ИСП Институт Системного Программирования
Российской Академии наук

ISSN 2079-8156 (Print)
ISSN 2220-6426 (Online)

Труды Института Системного Программирования РАН

Proceedings of the Institute for System Programming of the RAS

Том 29, выпуск 2

Volume 29, issue 2

Москва 2017

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

Труды Института системного программирования РАН

Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областях системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.

Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.

Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.

Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.

The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.

Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:

УДК004.45

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.
Телефон: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Сайт: http://www.ispras.ru/proceedings/

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Tel: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Web: http://www.ispras.ru/en/proceedings/

Содержание

Table of Contents

Размер памяти для хранения упорядоченного корневого графа

И.Б. Бурдонов <igor@ispras.ru>
А.С. Косачев <kos@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В статье рассматривается размер памяти, необходимый и достаточный для хранения графа из класса неориентированных корневых упорядоченных связных графов как нумерованных, так и ненумерованных. Введение содержит основные определения и постановку задачи. Граф корневой, если одна из вершин выделена и названа корнем. Граф упорядоченный, если для каждой вершины все инцидентные ей рёбра линейно упорядочены. Граф нумерованный, если все его вершины помечены различными идентификаторами, в частности, пронумерованы целыми числами от 0 до n-1, где n число вершин графа. Два неориентированных корневых упорядоченных графа G и $G`$ считаются слабо изоморфными, если существует взаимно-однозначное соответствие вершин такое, что: 1) соответствующие вершины имеют одинаковые степени, 2) рёбра $ab$ из графа $G$ и $a`b`$ из графа $G`$, инцидентные соответствующим вершинам a и $a`$ и имеющие в этих вершинах одинаковые номера, ведут в соответствующие вершины $b$ и $b`$, 3) корни соответствуют друг другу. Для нумерованных графов при изоморфизме дополнительно требуется совпадение номеров соответствующих вершин. Графы рассматриваются с точностью до слабого изоморфизма. Показано, что память, необходимая и достаточная для хранения любого графа из указанного класса, имеет размер $\Theta(mlogn)$ для нумерованных графов, $\Theta(n+(m-n+1)logn)$ для ненумерованных графов с числом вершин n и числом рёбер m, и $\Theta(n^2logn)$ для графов без кратных рёбер и петель с числом вершин n. Также показано, что память, достаточная для хранения последовательности рёбер длины $\boldsymbol{O}(n)$ или остова графа, имеет размер $\boldsymbol{O}(nlog(n\Delta))$ или $\boldsymbol{O}(nlog\Delta)$, соответственно, где Δ максимальная степень вершины.

Ключевые слова: неориентированный граф; упорядоченный граф; нумерованный граф; корневой граф; представление графа; перечисление графов

DOI: 10.15514/ISPRAS-2017-29(2)-1

Для цитирования: Бурдонов И.Б., Косачев А.С. Размер памяти для хранения упорядоченного корневого графа. Труды ИСП РАН, vol. 29, issue 2, 2017, pp.7-26. DOI: 10.15514/ISPRAS-2017-29(2)-1

1. Введение

В задачах исследования графов часто возникает проблема определения размера памяти, необходимой для хранения графа. Понятно, что минимальный размер такой памяти определяется числом графов интересующего нас класса и представляет собой логарифм от этого числа. Однако проблема определения числа графов того или иного класса, называемая также проблемой перечисления графов, – это, как правило, достаточно трудная проблема и редко удаётся получить результаты, выражаемые простыми формулами.

На рис. 1 показаны виды графов, которые рассматриваются в данной статье. Будем обозначать: n – число вершин, m – число рёбер, Δ – максимальная степень вершины.

Говорят, что граф *нумерованный* (или помеченный), если все его вершины помечены различными идентификаторами; мы будем считать, что они пронумерованы числами от 0 до n-1. Граф *корневой*, если одна из его вершин выделена, такая вершина называется *корнем*.

По теореме Кэли число некорневых нумерованных деревьев равно n^{n-2} [1]. Поскольку корень может быть выбран n разными способами, число корневых нумерованных деревьев равно n^{n-1}. Известно число нумерованных графов без петель и кратных рёбер с заданным числом вершин n. Оно равно $2^{n(n-1)/2}$ [2]. Если допускаются петли, то число графов возрастает до $2^{n(n-1)/2+n} = 2^{n(n+1)/2}$.

Будем называть граф *упорядоченным*, если задано локальное упорядочение его рёбер: для каждой вершины a все инцидентные ей рёбра перенумерованы от 1 до $\Delta(a)$, где $\Delta(a)$ степень вершины a. В неориентированном упорядоченном графе каждое ребро получает два номера – по одному в каждом из своих концов; петля также получает два номера в инцидентной ей вершине.

В корневом дереве рёбрам можно придать естественную ориентацию от корня: ребро ab получает ориентацию $a \to b$, если вершина a лежит на пути в дереве от корня до вершины b, т.е. b потомок a. В таком ориентированном дереве корень имеет полустепень захода 0, а некорневая вершина имеет полустепень захода 1. Когда мы ниже будем говорить об ориентированных рёбрах в корневом дереве, мы будем иметь в виду именно эту ориентацию.

