ответов учащихся на вопросы открытого теста. Анализ ответов на русском языке осуществлялся с использованием РуТеза и Википедии. С их помощью рассчитывалась семантическая близость слов ответа учащегося и эталонного ответа, расположение слов в тексте не учитывалось. По результатам экспериментов авторы отмечают, что качество работы системы сильно понижалось в случае появления синонимов, например, когда эталонный ответ содержал дублирующие варианты возможного ответа. Однако авторы используют только отношения «выше—ниже» и никак не учитывают синонимические отношения между словами.

Все виды связей из РуТеза используются автором работы [21] при расчете информации о схожести терминов. Это один из факторов схожести в разработанном методе построения групп семантически близких слов и выражений, описывающих тематические узлы кластера новостных сообщений. Метод был применен к задаче автоматического реферирования новостей.

Тезаурус может быть эталоном при оценке качества работы методов автоматической обработки текстов. Автор работы [22] предложил метод автоматического группирования семантически близких слов. Для оценки результатов работы использовались материалы РуТез и YARN. Качество результата было не очень высоким, особенно в случае сравнения с тезаурусом РуТез. Это объясняется тем, что метод автора базируется на графе синонимов, а синсет из YARN гораздо ближе к определению синонимов, чем единица тезауруса РуТез — понятие.

Из обзора работ видно, что тезаурусы используются в широком спектре исследований — РуТез чаще, YARN реже, — однако практически всегда совместно с другими лингвистическими ресурсами. В случаях анализа предметных областей эти тезаурусы можно применить очень ограниченно. В частности, нехватку терминов предметной области в использованных электронных ресурсах отмечают авторы работы [20].

Универсальные лексические ресурсы РуТез и YARN содержат полезную информацию, удобны для использования, кроме того, они динамично развиваются. Однако для решения огромного количества задач автоматической обработки текста нужны тезаурусы предметных областей. В силу сложности и трудоёмкости построения этих ресурсов их достаточно мало. Тем не менее, уровень развития автоматизированных методов построения тезаурусов делает эту задачу выполнимой, поэтому авторы хотели бы обратить внимание на актуальность построения открытых русскоязычных тезаурусов самых разных предметных областей.

2. Особенности автоматизированного построения русскоязычных тезаурусов

Задачу построения тезауруса можно разделить на две большие подзадачи: выделение ключевых слов и выделение связей между словами.

2.1. Выделение ключевых слов

Выделение ключевых слов — самая изученная часть процесса построения тезауруса, а также многих других задач автоматической обработки текста, в том числе на русском языке. Основную роль здесь играют статистические методы. Для английского языка способы выделения ключевых слов, включая предварительную обработку текста, хорошо исследованы и верифицированы [24]. Для построения русскоязычных тезаурусов применяются аналогичные методы, но количество исследований, тщательно оценивающих качество, значительно меньше.

При формировании РуТеза применялся автоматизированный способ получения терминологии [25]. Для каждой предметной области из рассматриваемого списка (математика, физика, химия, биология, геология) были сформированы коллекции документов (от 3 000 до 8 000 документов, объемом от 50 до 90 Мб каждый). Источниками коллекций являлись документы, доступные в интернете: материалы школьных уроков, рефераты, университетские лекции, материалы специализированных сайтов. Для выявления терминов были применены два алгоритма выделения терминоподобных слов и словосочетаний [26].

Первый алгоритм выделял существительные, прилагательные, согласованные пары и тройки прилагательных и существительных, а также заранее заданные конструкции (существительное — существительное в родительном падеже и т. п.). Второй алгоритм выделял часто повторяющиеся слова и группы в несколько слов. Полученные слова и словосочетания упорядочивались по убыванию частотности и убыванию количества содержащих их документов. Для проверки и добавления ключевых слов было проведено сопоставление с терминами Общественно-политического тезауруса, разработанного авторами ранее.

Следует отметить, что Общественно-политический тезаурус и в дальнейшем РуТез разрабатываются коллективом исследователей с конца 90-х годов. Основным способом оценки качества является экспертная оценка. Это гарантирует высокое качество, но лишний раз подтверждает сложность и трудоемкость работы. Решение задач автоматической обработки текстов требует развития методов быстрого построения узкопредметных тезаурусов и оценки их качества.

Большинство работ, описывающих автоматическое построение собственных тезаурусов или близких к ним структур, таких как лексико-семантические сети, онтологии предметных областей, при выделении терминов используют частоту встречаемости слов в используемом корпусе текстов и меру семантической близости терминов [27]. Частота встречаемости слов определяется количеством вхождений слова в корпусе текстов. Мера семантической близости предназначается для количественной оценки семантической схожести терминов. Эта характеристика может вычисляться разными способами и часто опирается на построение для каждого сло-

ва контекстного многомерного вектора и дальнейшего анализа пространства таких векторов.

Автор работы [28] решал задачу построения тезауруса, а точнее лексико-семантического поля вокруг заданного термина. В качестве исходного материала был выбран термин «двигатель» и корпус ruTenTen 2011, содержащий 14,55 миллиардов слов. В качестве статистических характеристик слов использовались степень семантической близости данного слова к ключевому и частоты данного слова в корпусе. Слова упорядочивались по степени семантической близости. Также на основе набора шаблонов и подсчёта частоты встречаемости были выделены словосочетания из двух и более слов. Однако окончательное формирование тезауруса осуществлялось экспертом.

Только на основе частоты встречаемости выбираются термины в работе [29]. В данной статье исследуется применение метода автоматической разработки тезаурусов предметных областей для построения тематических онтологий, применяемых в учебном процессе. Тезаурус строился на основе корпуса текстов предметной области. К сожалению, из работы неясно, каким образом определялись связи между терминами. Все этапы формирования тезауруса оценивал эксперт.

В статье [30] предлагается способ расчета весовых коэффициентов для определения степени значимости термина для заданной предметной области. Авторы рассматривали корпус научных статей, в рамках которого определили три числа. Первое — это ранг частоты, который позволяет уравнивать значимости самых встречаемых терминов любых текстов и одновременно распределять значимость терминов внутри одного текста. Второе — соответствие тематике текста. Тематическую группу научного текста формировали, выделяя термины из заголовка и подзаголовков, если при этом частота встречаемости терминов будет высокой в самом тексте; в этом случае вклад в значение весового коэффициента термина считается равным 1, иначе — 0. Третий коэффициент рассчитывался на основе принадлежности термина к содержательно смысловым (по терминологии авторов) блокам научной статьи. Авторы выделили индикаторы и маркеры четырех основных блоков: проблема, опыт, решение, итог. Если термин используется в предложении, содержащем формальный признак того или иного блока, то его вес корректируется на соответствующую величину. При этом если термин встретился в более чем одном блоке, его вес изменяется на сумму соответствующих величин. В качестве итоговой оценки из трех чисел рассчитывался интегральный весовой коэффициент термина. Таким образом, для анализа научных работ была предложена система метрик, тесно связанная со структурой текстовых документов, характерных для этой области.