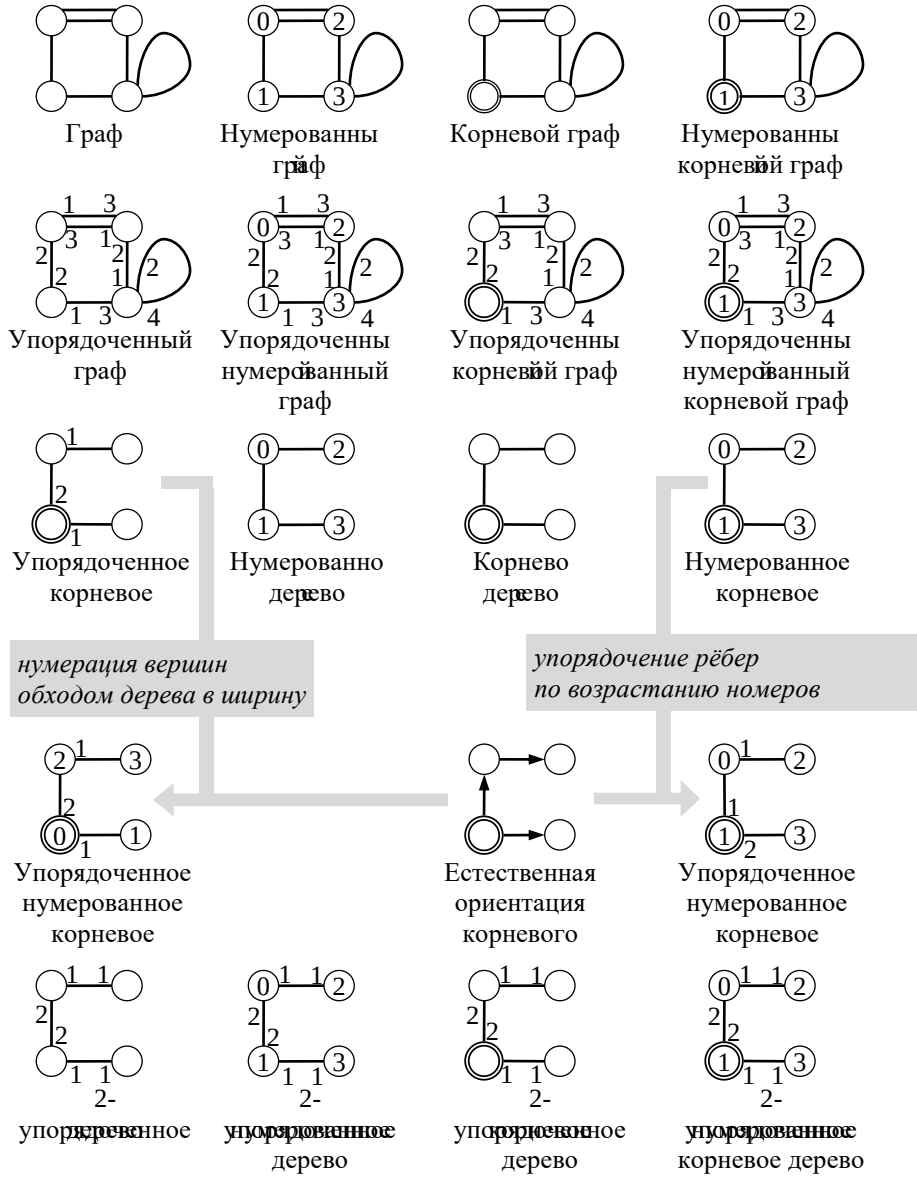


Рис. 1. Виды графов
Fig. 2. Kinds of graphs

Корневое дерево называется *упорядоченным*, если для каждой вершины *a* её потомки, т.е. концы рёбер, выходящих из *a* при естественной ориентации

рёбер, линейно упорядочены, т.е. пронумерованы от 1 до $\Delta(a)$-1. Заметим, что упорядоченное корневое дерево не является упорядоченным графом, если $n > 1$, поскольку в нем ребро $a \rightarrow b$ имеет номер только в вершине a, но не в вершине b. Однако если в вершине b ему присвоить любой номер i от 1 до $\Delta(b)$, а также увеличить на 1 каждый номер $j \geq i$ ребра, выходящего из b, то такое упорядоченное дерево станет упорядоченным графом. Мы будем называть его 2-*упорядоченным* деревом. Понятно, что одному упорядоченному дереву соответствует, вообще говоря, множество 2-упорядоченных деревьев при разных выборах i=1..$\Delta(b)$.

Число упорядоченных корневых деревьев с n вершинами – это число Каталана (последовательность A000108 в OEIS [3], называемая также *Segner numbers*) $C_{n-1} = C_{2n-2}^{n-1}/n = (2n\text{-}2)!/((n\text{-}1)!n!)$ [4][5].

Нумерованное корневое дерево задаёт естественное упорядочение рёбер, выходящих из вершины при естественной ориентации рёбер корневого дерева, по возрастанию номеров концов рёбер. Упорядоченное корневое дерево, наоборот, задает естественную нумерацию вершин обходом дерева в ширину. Но при этом получается не любое нумерованное корневое дерево: номер 0 всегда получает корень, номер 1 – вершина, которая отлична от корня и соединена с корнем ребром с минимальным номером в корне, и т.д. В общем, как видно из сравнения чисел n^{n-1} и C_{n-1}, нумерованных деревьев больше, чем упорядоченных.

Будем говорить, что два неориентированных упорядоченных графа G и $G`$ *сильно изоморфны*, если существует взаимно-однозначное соответствие вершин такое, что: 1) соответствующие вершины имеют одинаковые степени, 2) рёбра $ab$ из графа $G$ и $a`b`$ из графа $G`$, инцидентные соответствующим вершинам a и $a`$ и имеющие в этих вершинах одинаковые номера $i$=$i`$, ведут в соответствующие вершины b и $b`$ и имеют в них одинаковые номера $j$=$j`$. При *слабом изоморфизме* не требуется совпадение вторых номеров рёбер, т.е. допускается $j \neq j`$. Для корневых упорядоченных графов при изоморфизме будем дополнительно требовать, чтобы корни соответствовали друг другу. Для нумерованных упорядоченных графов при изоморфизме будем дополнительно требовать, чтобы номера соответствующих вершин совпадали.

При отсутствии кратных рёбер, в частности, для деревьев, оба изоморфизма совпадают. Действительно, в вершине $b`$ должно быть ребро с номером $j$, поскольку ребро с таким номером есть в вершине $b$, $b$ соответствует $b`$ и, следовательно, степени вершин совпадают. Если бы было $j \neq j`$, то ребро с номером $j$ должно было бы вести из $b`$ в вершину $c` \neq a`$, поскольку из $b`$ в $a`$ уже ведет ребро с другим номером $j`$, а кратных рёбер нет. Следовательно, вершины $b$ и $b`$ соответствуют друг другу, но ребро с номером j ведет из b в a и из $b`$ в $c` \neq a`$, что противоречит изоморфизму. При наличии кратных рёбер сильный и слабый изоморфизм различаются, пример приведён на рис. 2.

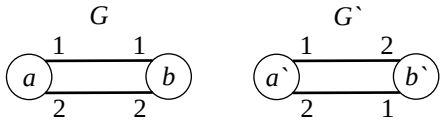


Рис. 2. Граф G слабо (но не сильно!) изоморфен графу G`
Fig. 2. The graph G is weakly (but not strongly!) isomorphic to the graph G`

Если $\boldsymbol{G}$ некоторый класс неориентированных упорядоченных графов, то через $N(\boldsymbol{G})$ обозначим число графов в нём с точностью до слабого изоморфизма.

В данной статье рассматриваются классы $\boldsymbol{G}_1(n,m)$ нумерованных и $\boldsymbol{G}_2(n,m)$ ненумерованных неориентированных упорядоченных корневых связных графов с n вершинами и m рёбрами (петли и кратные рёбра допускаются), а также классы $\boldsymbol{G}_1(n)$ нумерованных и $\boldsymbol{G}_2(n)$ ненумерованных графов с n вершинами без кратных рёбер и петель. Мы дадим точную по порядку оценку размера памяти, необходимой и достаточной для хранения любого графа из этих классов с точностью до слабого изоморфизма: $N(\boldsymbol{G}_1(n,m)) = \boldsymbol{\Theta}(mlogn)$, $N(\boldsymbol{G}_2(n,m)) = \boldsymbol{\Theta}(n+(m-n+1)logn)$, $N(\boldsymbol{G}_1(n)) = N(\boldsymbol{G}_2(n)) = \boldsymbol{\Theta}(n^2logn)$.

2. Нумерованные графы

В этом разделе мы рассмотрим графы из класса $\boldsymbol{G}_1(n,m)$: нумерованные неориентированные упорядоченные корневые связные графы с n вершинами и m рёбрами. Мы дадим оценку сверху $\boldsymbol{O}(mlogn)$ для размера $l_1(n,m)$ памяти, достаточной для хранения графа из этого класса, и оценку снизу для числа $N_1(n,m) = N(\boldsymbol{G}_1(n,m))$ графов этого класса с точностью до слабого изоморфизма, равную $\boldsymbol{\Omega}(n^m)$, что даст оценку снизу $\boldsymbol{\Omega}(mlogn)$ для размера $L_1(n,m)$ памяти, необходимой для хранения графа из этого класса. Поскольку эти оценки размера памяти совпадают, размер памяти, необходимой и достаточной для хранения графов из класса $\boldsymbol{G}_1(n,m)$, равен $\boldsymbol{\Theta}(mlogn)$.

Если граф нумерованный, мы для краткости будем говорить «вершина a» вместо «вершина с номером a». Через $\Delta_G(a)$ будем обозначать степень вершины a в графе G. Для нумерованного упорядоченного корневого графа G будем обозначать: $root(G)$ – номер корня графа G; (a,i) – ребро, инцидентное вершине a и имеющее в ней номер i; $\delta_G(a,i)$ – другой конец ребра (a,i); $\nu_G(a,i)$ – номер ребра (a,i) в вершине $\delta_G(a,i)$. По определению $\delta_G(\delta_G(a,i),\nu_G(a,i)) = a$ и $\nu_G(\delta_G(a,i),\nu_G(a,i)) = i$. Если граф G подразумевается, то его указание в нижнем индексе будем опускать.

2.1. Представление графа в памяти

В этом подразделе мы опишем способ представления графа из класса $\boldsymbol{G}_1(n,m)$ в памяти размером $l_1(n,m) = \boldsymbol{O}(mlogn)$. Таких представлений может быть несколько, мы дадим одно из них (рис. 3).

Мы будем применять два представления целых неотрицательных чисел. Обозначим $r(x) = 1$, если $x = 0$, и $r(x) = \lfloor lbx \rfloor + 1$, если $x > 0$, где $\lfloor z \rfloor$ обозначает ближайшее к z снизу целое число, а $lbx = log_2 x$ – двоичный логарифм. Если число x – это любое число из интервала $[0,p]$, где p известно, то будем использовать двоичный код фиксированной длины $r(p)$. Такое представление для $x = \Sigma\{x_i \cdot 2^{r(p)-i} \mid i=1..r(p)\}$, где $x_i \in \{0,1\}$, обозначим $D_p(x) = x_1,x_2,...x_{r(p)}$. Оно занимает $r(p)$ бит памяти. Если число x любое, не ограниченное сверху известной величиной, то минимально возможная разрядность его двоичного кода равна $r(x)$. В этом случае в представлении числа x каждый его разряд будем предварять кодом 0, а в конце поместим код 1. Такое представление для $x = \Sigma\{x_i \cdot 2^{r(x)-i} \mid i=1..r(x)\}$, где $x_i \in \{0,1\}$, обозначим $D(x) = 0,x_1,0,x_2,... 0,x_{r(x)},1$. Оно занимает $2r(x)+1$ бит памяти.

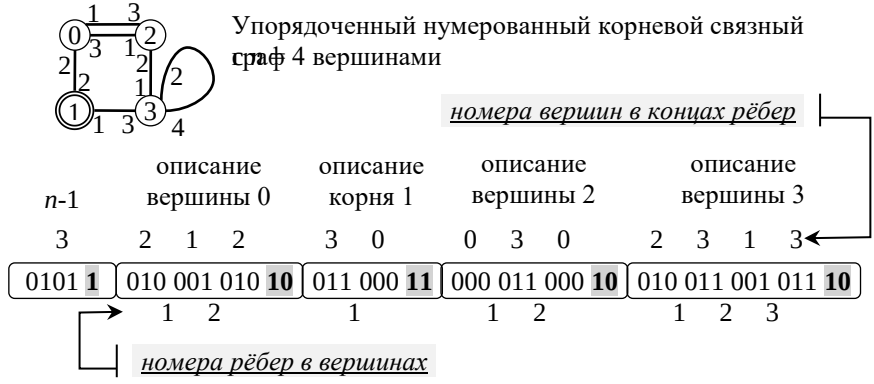


Рис. 3. Описание нумерованного графа
Fig. 3. Description of the numbered graph

Представление графа – это битовая последовательность:

D(*n*-1), *Вершина*(0), *Вершина*(1),... *Вершина*(*n*-1).

Здесь *Вершина*(*a*), где *a*=0..*n*-1, является описанием вершины *a*. Оно представляет собой битовую последовательность

Ребро(*a*,1), *Ребро*(*a*,2), ... *Ребро*(*a*,Δ(*a*)), *Ограничитель вершины*.

Здесь *Ребро*(*a*,*i*) – описание конца ребра (*a*,*i*), представляющее собой битовую последовательность $0,D_{n-1}(\delta(a,i))$. Это описание занимает $1+r(n-1)$ бит. *Ограничитель вершины* – это двухбитовый код 10, если вершина с номером *a* – это не корень, и 11 для корня.

Заметим, что каждое ребро (*a*,*i*) (включая петли) описывается дважды: в вершине *a*, где указывается вершина δ(*a*,*i*), и в вершине δ(*a*,*i*), где указывается вершина *a*; петля (δ(*a*,*i*) = *a*) также дважды описывается, поскольку она тоже имеет два номера *i* и ν(*a*,*i*), но оба в вершине *a*. Тем самым, число описаний *Ребро*(*a*,*i*) равно 2*m*.

При таком описании графа для любого ребра (a,i) мы можем по описанию *Вершина*(a) узнать вершину $\delta(a,i)$, в которую ведет это ребро. Однако, по описанию *Вершина*$(\delta(a,i))$ мы не можем узнать, какой номер $\nu(a,i)$ это ребро имеет в вершине $\delta(a,i)$, но только подмножество номеров рёбер, одним из которых является номер $\nu(a,i)$. Это подмножество номеров j таких, что в описании *Вершина*$(\delta(a,i))$ указана вершина a как другой конец ребра j: $\delta(\delta(a,i),j) = a$. Таким образом, описание определяет граф с точностью до слабого изоморфизма.