Описанный способ был использован при разработке подхода автоматического определения тематики научного документа [31]. Эта работа показывает, что для разных предметных областей и сами методы, и отдельные детали методов могут значительно отличаться или изменяться. Хотя анализ структуры документа чаще применяется в алгоритмах извлечения знаний из текстов, тем не менее, такой подход оправдан при построении тезаурусов предметных областей, если используемый корпус текстов или его часть имеет заранее понятный формат.

В работах [32, 33] описана разработка русского тонального лексикона РуСентиЛекс. Большая часть этого лингвистического ресурса была построена вручную, но часть его терминов была выделена из текстов из социальной сети Twitter при по-

мощи алгоритма, комбинирующего бинарные классификаторы Logistic Regression, LogitBoost и Random Forest. Точность выбора терминов оценивалась на основе англоязычного корпуса отзывов, термины которого уже были размечены по тональности, и достигала 78,6 %. Отметим, что оценка качества работы алгоритма для русскоязычных текстов проводилась экспертами вручную, без какой-либо автоматизации. Авторы также доказали практическую полезность лексикона для тональной классификации текстов, организовав соревнование SentiRuEval-2016, участники которого разработали алгоритмы классификации с применением РуСентиЛекс, показавшие качество классификации в диапазоне 55–68 % для F-меры.

Отдельно хотелось бы остановиться на оценке качества выделения ключевых слов. Для этого существуют стандартные числовые характеристики: точность, полнота и F-мера [24]. Однако в статьях, посвященных работе с русскоязычными текстами, расчет этих характеристик осуществляется редко. В основном результат верифицируется экспертом, чаще всего без упоминания каких-либо числовых параметров, кроме количественных. Скорее всего, это связано с отсутствием размеченных тематических корпусов текстов на русском языке. Этот факт отмечается и в исследовании [34]. Авторы конкретных работ не выкладывают в общий доступ разработанные и используемые ими корпуса текстов. Конечно, во многих случаях это обусловлено объективными причинами, но наличие даже небольшого количества размеченных русскоязычных корпусов текстов существенно бы способствовало развитию автоматизации обработки данных в этой области.

В работе [35] качество отбора словосочетаний как терминов предметной области оценивалось с использованием существующей онтологии по естественным наукам и технологиям (OEHT). В качестве меры была выбрана средняя точность выделенных терминов, значения которой оказались в диапазоне от 59 % до 75 % в зависимости от характеристик выделяемых словосочетаний. К сожалению, этот метод можно применить только в тех немногих предметных областях, для которых существуют доступные онтологии. Кроме того, в этом случае встает вопрос о формировании корпуса текстов для исследования.

Следует отметить, что для анализа русскоязычных текстов используется достаточно узкий набор методов выделения ключевых слов. Согласно выводам исследователей, качество работы применяемых подходов и ряда стандартных инструментов, не зависящих или почти не зависящих от языка, является удовлетворительным в рамках решаемых задач. Однако малое количество и не слишком высокое качество существующих электронных русскоязычных тезаурусов, особенно по сравнению с английским языком, по мнению авторов, напрямую связано с недостаточным исследованием всего существующего спектра методов выделения ключевых слов.

2.2. Выделение связей

Существует два типа алгоритмов выделения семантических связей: статистические и лингвистические. Первые вычисляют статистические характеристики терминов в зависимости от их встречаемости в текстах, затем применяют математические методы, чтобы определить, насколько близки термины. В большинстве случаев статистические методы находят ассоциативные связи. Лингвистические методы используют существующие ресурсы: словари, тезаурусы, онтологии и т. д., откуда выделя-

ются существующие связи, или применяют лингвистические правила или шаблоны. Лингвистические методы способны находить любые виды семантических связей. Обзор этих методов можно найти в работе авторов [36].

Методы автоматического выделения семантических связей активно применяются к англоязычным текстам. Анализ статей, посвященных разработке русскоязычных лингвистических ресурсов, сразу показывает слабую степень автоматизации методов решения рассматриваемой задачи.

Разработчики РуТеза уделяют большое внимание описанию разных видов связей, которые могут присутствовать в тезаурусе [37, 38]. Однако при построении тезауруса основная часть отношений между понятиями РуТеза была получена из Общественно-политического тезауруса, а дополнялись связи экспертами.

Выделение связей предоставляют экспертам и авторы некоторых других работ [28, 29, 39]. В статье [28] явно выделена актуальность автоматизации построения узкопредметных тезаурусов и то, что важной их частью являются связи между словами.

Многие авторы используют связи между терминами из доступных лингвистических ресурсов, в основном из Википедии и РуТеза. Для решения задачи автоматического определения тематики документа [31] построена собственная онтология, в которой используются синонимические, гипонимо-гиперонимические связи из Википедии. Экспертный анализ результатов показал повышение качества работы системы при использовании онтологии, однако авторы указали на сложность построения таких лексических ресурсов. Использование РуТеза обсуждалось в разделе 1.2.

Получение связей между словами статистическими методами на основе частоты совместной встречаемости применён в работе [40]. Исследователи не строили в явном виде тезаурус, но показали, что задача классификации русскоязычных текстов решается лучше, если использовать семантические связи между терминами предметной области.

Открытый дистрибутивный тезаурус русского языка [41] строился автоматически методом Skip-Gram, реализованным в инструменте word2vec (https://code.google.com/archive/p/word2vec/), который находит семантические связи между терминами при помощи статистических алгоритмов. Авторы также автоматизировали и оценку качества тезауруса, сравнивая выделенные для него связи с существующими ресурсами с размеченными отношениями между словами: корпус BLESS, корпус когнитивных ассоциаций и другие. Средняя точность выбора связей достигала 97 %.

В работе [42] предложен полностью статистический метод автоматического построения ассоциативных связей для тезаурусов для нескольких языков, в том числе и для русского. Метод основан на подсчёте совместной встречаемости слов, сингулярном разложении матрицы «термины-на-документы» и нескольких мерах семантической близости. Для тезауруса английского языка экспертная оценка качества продемонстрировала очень высокие значения метрик: точность и полнота достигали 86–97 %. Следует отметить, что в работе недостаточно подробно описана оценка качества полученного русскоязычного тезауруса, в частности, не приведены значения метрик, которые бы описывали качество тезауруса в целом.

В работах [43, 44] описано построение ассоциативного портрета предметной области, т.е. совокупности наиболее характерных предметных и лингвистических зна-

ний, свойственных такой области. Авторы использовали методы для выделения связей на основе векторного алгоритма определения семантической близости слов. Связи не делились на отдельные виды, а рассматривались как единый набор ассоциаций. Получаемые структуры успешно применялись для решения задач [45, 46].