Заметим, что таким способом можно описать любой нумерованный упорядоченный корневой граф, а не только связный. Если граф не корневой, то *Ограничитель вершины* может быть однобитовым кодом 1, что уменьшает описание на n бит.

При $n \ge 2$ описание графа занимает память размером

$l_1(n,m) = 2r(n-1)+1 + 2m(1+r(n-1)) + 2n = 2r(n-1) + 2mr(n-1) + 2m + 2n + 1.$

При $n =1$, имеем $l_1(1,m) = 2\cdot1 + 2m\cdot1 + 2m + 2\cdot1 + 1 = 4m + 5.$

При $n \ge 2$ имеем $l_1(n,m) = 2(\lfloor lb(n-1)\rfloor+1) + 2m(\lfloor lb(n-1)\rfloor+1) + 2m + 2n + 1 =$

$= 2\lfloor lb(n-1)\rfloor + 2m\lfloor lb(n-1)\rfloor + 4m + 2n + 3.$

В частности, $l_1(2,m) = 2\cdot0 + 2m\cdot 0 + 4m + 2\cdot2 + 3 = 4m + 7 > l_1(1,m).$

Поскольку $lb(n-1) < lbn$ и $\lfloor x\rfloor \le x$, имеем $l_1(n,m) < 2lbn + 2mlbn + 4m + 2n + 3.$

Поскольку в связном графе $n \le m+1$, имеем

$l_1(n,m) < 2lbn + 2mlbn + 4m + 2(m+1) +3 = 2lbn + 2mlbn + 6m + 5.$

Для $m \ge 1$ имеем $l_1(n,m) \le 2mlbn + 2mlbn + 6m + 5m = 4mlbn + 11m.$

Для $n \ge 2$ имеем $1 \le lbn$ и поэтому $l_1(n,m) \le 4mlbn + 11mlbn = 15mlbn.$

Итак, для $m \ge 1$ и $n \ge 2$ имеем $l_1(n,m) \le 15mlbn$, т.е. $l_1(n,m) = \boldsymbol{O}(mlogn)$.

2.2. Хранение графов с меньшим числом вершин и/или рёбер

Функция $l_1(n,m)$ монотонно неубывающая по каждой переменной n и m: $l_1(1,m) < l_1(1,m+1)$, $l_1(1,m) < l_1(2,m)$ и, если $n \ge 2$, то $l_1(n,m) \le l_1(n,m+1)$ и $l_1(n,m) \le l_1(n+1,m)$. Поэтому память размером $l_1(n,m)$ достаточна для хранения любого нумерованного упорядоченного корневого графа с числом вершин $n` \le n$ и числом рёбер $m` \le m$.

2.3. Число графов

В этом подразделе мы оценим снизу число $N_1(n,m)$ графов в классе $\boldsymbol{G}_1(n,m)$ с точностью до слабого изоморфизма и, соответственно, размер $L_1(n,m)$ памяти, необходимой для хранения таких графов. Мы построим класс графов $\boldsymbol{G}_1^2(n,m) \subseteq \boldsymbol{G}_1(n,m)$ и подсчитаем число графов в нем (рис. 4).

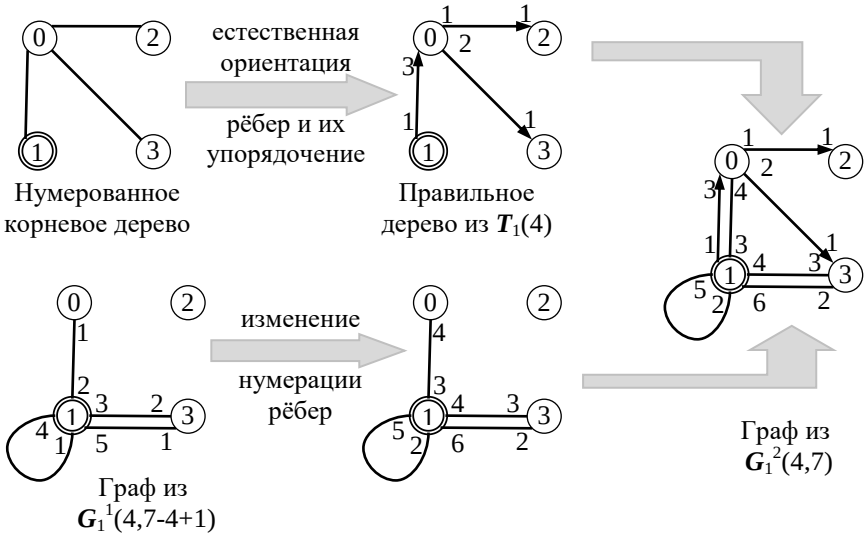


Рис. 4. Построение класса графов $G_1^2(n,m)$ для n=4 и m=7
Fig. 4. The construction of the graph class $G_1^2(n,m)$ for n=4 and m=7

Правильным деревом будем называть 2-упорядоченное нумерованное корневое дерево, в котором при естественной ориентации рёбер ребро $b \to a$, входящее в вершину a, имеет в вершине a номер $\Delta(a)$, а последовательности рёбер, выходящих из вершины a, в порядке их нумерации в вершине a от 1 до $\Delta(a)$-1 соответствует возрастающая последовательность номеров конечных вершин этих рёбер: $\forall a=0..n\text{-}1\ \forall i,j=1..\Delta(a)\text{-}1\ (\ i < j \Rightarrow \delta(i) < \delta(j)\ )$.

Правильное дерево однозначно получается из заданного нумерованного корневого дерева преобразованием f_1, которое нумерует рёбра в вершинах в соответствии с номерами конечных вершин рёбер при естественной ориентации рёбер. А именно: для каждой некорневой вершины a все рёбра, выходящие из a, т.е. рёбра вида $a \to b$, нумеруются от 1 до $\Delta(a)$-1 по порядку возрастания номеров концов c этих рёбер, а входящее ребро $c \to a$ получает в вершине a номер $\Delta(a)$; в корне e нет входящего ребра, а все выходящие рёбра перенумерованы от 1 до $\Delta(e)$ также по порядку возрастания номеров концов этих рёбер.

Очевидно, что операция удаления из правильного дерева нумерации рёбер в вершинах определена для любого правильного дерева и обратна к преобразованию f_1. Поэтому отображение f_1 биективно. Следовательно, число правильных деревьев с n вершинами равно числу нумерованных корневых деревьев с n вершинами. Множество правильных деревьев с n вершинами обозначим $\boldsymbol{T}_1(n)$. Тогда по теореме Кэли $\boldsymbol{T}_1(n) = n^{n-1}$.