Хотелось бы обратить внимание на то, что в большинстве работ виды связей между словами не учитываются. Трудно сказать, связано ли это с достаточным качеством решения задач обработки текстов без учета видов связей или с трудностью определения разных типов иерархических и ассоциативных связей. Авторы данной работы показали существование отличия в степени влияния синонимических, иерархических и ассоциативных связей при использовании тезауруса для анализа тональности текстов [47]. Однако исследование влияния разных типов связей между терминами тезаурусов на качество результатов в задачах компьютерной лингвистике — мало изученная проблема, решение которой может привести к улучшению и развитию методов автоматического анализа текстов на естественных языках.

Одним из методов выделения разных типов связей является применение лексико-синтаксических шаблонов. Лексико-синтаксические шаблоны представляют собой характерные выражения (словосочетания и обороты), конструкции из определенных элементов языка. Примеры наиболее распространённых в русском языке шаблонов, в которых встречаются гипонимо-гиперонимические отношения, хорошо описаны в работе [48].

Группа разработчиков сформулировала язык для записи лексико-синтаксических шаблонов — LSPL [49]. Созданные при помощи предложенного языка шаблоны типичных фраз научной речи применялись авторами для автоматического анализа научно-технических документов на русском языке. Следует отметить, что для обработки текста были использованы традиционные словари (терминологический и морфологический), а также словарь общенаучных слов и выражений. Однако в данном случае шаблоны использованы не как средство получения связей между терминами предметной области, а как средство извлечения знаний из текста на естественном язке.

В другой работе [50] обсуждается проблема автоматического построения онтологий на основе использования лексико-синтаксических шаблонов. Автор предложил собственный язык лексико-синтаксических шаблонов и разработал на его основе программный комплекс, который позволяет хранить шаблоны и корпус текстов на русском языке в базе данных, редактировать и проводить валидацию шаблонов на корпусе русскоязычных текстов, проводить семантический анализ текстов корпуса. Оценку своего метода построения онтологий автор не выполнял, но предложил способ оценки качества работы на основе решения задачи информационного поиска.

Большинство лексико-синтаксических шаблонов разрабатываются экспертами. Было бы интересно провести исследование, описывающее и анализирующее широкий спектр таких шаблонов, применимых к построению русскоязычных тезаурусов предметных областей.

Таким образом, методы автоматического выделения связей между словами русского языка применяются исследователями в основном при построении лексических конструкций в рамках непосредственного решения задач обработки текста. Важно, что практически во всех работах отмечается существенное повышение качества решения после использования найденных связей. Это подчеркивает необходимость

разработки и исследования методов автоматизации для построения качественных тезаурусов, особенно в рамках задачи выделения связей между словами.

Интересно, что оценка лингвистических ресурсов, построенных авторами работ рассмотренных в этом разделе, как правило, проводится на основе качества решения с их помощью задач автоматической обработки текста. Это ещё раз подчеркивает нехватку размеченных экспертами эталонных корпусов текстов. Поскольку создание таких корпусов — задача очень трудоемкая, она также нуждается в методах и средствах автоматизации.

3. Эксперименты по построению русскоязычного тезауруса

Обзор существующих работ по построению тезауруса показывает, что создание специализированных лингвистических ресурсов автоматизировано недостаточно и даже после автоматизации требует большой работы эксперта-филолога во многих аспектах в целях повышения качества тезауруса для дальнейшего применения. Чтобы наглядно проиллюстрировать, каким качеством обладают тезаурусы, построенные полностью автоматически, и какие методы из области обработки естественного языка наиболее полезны для генерации тезауруса, авторы предложили комбинированный метод и провели эксперименты с созданием двух тезаурусов предметной области.

Проанализировав методы построения тезаурусов, авторы выделили наиболее подходящие, по их мнению, подходы к выделению ключевых слов и связей между ними для построения тезауруса как лексико-семантической модели предметной области. Этот раздел описывает результаты экспериментов по реализации разработанного алгоритма. Особое внимание было уделено способам выделения различных типов связей как наименее исследованной задаче.

Алгоритм построения русскоязычного тезауруса основан на алгоритме построения тезауруса, описанном в предыдущем исследовании авторов [36], и состоит из следующих этапов:

1. Выделение ключевых слов из текстов как терминов тезауруса алгоритмом TextRank [51].
2. Выделение ассоциативных связей статистическими алгоритмами.
3. Выделение синонимических связей из существующих лингвистических ресурсов и методом расстояния Левенштейна [53].
4. Выделение иерархических связей линвистическими методами.
5. Фильтрация терминов без связей.

На каждом этапе выделения связей предыдущие связи могут быть перезаписаны, например, если алгоритм определил пару терминов как синонимы, то если между ними была ранее установлена ассоциативная связь, она будет удалена, а синонимическая — записана вместо неё.

После построения основных структурных элементов тезауруса из него удаляются термины, для которых не было найдено ни одной связи, так как они бесполезны для дальнейшего применения тезауруса в обработке текстов.

3.1. Выделение терминов

В первую очередь для тезауруса выбираются термины, которые описывают основные темы предметной области, т.е. ключевые слова. Алгоритмы выделения ключевых слов были исследованы в предыдущей работе авторов [4]. По результатам исследования обучаемые алгоритмы работают более эффективно, чем алгоритмы без учителя, но составление материала для обучения требует огромных затрат со стороны эксперта-лингвиста. Алгоритмы типа «обучение без учителя» — PageRank и Topical TextRank — демонстрируют достаточно близкие характеристики качества работы, причём Topical PageRank является вариацией TextRank, которая более точно выбирает ключевые слова для конкретных текстов. Тем не менее, итоговый набор ключевых слов для всех текстов, который и берётся для основы тезауруса, оказывается одинаков для обоих алгоритмов.

Исходя из особенностей методов выделения терминов и требования максимальной автоматизации построения тезауруса, для создания русскоязычного тезауруса был выбран алгоритм TextRank, обладающий хорошими характеристиками качества и не требующий дополнительного обучения. Эксперименты с построением тезауруса проводились на двух базах текстов: статьи о российских мигрантах, выбранные экспертами-филологами ЯрГУ, и корпус русскоязычных твитов (http://linis-crowd.org/).

Корпус статей о мигрантах содержит 103 текста, из них 20 положительных и 83 отрицательных. В среднем в каждом тексте содержится 682 слова или 4 785 символов.

Корпус твитов содержит 4 320 текстов, из них 2 160 положительных и 2 160 отрицательных. В среднем в каждом тексте содержится 157 слов или 1 044 символа.

3.2. Выделение ассоциативных связей

Для выделения ассоциаций были выбраны два алгоритма, предназначенных для поиска семантически близких терминов: латентно-семантический анализ (ЛСА) [52] и word2vec (https://code.google.com/archive/p/word2vec/).

В методе латентно-семантического анализа строится матрица «термины-на-документы», где строкам соответствуют термины тезауруса, выбранные на предыдущем этапе, а столбцам — тексты предметной области, на которых генерируется тезаурус. Элементами матрицы служат частоты встречаемости терминов в конкретных текстах. После того как матрица построена, она раскладывается по сингулярным значениям, что позволяет найти матрицу меньшей размерности, которая является достаточно точным приближением исходной матрицы. Строки итоговой матрицы интерпретируются как векторы, характеризующие взаимосвязь соответствующего термина с текстами. Эти векторы сравниваются между собой по одной из стандартных мер близости векторов, например, по косинусной мере. В конце для пар терминов, у которых векторы обладают лучшими характеристиками близости, выделяются ассоциативные связи.