Рассмотрим следующую операцию добавления к правильному дереву $T \in \boldsymbol{T}_1(n)$ графа $G \in \boldsymbol{G}_1(n,m\text{-}n\text{+}1)$. Сначала изменим в G нумерацию рёбер: для каждого ребра (a,i) его номер i увеличим на $\Delta_T(a)$. После этого добавим рёбра графа G к остову T, используя взаимно-однозначное соответствие вершин T и G по номерам вершин. В качестве корня выберем корень дерева T. Результат этой операции обозначим через $T+G$. Формально:

$root(T+G) = root(T)$, $\forall a=0..n\text{-}1\ \Delta_{T+G}(a) = \Delta_T(a) + \Delta_G(a)$,

$\forall i=1..\Delta_T(a)\ \delta_{T+G}(a,i) = \delta_T(a,i)\ \&\ \nu_{T+G}(a,i) = \nu_T(a,i)$,

$\forall i=\Delta_T(a)\text{+}1..\Delta_{T+G}(a)\ \delta_{T+G}(a,i) = \delta_G(a,i\text{-}\Delta_T(a))\ \&\ \nu_{T+G}(a,i) = \nu_G(a,i\text{-}\Delta_T(a))$.

Если T изоморфно $T`$, а $G$ слабо изоморфно $G`$, то

$root(T+G) = root(T) = root(T`) = root(T`+G)$,

$\forall a=0..n\text{-}1\ \Delta_T(a) = \Delta_{T`}(a)\ \&\ \Delta_G(a) = \Delta_{G`}(a)\ \&$

$\forall i=1..\Delta_T(a)\ \delta_T(a,i) = \delta_{T`}(a,i)\ \&\ \forall i=1..\Delta_G(a)\ \delta_G(a,i) = \delta_{G`}(a,i)$.

Отсюда непосредственно следует, что $T+G$ слабо изоморфно $T`+G`$.

Если G не слабо изоморфно $G`$, то

$\exists a=0..n\text{-}1\ \Delta_G(a) \neq \Delta_{G`}(a) \vee \Delta_G(a) = \Delta_{G`}(a)\ \&\ \exists i=1..\Delta_G(a)\ \delta_G(a,i) \neq \delta_{G`}(a,i)$.

Поэтому, если T изоморфно $T`$, а $G$ не слабо изоморфно $G`$, то $T+G$ не слабо изоморфно $T`+G`$.

Если T не изоморфно $T`$, то 1) $root(T) \neq root(T`) \vee \exists a=0..n\text{-}1$

2) $\Delta_T(a) \neq \Delta_{T`}(a) \vee$ 3) $\Delta_T(a) = \Delta_{T`}(a)\ \&\ \exists i=1..\Delta_T(a)\ \delta_T(a,i) \neq \delta_{T`}(a,i)$.

Если выполнено условие 1, то $root(T+G) \neq root(T`+G)$, т.е. графы $T+G$ и $T`+G`$ не слабо изоморфны. Если для некоторого $a=0..n\text{-}1$ выполнено условие 3, то $\delta_{T+G}(a,i) = \delta_T(a,i) \neq \delta_{T`}(a,i) = \delta_{T`+G`}(a,i)$, т.е. графы $T+G$ и $T`+G`$ не слабо изоморфны. Пусть для любого $a=0..n\text{-}1$ условие 3 не выполнено, но для некоторого $a=0..n\text{-}1$ выполнено условие 2: $\Delta_T(a) \neq \Delta_{T`}(a)$. Поскольку сумма степеней вершин дерева равна удвоенному числу его рёбер, т.е. $2(n\text{-}1)$, то хотя бы одна из таких вершин $a$ не является корнем. Тогда, поскольку условие 3 не выполнено, в деревьях $T$ и $T`$ существует общий путь ненулевой длины от корня до некоторой вершины a, в котором степени всех вершин, кроме a, совпадают в T и $T`$, а в $a$ не совпадают. Пусть для определенности $\Delta_T(a) < \Delta_{T`}(a)$. Рассмотрим ребро $(a,\Delta_T(a))$ в T и в $T`+G`$. Поскольку дерево T правильное, это ребро $b \to a$ входящее в вершину a, т.е. $\delta_T(a,\Delta_T(a)) = b$. Поскольку $\Delta_T(a) < \Delta_{T`}(a)$, в $T`$ тоже есть ребро $(a,\Delta_T(a))$, но там оно не может заканчиваться в b, поскольку дерево $T`$ тоже правильное и в $b$ должно заканчиваться ребро $(a,\Delta_{T`}(a))$. Поскольку $\Delta_T(a) \leq \Delta_T(a) < \Delta_{T`}(a)$, имеем $\delta_{T+G}(a,\Delta_T(a)) = \delta_T(a,\Delta_T(a)) \neq \delta_{T`}(a,\Delta_T(a)) = \delta_{T`+G`}(a,\Delta_T(a))$, т.е. графы $T+G$ и $T`+G`$ не слабо изоморфны.

Итак, мы доказали, что $T+G$ и $T`+G`$ слабо изоморфны тогда и только тогда, когда T изоморфно $T`$, а $G$ слабо изоморфно $G`$.

Теперь рассмотрим класс $\boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1)$ графов, в которых все рёбра инцидентны корню, имеющему номер 0. Каждое из $m\text{-}n\text{+}1$ рёбер, инцидентных корню, можно провести независимым образом в любую из n вершин. Следовательно, $N(\boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1)) = n^{m-n+1}$. Для ребра, ведущего из корня a в вершину $b\neq a$, указывается только его номер в a и вершина b. После определения всех таких кратных рёбер ab они должны быть упорядочены в вершине b, но способ упорядочивания мы не фиксируем. Для петли с номером i в корне её другой номер $\nu(a,i)$ – это один из номеров $j \neq i$, выделенных в корне для петель, но какой именно номер, не указывается. Поэтому таким способом мы определяем нумерованные упорядоченные корневые графы с точностью до слабого изоморфизма.

Обозначим: $\boldsymbol{G}_1^{\,2}(n,m) = \boldsymbol{T}_1(n)+\boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1) = \{T+G \mid T\in\boldsymbol{T}_1(n)\ \&\ G\in\boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1)\}$. В силу доказанного, число графов в классе $\boldsymbol{G}_1^{\,2}(n,m)$ с точностью до слабого изоморфизма равно произведению числа деревьев в классе $\boldsymbol{T}_1(n)$ с точностью до изоморфизма и числа графов в классе $\boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1)$ с точностью до слабого изоморфизма: $N_1(n,m) \geq N(\boldsymbol{G}_1^{\,2}(n,m)) = N(\boldsymbol{T}_1(n))N(\boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1)) = n^{n-1}n^{m-n+1} = n^m$.

Тем самым, для хранения графов из класса $\boldsymbol{G}_1(n,m)$ требуется память $L_1(n,m) = \lfloor lb(N_1(n,m)\text{-}1)\rfloor\text{+}1 \geq \lfloor lb(n^m\text{-}1)\rfloor\text{+}1$. Поскольку $\lfloor x\rfloor \geq x\text{-}1$, и $n^m\text{-}1 \geq n^{m-1}$ при $n \geq 2$ и $m \geq 1$, имеем $L_1(n,m) \geq lb(n^m\text{-}1)\text{-}1\text{+}1 \geq lbn^{m-1} = (m\text{-}1)lbn$, т.е. $L_1(n,m) = \boldsymbol{\Omega}(mlogn)$.

3. Ненумерованные графы

В этом разделе мы рассмотрим графы из класса $\boldsymbol{G}_2(n,m)$: ненумерованные неориентированные упорядоченные корневые связные графы с n вершинами и m рёбрами. Поскольку графы связные, $n \leq m\text{+}1$ и $m\text{-}n\text{+}1 \geq 0$. Мы дадим оценку сверху $\boldsymbol{O}(n\text{+}(m\text{-}n\text{+}1)logn)$ для размера $l_2(n,m)$ памяти, достаточной для хранения графа из этого класса, и оценку снизу для числа $N_2(n,m) = N(\boldsymbol{G}_2(n,m))$ графов этого класса с точностью до слабого изоморфизма, равную $\boldsymbol{\Omega}(2^n n^{m-n+1})$, что даст оценку снизу $\boldsymbol{\Omega}(n\text{+}(m\text{-}n\text{+}1)logn)$ для размера $L_2(n,m)$ памяти, необходимой для хранения графа из этого класса. Поскольку эти оценки совпадают, размер памяти, необходимой и достаточной для хранения графов из класса $\boldsymbol{G}_2(n,m)$, равен $\boldsymbol{\Theta}(n\text{+}(m\text{-}n\text{+}1)logn)$.

3.1. Представление графа в памяти

В этом подразделе мы опишем способ представления графа из класса $\boldsymbol{G}_2(n,m)$ в памяти размером $l_2(n,m) = \boldsymbol{O}(n\text{+}(m\text{-}n\text{+}1)logn)$. Таких представлений может быть несколько, мы дадим одно из них (рис. 5). Это представление похоже на представление графа из подраздела 2.1 для нумерованных графов. Для этого используется естественная нумерация вершин ненумерованного упорядоченного корневого графа, основанная на обходе графа в ширину. Опишем эту процедуру нумерации вершин.

Корень получает номер 0. Выбирается первое ребро, инцидентное корню и не являющееся петлей; его конец получает номер 1. Далее выбирается

следующее ребро, инцидентное корню и не ведущее в вершины 0 и 1; его конец получает номер 2. И так далее. Когда исчерпаются все рёбра, инцидентные корню, аналогичная процедура выполняется для вершины 1.

В общем, на каждом шаге у нас определены номера вершин 0,...,*a*` и перебраны все рёбра, инцидентные вершинам 0,...,*a*-1, где $a < a$`, а для вершины *a* перебраны рёбра с номерами 1,...,*i*-1. Если $i \le \Delta(a)$, то для вершины *a* рассматривается ребро с номером *i*. Если оно ведет в одну из вершин, уже получивших номера, т.е. в вершины 0,...,*a*`, то переходим к следующему ребру *i*+1. В противном случае конец ребра с номером *i* получает номер *a*`+1. Когда все рёбра, инцидентные вершине с номером *a*, будут перебраны, т.е. когда $i > \Delta(a)$, переходим к следующей нумерованной вершине *a*+1, если $a < a$`. Если же $a = a$`, то процедура нумерации вершин закончена.

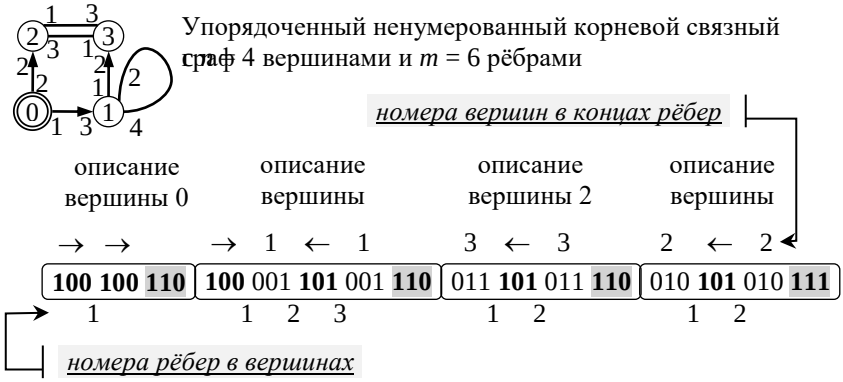


Рис. 5. Описание ненумерованного упорядоченного графа (показана естественная нумерация вершин)
Fig. 5. Description of the unnumbered ordered graph (natural numbering of vertices is shown)

В получившемся нумерованном графе однозначно выделяется корневой остов: ребро (*a*,*i*) принадлежит остову тогда и только тогда, когда вершина $\delta(a,i)$ получает свой номер при просмотре именно этого ребра в процедуре нумерации. Формально это означает, что из вершин, номера которых меньше *a*, ни одно ребро не ведет в вершину $\delta(a,i)$, т.е. $c < a \Rightarrow \forall j{=}1..\Delta(c)\ \delta(c,j) \ne \delta(a,i)$, и рёбра, инцидентные вершине *a* и имеющие в ней номера меньшие *i*, также не ведут в вершину $\delta(a,i)$, т.е. $\forall j{=}1..i{-}1\ \delta(a,j) \ne \delta(a,i)$. Заметим, что таким образом выделяется корневой остов только при естественной нумерации вершин, описанной выше.