Инструмент word2vec строит векторные представления слов и по косинусной мере ищет семантически близкие термины так же, как и предыдущий. Этот алгоритм требует предварительного обучения, но для него существуют уже обученные модели, в том числе и для русского языка, которыми можно свободно пользоваться. Век-

торы для слов строятся при помощи стандартных алгоритмов CBOW (Continuous Bag of Words) и Skip-Gram.

Оба алгоритма выделяют достаточно большое число качественных ассоциативных связей статистическими методами, не зависящими от языковых особенностей, поэтому они были выбраны для автоматического построения русскоязычного тезауруса.

3.3. Выделение синонимов

Выбор синонимов осуществляется тремя алгоритмами, использующими соответственно расстояние Левенштейна [53]; выделение синонимических связей из словаря синонимов Synmaster (http://usyn.ru/blog.php?id_blog=11), в котором содержится около 1 200 000 слов и от одного до 20 синонимов к каждому слову; тезаурус РуТез.

Метод, основанный на расстоянии Левенштейна, ищет однокоренные слова и формы одного и того же слова, которые в тезаурусах считаются синонимами. Расстояние Левенштейна — это число изменений (удалений, добавлений или замен букв), которое требуется для преобразования одного слова в другое. В зависимости от этого расстояния и длины слов вычисляется мера близости, которая у схожих терминов будет больше, чем у других пар.

Алгоритмы, ищущие синонимы среди лингвистических ресурсов, работают по одному и тому же принципу: из словаря или тезауруса РуТез в автоматический тезаурус добавляются только те синонимические связи, которые связывают уже существующие в тезаурусе термины, новые термины в него не записываются.

3.4. Выделение иерархических связей

Иерархические, или гипонимо-гиперонимические, связи выбираются для автоматического тезауруса лингвистическими методами морфо-синтаксических правил [54] или выделения связей из РуТеза. Выделение гипонимов и гиперонимов из тезауруса РуТез происходит по тому же алгоритму, что и для синонимов.

Метод морфо-синтаксических правил считает, что термины связаны иерархическими отношениями, если один из них включает в себя второй как строку-суффикс или часть многословного термина. Первый в обоих случаях считается гипонимом, второй — гиперонимом, так как он является более общим. Например, «таможенный союз» — это гипоним к слову «союз».

3.5. Комплексная оценка тезауруса

Метрики для оценки качества тезауруса были взяты из предыдущей работы авторов [36], в которой предлагалась комплексная оценка тезауруса, построенного полностью автоматически гибридными методами. В частности, были выбраны графовые характеристики: число терминов, семантических связей различных типов и количество компонент связности графа тезауруса и размер наибольшей компоненты.

Выбор данной комплексной оценки для исследования качества тезауруса обусловлен тем, что она была разработана авторами специально для автоматически

Таблица 2: Характеристики автоматически построенных тезаурусов для корпуса статей о мигрантах

Table 2: Characteristics of automatically generated thesauri for the corpus of texts about migrants

Методы выделения			Количество выделенных				Количество	
гиперонимов	синонимов	ассоциаций	терминов	гиперонимов	синонимов	ассоциаций	компон. связности	вершин в наиб. комп.
Extraction methods for			Quantity of extracted				Quantity of	
hypernyms	synonyms	associations	terms	hypernyms	synonyms	associations	connected comp.	vertices in max. comp.
РуТез	РуТез	ЛСА	2 321	239	484	29 345	5	2 313
РуТез	Synmaster	ЛСА	2 413	239	59 844	27 335	1	2 413
Морф	Лев	word2vec	728	20	50	5 076	1	728
Гибрид	Гибрид	Гибрид	2 413	248	60 328	32 360	1	2 413

ЛСА — латентно-семантический анализ.
РуТез — метод, основанный на использовании РуТез.
Synmaster — метод, основанный на использовании Synmaster.
Лев — метод, использующий расстояние Левенштейна.
Морф — морфо-синтаксические правила.
Гибрид — совместное применение методов ЛСА и word2vec для ассоциаций, РуТез, Synmaster и Лев для синонимов, Морф и Гибрид для гипонимов и гиперонимов.

сгенерированных тезаурусов. Её главным преимуществом является возможность оценить тезаурус по нескольким аспектам сразу: качеству терминов и семантических связей, а также структуре и её связности. Второе достоинство комплексной оценки — это автоматизация её вычисления: графовые характеристики считаются полностью автоматически, не зависят от эксперта и сторонних ресурсов, а для автоматического подсчёта качественных характеристик нужен только альтернативный тезаурус выбранной предметной области как эталонный, с которым и проводится сравнение.

К сожалению, стандартные характеристики точности и полноты выделения терминов и связей в случаях с предлагаемыми тезаурусами невозможно подсчитать автоматически, так как для выбранных предметных областей не существует тезаурусов в открытом доступе, с которыми можно было бы сравнить результаты. Поэтому оценка точности выделения тезаурусных терминов и связей проводилась экспертом вручную.

3.6. Результаты экспериментов

Для экспериментов с методами выделения тезаурусных связей был построен программный стенд, реализующий описанный выше метод построения русскоязычного тезауруса. Для реализации стенда и вычисления оценочных характеристик тезауруса использовались языки программирования Python и Java и библиотеки NLTK и Gensim, в которых реализованы стандартные методы обработки данных на естественном языке.

В таблице 2 представлены графовые оценки качества тезауруса для корпуса статей о мигрантах, описанного в подразделе 3.1. Очевидно, что предложенный авторами комбинированный метод показывает лучшие характеристики тезауруса:

Таблица 3: Характеристики автоматически построенных тезаурусов для корпуса твитов

Table 3: Characteristics of automatically generated thesauri for the corpus of tweets

Методы выделения			Количество выделенных				Количество	
гиперонимов	синонимов	ассоциаций	терминов	гиперонимов	синонимов	ассоциаций	компон. связности	вершин в наиб. комп.
Extraction methods for			Quantity of extracted				Quantity of	
hypernyms	synonyms	associations	terms	hypernyms	synonyms	associations	connected comp.	vertices in max. comp.
РуТез	РуТез	ЛСА	15 629	2 808	608	105 998	333	14 605
РуТез	Synmaster	ЛСА	15 629	2 808	1 487 526	96 436	46	15 527
Морф	Лев	word2vec	527	80	445	78	49	41
Гибрид	Гибрид	Гибрид	15 629	2 888	1 488 577	96 476	46	15 527

ЛСА — латентно-семантический анализ.
РуТез — метод, основанный на использовании РуТез.
Synmaster — метод, основанный на использовании Synmaster.
Лев — метод, использующий расстояние Левенштейна.
Морф — морфо-синтаксические правила.
Гибрид — совместное применение методов ЛСА и word2vec для ассоциаций, РуТез, Synmaster и Лев для синонимов, Морф и Гибрид для гипонимов и гиперонимов.