Теперь мы уже можем использовать представление графа, описанного в подразделе 2.1. Отличие будет в том, что для рёбер выделенного остова мы не будем указывать номера вершин. Пусть (*a*,*i*) ребро остова. Тогда описание *Ребро*(*a*,*i*) будет не последовательностью $0,D_{n-1}(\delta(a,i))$, а трёхбитовым кодом

100, а описание *Ребро*(δ(*a*,*i*),ν(*a*,*i*)) будет не последовательностью $0,D_{n-1}(a)$, а кодом 101. Тем самым, описание ребра остова будет занимать 3 бита, число таких описаний равно 2(*n*-1). Описание не остовного ребра, по-прежнему, занимает 1+*r*(*n*-1) бит; число таких описаний равно 2(*m*-*n*+1). Ограничителем вершин будет код 110, занимающий также 3 бита. Специальная разновидность ограничителя для корневой вершины не требуется, поскольку корень имеет фиксированный номер 0. Зато мы можем использовать трёхбитовый код 111 вместо кода 110 для ограничителя последней вершины в списке вершин и, тем самым, не указывать число вершин.

В итоге описание графа занимает память размером

$l_2(n,m) = 2(n-1)\cdot 3 + 2(m-n+1)(1+r(n-1)) + 3n = 2(m-n+1)(1+r(n-1)) + 9n - 6.$

При $n=1$, имеем $l_2(1,m) = 2(m-1+1)\cdot(1+1) + 9\cdot 1 - 6 = 4m + 6.$

При $n \geq 2$ имеем $l_2(n,m) = 2(m-n+1)(1+\lfloor lb(n-1)\rfloor+1) + 9n - 6 =$

$= 2(m-n+1)(\lfloor lb(n-1)\rfloor+2) + 9n - 6.$

В частности, $l_2(2,m) = 2(m-2+1)(0+2) + 9\cdot 2 - 6 = 4m + 8 > l_2(1,m)$.

Поскольку $lb(n-1) < lbn$ и $\lfloor x\rfloor \leq x$, имеем

$l_2(n,m) < 2(m-n+1)(lbn+2) + 9n - 6.$

Для $n \geq 2$ имеем $1 \leq lbn$ и, поскольку также $-6 < 0$, получаем

$l_2(n,m) < 2(m-n+1)(lbn+2lbn) + 9n = 6(m-n+1)lbn + 9n.$

Поскольку в связном графе $m-n+1 \geq 0$, имеем $l_2(n,m) < 9((m-n+1)lbn + n)$.

Итак, для $n \geq 2$ имеем $l_2(n,m) < 9((m-n+1)lbn + n)$.

Следовательно, $l_2(n,m) = \boldsymbol{O}(n+(m-n+1)logn)$.

3.2. Хранение графов с меньшим числом вершин и/или рёбер

Для связного графа, когда $m-n+1 \geq 0$, функция $l_2(n,m)$ монотонно неубывающая по m: $l_2(n,m) \leq l_2(n,m+1)$. Также $l_2(1,m) < l_2(2,m)$, но, в отличие от функции $l_1(n,m)$, функция $l_2(n,m)$ не является монотонно неубывающей по n при $n \geq 2$. Например,

$l_2(9,9) = 2(9-9+1)(\lfloor lb(9-1)\rfloor+2) + 9\cdot 9 - 6 = 2\cdot 1\cdot(3+2) + 75 = 85$, но

$l_2(10,9) = 2(9-10+1)(\lfloor lb(10-1)\rfloor+2) + 9\cdot 10 - 6 = 84.$

Поэтому для того чтобы размер памяти был достаточен для хранения графов с меньшим или равным числом вершин и/или рёбер, т.е. графов из объединения $\cup\{\boldsymbol{G}_2(n`,m`) \mid 2 \leq n` \leq n \;\&\; 0 \leq m` \leq m \;\&\; n` \leq m`+1\}$ при $n \leq m+1$, нужно найти такое число вершин $n\hat{}$, на котором функция $l_2(n`,m)$ достигает максимума при заданном $m$ и в интервале $2 \leq n` \leq n$. Искомый размер памяти равен $l_2(n\hat{},m)$.

Как найти $n\hat{}$? Обозначим $x = lb(n`-1)$. Тогда $n` = 2^x+1$. Если $n`$ целое число в интервале $2 \leq n` \leq n$, то x вещественное число в интервале $0 \leq x \leq lb(n-1)$. Подставляя 2^x+1 вместо $n`$, имеем $l_2(n`,m) = 2(m-2^x)(\lfloor x\rfloor+2) + 9(2^x+1) - 6$. Обозначим $l_2`(x,m) = 2(m-2^x)(x+2) + 9(2^x+1) - 6$. Обозначим через $x\hat{}$ значение $x$, при котором функция $l_2`(x,m)$ на интервале $0 \leq x \leq lb(n-1)$ принимает

максимум. Программа Maple2015 и программа решения уравнений на сайте «Решение математики онлайн» [7] дают одно решение: $x\wedge = (W((m+1)e2^{1/2}/8) - 1 + 2.5ln2)/ln2$, где W функция Ламберта. Значение $2^{x\wedge}+1$ не обязательное целое, но $n_1 = max\{\lfloor 2^{x\wedge}+1\rfloor, 2\} \le 2^{x\wedge}+1 \le min\{\lceil 2^{x\wedge}+1\rceil, n\} = n_2$. Тогда искомое значение $n\wedge$ есть n_1 или n_2 в зависимости от того, на каком из этих двух значений n_1 или n_2 функция $l_2(n`,m)$ принимает максимум: если $l_2(n_1,m) > l_2(n_2,m)$, то $n\wedge = n_1$, иначе $n\wedge = n_2$.

3.3. Число графов

В этом подразделе мы оценим снизу число $N_2(n,m)$ графов в классе $\boldsymbol{G}_2(n,m)$ с точностью до слабого изоморфизма и, соответственно, размер $L_2(n,m)$ памяти, необходимой для хранения таких графов.

Упорядоченность графа, заданного с точностью до слабого изоморфизма, однозначно определяет естественную нумерацию вершин, основанную на обходе графа в ширину. Поэтому любая нумерация, задающая нумерованный упорядоченный граф из $\boldsymbol{G}_1(n,m)$, есть, фактически, перенумерация графа из $\boldsymbol{G}_2(n,m)$. Тем самым, $N_1(n,m) = N_2(n,m)\cdot n!$, где $n!$ – число различных нумераций вершин при фиксированном линейном порядке вершин. Поскольку $N_1(n,m) \ge n^m$, имеем $N_2(n,m) \ge n^m/n! \ge n^m e^n/(n+1)^{n+1}$. При $n \ge 35$ имеем $(n+1)^{n+1} \le n^{n+2}$ и поэтому $N_2(n,m) \ge n^m e^n/n^{n+2} = n^{m-n-2}e^n$. Отсюда $L_2(n,m) = \lfloor lb(N_2(n,m)-1)\rfloor+1 = \lfloor lb(n^{m-n-2}e^n-1)\rfloor+1 \ge \lfloor lb(n^{m-n-2}e^{n-1})\rfloor+1 \ge lb(n^{m-n-2}e^{n-1})-1+1 = (m-n-2)lbn+(n-1)lbe = (m-n+1)lbn-3lbn+(n-1)lbe \ge (m-n+1)lbn+n/2 \ge (1/2)((m-n+1)lbn+n)$. Поэтому $L_2(n,m) = \Omega(n+(m-n+1)lbn)$.