он содержит больше всего терминов и связей всех типов, а также обеспечивает связный тезаурус.

Наименьший вклад вносят методы, основанные на морфо-синтаксических правилах и расстоянии Левенштейна, они выделяют меньше всего связей. Наибольшее число связей выделяет метод, берущий информацию из словаря Synmaster, за его счёт тезаурус оказывается состоящим из одной компоненты связности. Тезаурус, построенный при помощи методов word2vec, расстояния Левенштейна и морфо-синтаксических правил, также связный, но он существенно меньше остальных — в нём всего 728 терминов.

Для ассоциативных связей метод ЛСА существенно превосходит word2vec: он выделяет около 29 тысяч связей, часть из которых переходит в синонимические на следующих этапах алгоритма, а word2vec выбирает около 5 тысяч ассоциаций, что приблизительно в 6 раз меньше.

Гипонимо-гиперонимических связей в текстах оказалось очень мало по сравнению с остальными типами связей, и большая их часть выделена из тезауруса общего назначения РуТез.

В экспериментах с корпусом твитов (табл. 3) повторяются те же закономерности: комбинированный метод превосходит другие методы, взятые по отдельности, лучшим методом для выделения иерархических связей оказывается метод, использующий РуТез, синонимических — метод, применяющий Synmaster, а ассоциативных — ЛСА. При этом статистический метод word2vec показывает худший результат, чем для предыдущей выборки текстов: он находит существенно меньше связей, всего 78.

Таким образом, предложенный авторами метод демонстрирует лучшие характеристики качества итогового тезауруса. Отметим, что качество результата обеспечивается в первую очередь лингвистическими методами, выбирающими связи из существующих лингвистических ресурсов. Среди статистических методов самым эффективным оказался ЛСА, он в обоих случаях выделил достаточно большое число

связей. Остальные статистические методы либо выделяют малое число синонимов (метод расстояния Левенштейна), либо демонстрируют нестабильные результаты, отличающиеся от выборки к выборке (word2vec).

Экспертная оценка точности выделения терминов и связей была проведена для автоматического тезауруса, построенного комбинированным методом для корпуса статей о мигрантах, поскольку он обладает существенно меньшими размерами, чем второй, и эксперт может оценить его за разумное время. Оценка показала, что точность выделения терминов 94,3 %, всего 137 терминов оказались некорректными; синонимов — 99,9 %, а гипонимов и гиперонимов — 98,3 %, что объясняется тем, что практически все они были выделены из надёжных лексических ресурсов.

Что касается структуры автоматически сгенерированного тезауруса, то он в первую очередь состоит из синонимов и ассоциаций. Гипонимов и гиперонимов в нём оказалось существенно меньше, и эта закономерность прослеживается для обеих предметных областей. Возможной причиной этого может быть особенность корпусов текстов, которые содержат достаточно мало терминов, находящихся в иерархических отношениях между собой.

Заключение

Анализ существующих тезаурусов позволил выделить два доступных ресурса: РуТез и YARN, являющихся общими тезаурусами русского языка. Оба тезауруса могут успешно использоваться в задачах автоматической обработки текстов. И тот и другой сопоставимы по количеству слов и связей между ними, хотя и отличаются моделями основных единиц: понятие в тезаурусе РуТез и синсет в YARN.

Однако практически полное отсутствие доступных тезаурусов предметных областей показывает актуальное направление работы для создателей тезаурусов. Также следует отметить, что открытость как самих таких лексических ресурсов, так и моделей их разработки может позволить добиться повышения качества тезаурусов, а также внести существенный вклад в решение задач автоматической обработки текстов в соответствующих областях.

Методы построения русскоязычных тезаурусов также нуждаются в развитии и анализе. Здесь стоит обратить внимание на использование методов машинного обучения, комбинации статистических и лингвистических методов.

Отдельной большой проблемой является оценка методов построения тезаурусов. С точки зрения авторов работы, можно выделить два направления в развитии этих методов. Первое направление заключается в оценивании самого тезауруса с помощью как количественных характеристик содержащихся в нем слов и связей, так и с помощью оценки его внутренней структуры. Второе направление связано с оценкой тезауруса опосредованно через качество решения задач обработки текстов с его применением. Для реализации этого подхода необходима разработка открытых размеченных корпусов текстов.

Анализ современных работ в области построения русскоязычных тезаурусов и собственный опыт авторов позволяют надеяться на успешное решение обозначенных проблем в обозримом будущем.

Список литературы / References

[1] Aitchison J., Gilchrist A. and Bawden D., *Thesaurus construction and use: a practical manual*, Psychology Press, 2000, 230 pp.

[2] Сидорова Е. А., "Подход к моделированию процесса извлечения информации из текста на основе онтологии", *Онтология проектирования*, **8**:1(27) (2018), 134–151; [Sidorova E. A., "Ontology-based approach to modeling the process of extracting information from text", *Ontology of design*, **8**:1(27) (2018), 134–151, (in Russian).]

[3] Еленевская М. Н., Овчинникова И. Г., "Хранение и описание вербальных ассоциаций: словари и тезаурусы", *Вопросы психолингвистики*, 2016, №29, 69–92; [Yelenevskaya M. N., Ovchinnikova I. G., "The storage and description of the verbal associations", *Questions of psycholinguistics*, 2016, №29, 69–92, (in Russian).]

[4] Paramonov I. et al., "Thesaurus-Based Method of Increasing Text-via-Keyphrase Graph Connectivity During Keyphrase Extraction for e-Tourism Applications", Communications in Computer and Information Science, **649**, Springer, 2016, 129–141.

[5] Shchitov I., Lagutina K., Lagutina N., Paramonov I., "Sentiment classification of long newspaper articles based on automatically generated thesaurus with various semantic relationships", *Proceedings of the 21st Conference of Open Innovations Association FRUCT, University of Helsinki, Helsinki, Finland*, 2017, 290–295.

[6] Бленда Н. А., "Обзор русскоязычных тезаурусов для решения задачи расчета семантической близости между научными публикациями", *Информационные технологии и системы*, Труды Четвертой Международной научной конференции, 2015, 70–74; [Blenda N. A., "Overview of russian-language thesauri to solve the problem of calculating the semantic similarity for scientific publications", *Information Technologies and Systems*, Proceedings of the Fourth International Scientific Conference, 2015, 70–74, (in Russian).]

[7] Поршнев С. В., "О качестве открытых электронных тезаурусов русского языка", *Сборник материалов Всероссийской молодежной школы-семинара «Актуальные проблемы информационных технологий, электроники и радиотехники – 2015» (ИТЭР –2015)*, **2** (2015), 45–48; [Porshnev S. V., "O kachestve otkrytyh ehlektronnyh tezaurusov russkogo yazyka", *Sbornik materialov Vserossijskoj molodezhnoj shkoly-seminara «Aktualnye problemy informacionnyh tekhnologij, ehlektroniki i radiotekhniki – 2015» (ITER –2015)*, **2** (2015), 45–48, (in Russian).]

[8] Loukachevitch N., Dobrov B., "RuThes linguistic ontology vs. Russian wordnets", *Proceedings of the Seventh Global WordNet Conference*, 2014, 154–162.

[9] Loukachevitch N., Dobrov B., Chetviorkin I., "RuThes-Lite, a publicly available version of Thesaurus of Russian language RuThes", *Computational Linguistics and Intellectual Technologies: papers from the Annual conference "Dialogue"*, 2014, №13(20), 340–349.

[10] Loukachevitch N. V., Lashevich G., Gerasimova A. A., Ivanov V. V., Dobrov B. V., "Creating Russian WordNet by conversion", *Computational Linguistics and Intellectual Technologies: papers from the Annual conference "Dialogue"*, 2016, №15(22), 405–415.

[11] Braslavski P., Ustalov D., Mukhin M., Kiselev Y., "YARN: Spinning-in-Progress", *Proceedings of the Eight Global Wordnet Conference*, 2016, 58–65.

[12] Сухоногов А. М., Яблонский С. А., "Автоматизация построения англо-русского WordNet", *Компьютерная лингвистика и интеллектуальные технологии*, Труды Международного семинара "Диалог", 2005, 25–31; [Suhonogov A. M., Yablonskij S. A., "Avtomatizaciya postroeniya anglo-russkogo WordNet", *Computational Linguistics and Intellectual Technologies*, Papers from the Annual conference "Dialogue", 2005, 25–31, (in Russian).]

[13] Azarowa I., "RussNet as a computer lexicon for Russian", *Proceedings of the Intelligent Information systems IIS-2008*, 2008, 341–350.

[14] Азарова И. В., Захаров В. П., Киселев Ю., Усталов Д. А., Хохлова М. В., "Интеграция тезаурусов RussNet и YARN", *Компьютерная лингвистика и вычислительные онтологии*, Труды XIX Международной объединённой научной конференции «Интернет и современное общество» (IMS-2016), Санкт-Петербург, 22–24 июня 2016 г., Университет ИТМО, СПб, 2016, 7–13; [Azarova I. V., Zaharov V. P., Kiselev YU.,

Ustalov D. A., Hohlova M. V., "Integraciya tezaurusov RussNet i YARN", *Kompyuternaya lingvistika i vychislitelnye ontologii*, Trudy XIX Mezhdunarodnoj objedinyonnoj nauchnoj konferencii «Internet i sovremennoe obshchestvo» (IMS-2016), Sankt-Peterburg, 22–24 iyunya 2016 g., Universitet ITMO, SPb, 2016, 7–13, (in Russian).]

[15] Сладкова О., Пирумова Л., Пирумов А., "Информационные ресурсы Интернет для специалистов сельского хозяйства", *Международный сельскохозяйственный журнал*, 2016, № 2, 44–48; [Sladkova O., Pirumova L., Pirumov A., "Informacionnye resursy Internet dlya specialistov selskogo hozyajstva", *International Agricultural Journal*, 2016, № 2, 44–48, (in Russian).]

[16] Галиева А. М., Якубова Д. Д., "Принципы представления лексики в общественно-политическом тезаурусе татарского языка", *Филологические науки. Вопросы теории и практики*, 2016, № 12-2 (66), 80–84; [Galieva A. M., Yakubova D. D., "Principles of vocabulary presentation in socio-political thesaurus of the tatar language", *Philological Sciences. Questions of theory and practice*, 2016, № 12-2 (66), 80–84, (in Russian).]

[17] Галиева А. М., Кириллович А. В., Лукашевич Н. В., Невзорова О. А., Сулейманов Д. Ш., Якубова Д. Д., "Русско-татарский общественно-политический тезаурус: публикация в облаке лингвистических открытых связанных данных", *International Journal of Open Information Technologies*, **5**:11 (2017), 64–73; [Galieva A. M., Kirillovich A. V., Lukashevich N. V., Nevzorova O. A., Sulejmanov D. SH., YAkubova D. D., "Russian-tatar socio-political thesaurus: publishing in the linguistic linked open data cloud", *International Journal of Open Information Technologies*, **5**:11 (2017), 64–73, (in Russian).]

[18] Агеев М. С., Добров Б. В., Лукашевич Н. В., "Автоматическая рубрикация текстов: методы и проблемы", *Учён. зап. Казан. гос. ун-та. Сер. Физ.-матем. науки*, **150**:4 (2008), 25–40; [Ageev M. S., Dobrov B. V., Lukashevich N. V., "Automatic Text Categorization: Methods and Problems", *Kazan. Gos. Univ. Uchen. Zap. Ser. Fiz.-Mat. Nauki*, **150**:4 (2008), 25–40, (in Russian).]

[19] Лукашевич Н. В., Добров Б. В., Павлов А. М., Штернов С. В., "Онтологические ресурсы и информационно-аналитическая система в предметной области «Безопасность»", *Онтология проектирования*, **8**:1 (27) (2018), 74–95; [Lukashevich N. V., Dobrov B. V., Pavlov A. M., SHternov S. V., "Ontological resources and information-analytical system in security domain", *Ontology of design*, **8**:1 (27) (2018), 74–95, (in Russian).]

[20] Мишунин О. Б., Савинов А. П., Фирстов Д. И., "Проблемы, возникающие в интеллектуальных обучающих системах при оценке ответов на естественном языке", *Современные проблемы науки и образования*, 2015, № 2–2, 189–199; [Mishunin O. B., Savinov A. P., Firstov D. I., "Problems of automatic free-text answer grading in intelligent tutoring systems", *Modern problems of science and education*, 2015, № 2–2, 189–199, (in Russian).]

[21] Алексеев А. А., "Тематический анализ новостного кластера как основа тематического аннотирования", *Программная инженерия*, 2014, № 3, 41–48; [Alekseev A. A., "Thematic representation of a news cluster as a basis for summarization", *Software Engineering*, 2014, № 3, 41–48, (in Russian).]

[22] Усталов Д. А., "Обнаружение понятий в графе синонимов", *Вычислительные технологии*, **22**:S1 (2017), 99–112; [Ustalov D. A., "Concept discovery from synonymy graphs", *Computational Technologies*, **22**:S1 (2017), 99–112, (in Russian).]

[23] Kolchin M., Chistyakov A., Lapaev M., Khaydarova R., "FOODpedia: Russian food products as a linked data dataset", *International Semantic Web Conference*, 2015, 87–09.

[24] Hasan K., Vincent N., "Automatic keyphrase extraction: A survey of the state of the art", *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014, 1262–1273.

[25] Добров Б. В., Лукашевич Н. В., "Лингвистическая онтология по естественным наукам и технологиям для приложений в сфере информационного поиска", *Учен. зап. Казан. гос. ун-та. Сер. Физ.-матем. науки*, **149**:2 (2007), 49–72; [Dobrov B. V., Lukashevich N. V., "Linguistic ontology on natural sciences and technologies for information-retrieval applications", *Kazan. Gos. Univ. Uchen. Zap. Ser. Fiz.-Mat. Nauki*, **149**:2 (2007), 49–72, (in Russian).]

[26] Лукашевич Н. В., Добров Б. В., Чуйко Д. С., "Отбор словосочетаний для словаря системы автоматической обработки текстов", *Компьютерная лингвистика и интеллектуальные технологии: Тр. Международной конференции "Диалог"*, 2008, № 7(14), 339–344; [Lukashevich N. V., Dobrov B. V., Chujko D. S., "Automated analysis of multiword expressions for computational dictionaries", *Computational Linguistics and Intellectual Technologies: Papers from the Annual International Conference "Dialogue"*, 2008, № 7(14), 339–344, (in Russian).]

[27] Turney P. D., Pantel P., "From frequency to meaning: Vector space models of semantics", *Journal of artificial intelligence research*, **37** (2010), 141–188.

[28] Захаров В. П., "Корпусно-ориентированный подход к построению тезаурусов и онтологий", *Структурная и прикладная лингвистика*, 2015, № 11, 123–141; [Zakharov V. P., "Corpus-based approach to thesaurus and ontology construction", *Structural and applied linguistics*, 2015, № 11, 123–141, (in Russian).]

[29] Котова Е. Е., Писарев И. А., "Построение тематических онтологий с применением метода автоматизированной разработки тезаурусов", *Известия СПбГЭТУ «ЛЭТИ»*, 2016, № 3, 37–47; [Kotova E. E., Pisarev I. A., "Construction of thematic ontologies using the method of automated thesauri development", *Proceedings of Saint Petersburg Electrotechnical University*, 2016, № 3, 37–47, (in Russian).]

[30] Аюшеева Н. Н., Кушеева Т. Н., "Способ вычисления весовых коэффициентов вершин семантической сети научного текста", *Фундаментальные исследования*, 2012, № 6-3, 626–630; [Ayusheeva N.N., Kusheeva T.N., "Method of calculation of weight factors tops semantic network scientific text", *Fundamental Research*, 2012, № 6-3, 626–630, (in Russian).]

[31] Аюшеева Н. Н., Гомбожапова Т. Н., Доржаев Т. В., "Способ автоматического определения тематики научного текста", *Фундаментальные исследования*, 2016, № 8-2, 229–233; [Ayusheeva N. N., Gombozhapova T. N., Dorzhaev T. V., "An automatic scientific text topic identification method", *Fundamental Research*, 2016, № 8-2, 229–233, (in Russian).]

[32] Chetviorkin I, Loukachevitch N., "Extraction of Russian sentiment lexicon for product meta-domain", *Proceedings of COLING 2012*, 2012, 593–610.

[33] Loukachevitch N., Levchik A., "Creating a General Russian Sentiment Lexicon", *Proceedings of Language Resources and Evaluation Conference*, 2016, 1171–1176.

[34] Ванюшкин А. С., Гращенко Л. А., "Оценка алгоритмов извлечения ключевых слов: инструментарий и ресурсы", *Новые информационные технологии в автоматизированных системах*, **20** (2017), 95–102; [Vanyushkin A. S., Grashchenko L. A., "Ocenka algoritmov izvlecheniya klyuchevyh slov: instrumentarij i resursy", *Novye informacionnye tekhnologii v avtomatizirovannyh sistemah*, **20** (2017), 95–102, (in Russian).]

[35] Лукашевич Н. В., Логачев Ю. М., "Комбинирование признаков для автоматического извлечения терминов", *Вычислительные методы и программирование*, **11**:4 (2010), 108–116; [Lukashevich N. V., Logachev YU. M., "Automatic term extraction based on feature combination", *Vychislitel'nye metody i programmirovanie*, **11**:4 (2010), 108–116, (in Russian).]

[36] Лагутина Н. С., Лагутина К. В., Мамедов Э. И., Парамонов И. В., "Методические аспекты выделения семантических отношений для автоматической генерации специализированных тезаурусов и их оценки", *Моделирование и анализ информационных систем*, **23**:6 (2016), 826–840; [Lagutina N. S., Lagutina K. V., Mamedov E. I., Paramonov I. V., "Methodological aspects of semantic relationship extraction for automatic thesaurus generation", *Modeling and Analysis of Information Systems*, **23**:6 (2016), 826–840, (in Russian).]

[37] Лукашевич Н. В., "Квазисинонимы в лингвистических онтологиях", *Компьютерная лингвистика и интеллектуальные технологии: По материалам ежегодной Международной конференции "Диалог"*, 2010, № 9(16), 307–312; [Lukashevich N. V., "Near-synonyms in linguistic ontologies", *Computational Linguistics and Intellectual Technologies. Papers from the Annual International Conference "Dialogue"*, 2010, № 9(16), 307–312, (in Russian).]

[38] Лукашевич Н. В., "Моделирование отношений ЧАСТЬ–ЦЕЛОЕ в лингвистическом ресурсе для информационно-поисковых приложений", *Информационные технологии*, 2007, № 12, 28–34; [Lukashevich N. V., "Modeling of PART–WHOLE Relations in Linguistic Resource for Information-Retrieval Applications", *Information Technology*, 2007, № 12, 28–34, (in Russian).]

[39] Баранюк В. В., Богорадникова А. В., Смирнова О. С., "Определение семантического содержания предметной области на основе формирования тезауруса", *International Journal of Open Information Technologies*, **4**:9 (2016), 74–79; [Baranjuk V.V., Bogoradnikova A.V., Smirnova O.S., "Defining the scope semantics by forming its thesaurus", *International Journal of Open Information Technologies*, **4**:9 (2016), 74–79, (in Russian).]

[40] Нугуманова А. Б., Бессмертный И. А., Пецина П., Байбурин Е. М., "Обогащение модели Bag-of-Words семантическими связями для повышения качества классификации текстов предметной области", *Программные продукты и системы*, 2016, № 2(114), 89–99; [Nugumanova A. B., Bessmertnyj I. A., Pecina P., Bajburin E. M., "Semantic relations in text classification based on bag-of-words model", *Software products and systems*, 2016, № 2(114), 89–99, (in Russian).]

[41] Panchenko A., Ustalov D., Arefyev N., Paperno D., Konstantinova N., Loukachevitch N., Biemann C., "Human and machine judgements for russian semantic relatedness", *Analysis of Images, Social Networks and Texts. 5th International Conference, AIST 2016*, Springer, 2016, 221–235.

[42] Rapp R., "The automatic generation of thesauri of related words for English, French, German, and Russian", *International Journal of Speech Technology*, **11**:3–4 (2008), 147–156.

[43] Галина И. В., Козеренко Е. Б., Морозова Ю. И., Сомин Н. В., Шарнин М. М., "Ассоциативные портреты предметной области—инструмент автоматизированного построения систем big data для извлечения знаний: теория, методика, визуализация, возможное применение", *Информатика и её применения*, **9**:2 (2015), 92–110; [Galina I. V., Kozerenko E. B., Morozova Yu. I., Somin N. V., Sharnin M. M., "Associative portraits of subject areas as a tool for automated construction of big data systems for knowledge extraction: theory, methods, visualization, and application", *Informatics and its Applications*, **9**:2 (2015), 92–110, (in Russian).]

[44] Kuznetsov I. P., Kozerenko E. B., Charnine M. M., "Technological peculiarity of knowledge extraction for logical-analytical systems", *Proceedings of ICAI*, **12**, 2012, 18–21.

[45] Золотарев О. В., Шарнин М. М., "Методы извлечения знаний из текстов естественного языка и построение моделей бизнес-процессов на основе выделения процессов, объектов, их связей и характеристик", *Труды Международной научной конференции CPT2014*, 2015, 92–98; [Zolotarev O. V., Charnin M. M., "Methods of extracting knowledge from natural language texts and the construction of models of business processes on the basis of allocation processes, objects, their relationships and characteristics", *Proceedings of the International Scientific Conference CPT2014*, 2015, 92–98, (in Russian).]

[46] Золотарев О. В., Шарнин М. М., Клименко С. В., "Семантический подход к анализу террористической активности в сети Интернет на основе методов тематического моделирования", *Вестник Российского нового университета. Серия: Сложные системы: модели, анализ и управление*, 2016, № 3, 64–71; [Zolotarev O. V., Charnin M. M., Klimenko S. V., "A semantic approach to the analysis of terrorist activity on the internet based on the methods of topic modeling", *Bulletin of the Russian New University. Series: Complex systems: models, analysis and management*, 2016, № 3, 64–71, (in Russian).]

[47] Лагутина Н. С., Лагутина К. В., Щитов И. А., Парамонов И. В., "Анализ использования различных типов связей между терминами тезауруса, сгенерированного с помощью гибридных методов, в задачах классификации текстов", *Моделирование и анализ информационных систем*, **24**:6 (2017), 772–787; [Lagutina N. S., Lagutina K. V., Shchitov I. A., Paramonov I. V., "Analysis of Influence of Different Relations Types on the Quality of Thesaurus Application to Text Classification Problems", *Modeling and Analysis of Information Systems*, **24**:6 (2017), 772–787, (in Russian).]

[48] Sabirova K., Lukanin A., "Automatic Extraction of Hypernyms and Hyponyms from Russian Texts", *Supplementary Proceedings of the 3rd International Conference on Analysis of Images, Social Networks and Texts (AIST'2014)*, 2014, 35–40.

[49] Большакова Е. И., Иванов К. М., Сапин А. С., Шариков Г. Ф., "Система для извлечения информации из текстов на базе лексико-синтаксических шаблонов", *Пятнадцатая национальная конференция по искусственному интеллекту с международным участием*, 2016, 14–22; [Bol'shakova E. I., Ivanov K. M., Sapin A. S., SHarikov G. F., "Sistema dlya izvlecheniya informacii iz tekstov na baze leksiko-sintaksicheskih shablonov", *Pyatnadcataya nacionalnaya konferenciya po iskusstvennomu intellektu s mezhdunarodnym uchastiem*, 2016, 14–22, (in Russian).]

[50] Рабчевский Е. А., "Автоматическое построение онтологий на основе лексико-синтаксических шаблонов для информационного поиска", *Электронные библиотеки: перспективные методы и технологии, электронные коллекции*, сб. науч. тр. 11-й Всероссийской научной конференции RCDL-2009, Петрозаводск, 2009, 69–77; [Rabchevskij E. A., "Avtomaticheskoe postroenie ontologij na osnove leksiko-sintaksicheskih shablonov dlya informacionnogo poiska", *Elektronnye biblioteki: perspektivnye metody i tekhnologii, ehlektronnye kollekcii*, sb. nauch. tr. 11-j Vserossijskoj nauchnoj konferencii RCDL-2009, Petrozavodsk, 2009, 69–77, (in Russian).]

[51] Mihalcea R., Tarau P., "TextRank: Bringing order into texts", *Proceedings of Empirical Methods in Natural Language Processing – EMNLP*, ACL, Barcelona, Spain, 2004, 404–411.

[52] Wiemer-Hastings P., Wiemer-Hastings K., Graesser A., "Latent semantic analysis", *Proceedings of the 16th international joint conference on Artificial intelligence*, 2004, 1–14.

[53] Noh S., Kim S., Jung C., "A Lightweight Program Similarity Detection Model using XML and Levenshtein Distance", *FECS*, 2006, 3–9.

[54] Lefever E., Van de Kauter M., Hoste V., "Evaluation of automatic hypernym extraction from technical corpora in English and Dutch", *9th International Conference on Language Resources and Evaluation (LREC)*, 2014, 490–497.

Lagutina N. S., Lagutina K. V., Adrianov A. S., Paramonov I. V., "Russian-Language Thesauri: Automated Construction and Application For Natural Language Processing Tasks", *Modeling and Analysis of Information Systems*, **25**:4 (2018), 435–458.

DOI: 10.18255/1818-1015-2018-4-435-458

Abstract. The paper reviews the existing Russian-language thesauri in digital form and methods of their automatic construction and application. The authors analyzed the main characteristics of open access thesauri for scientific research, evaluated trends of their development, and their effectiveness in solving natural language processing tasks. The statistical and linguistic methods of thesaurus construction that allow to automate the development and reduce labor costs of expert linguists were studied. In particular, the authors considered algorithms for extracting keywords and semantic thesaurus relationships of all types, as well as the quality of thesauri generated with the use of these tools. To illustrate features of various methods for constructing thesaurus relationships, the authors developed a combined method that generates a specialized thesaurus fully automatically taking into account a text corpus in a particular domain and several existing linguistic resources. With the proposed method, experiments were conducted with two Russian-language text corpora from two subject areas: articles about migrants and tweets. The resulting thesauri were assessed by using an integrated assessment developed in the previous authors' study that allows to analyze various aspects of the thesaurus and the quality of the generation methods. The analysis revealed the main advantages and disadvantages of various approaches to the construction of thesauri and the extraction of semantic relationships of different types, as well as made it possible to determine directions for future study.

Keywords: thesaurus, semantic relationships, automatic thesaurus construction, automatic relationship extraction, keyword extraction

On the authors:
Nadezhda S. Lagutina, orcid.org/0000-0002-6137-8643, associate professor, PhD,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003, Russia, e-mail: lagutinans@rambler.ru

Ksenia V. Lagutina, orcid.org/0000-0002-1742-3240, student,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003, Russia, e-mail: lagutinakv@mail.ru

Aleksey S. Adrianov, orcid.org/0000-0002-3073-0982, student,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003, Russia, e-mail: alex.a4.25@yandex.ru

Ilya V. Paramonov, orcid.org/0000-0003-3984-8423, associate professor, PhD,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003, Russia, e-mail: Ilya.Paramonov@fruct.org