Мы также докажем это утверждение конструктивно, построив класс графов $\boldsymbol{G}_2^2(n,m) \subseteq \boldsymbol{G}_2(n,m)$ и подсчитав число графов в нем (рис. 6).

Как сказано во введении, число упорядоченных корневых деревьев с n вершинами – это число Каталана: $C_{n-1} = C_{2n-2}^{n-1}/n = (2n-2)!/((n-1)!n!)$.

В [6] доказана оценка снизу: $C_{n-1} > 2^{2n-3}(4n-5)^{1/2}/((n-1)n\pi^{1/2})$.

Обозначим: $A(n) = 2^{n-3}(4n-5)^{1/2}$, $B(n) = (n-1)n\pi^{1/2}$. Тогда $C_{n-1} > 2^n A(n)/B(n)$.

Функция $A(n)$ при $n \ge 2$ увеличивается больше чем вдвое при увеличении аргумента на 1: $A(n+1) > 2A(n)$. Действительно, $A(n+1) = 2^{n+1-3}(4(n+1)-5)^{1/2} = 2\cdot 2^{n-3}(4(n+1)-5)^{1/2} > 2\cdot 2^{n-3}(4n-5)^{1/2} = 2A(n)$.

Функция $B(n)$ при $n \ge 3$ увеличивается не более чем вдвое при увеличении аргумента на 1: $B(n+1) \le 2B(n)$. Действительно, $n \ge 3$ влечет $2n-2 \ge n+1$, что при $n>0$ влечет $2n(n-1) \ge n(n+1)$, что влечет $2B(n) = 2n(n-1)\pi^{1/2} \ge n(n+1)\pi^{1/2} = B(n+1)$.

Тем самым, при $n \ge 3$ имеем $A(n+1)/B(n+1) > 2A(n)/2B(n)$, т.е. функция $A(n)/B(n)$ монотонно увеличивается.

При $n=7$ имеем $A(7)/B(7) = 2^{7-3}(4\cdot 7-5)^{1/2}/(7-1)\cdot 7\pi^{1/2} = 8\cdot 23^{1/2}/21\pi^{1/2} > 1$.

Следовательно, при $n \ge 7$ имеем $C_{n-1} > 2^n A(n)/B(n) > 2^n$.

Упорядоченное дерево однозначным образом превращается в правильное дерево (см. подраздел 2.3) следующим преобразованием f_2. Сначала, используя естественную ориентацию корневого дерева, каждому ребру, входящему в вершину a, присвоим в этой вершине номер $\Delta(a)$. Напомним, что при естественной ориентации корневого дерева в каждую вершину, кроме корня, входит ровно одно ребро. Мы получим 2-упорядоченное дерево, в котором ребро имеет два номера – по одному в каждом из его концов. Затем выполним естественную нумерацию вершин, основанную на обходе дерева в ширину, которую мы применяли в предыдущем подразделе 3.1. После такой нумерации дерево становится правильным. Множество полученных таким образом правильных деревьев обозначим через $\boldsymbol{T}_2(n)$.

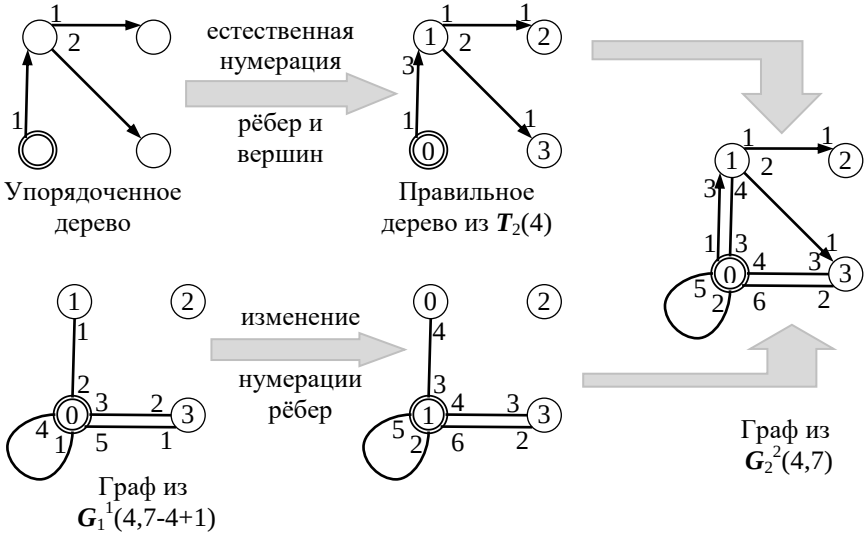


Рис. 6. Построение класса графов $\boldsymbol{G}_2^2(n,m)$ для n=4 и m=7
Fig. 6. The construction of the graph class $\boldsymbol{G}_2^2(n,m)$ for n=4 and m=7

Очевидно, что если для упорядоченного корневого дерева T из правильного дерева $f_2(T)$ удалить нумерацию вершин и добавленные номера рёбер $\Delta(a)$, то получится исходное дерево T. Это означает, что преобразование f_2 является инъекцией. Поэтому $N(\boldsymbol{T}_2(n)) = C_{n\text{-}1}$.

Заметим, что f_2 не сюръективно, например, в дереве $f_2(T)$ корень всегда имеет номер 0, что не обязательно в произвольном правильном дереве. Иными словами, дерево из $\boldsymbol{T}_2(n)$ – это не любое правильное дерево из $\boldsymbol{T}_1(n)$. Дерево из $\boldsymbol{T}_1(n)$ биективно строится из нумерованного корневого дерева, а дерево из $\boldsymbol{T}_2(n)$ биективно строится из упорядоченного дерева; нумерованных корневых деревьев больше, чем упорядоченных деревьев. Поэтому $\boldsymbol{T}_2(n) \subset \boldsymbol{T}_1(n)$.

Теперь рассмотрим операцию «+» добавления к дереву $T \in \boldsymbol{T}_2(n)$ графа $G \in \boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1)$, которую в подразделе 2.3 мы применяли для добавления графа G к правильному дереву из класса $\boldsymbol{T}_1(n)$. Поскольку выше мы доказали, что $T+G$ и $T`+G`$ слабо изоморфны тогда и только тогда, когда T изоморфно $T`$, а $G$ слабо изоморфно $G`$, в графе $T+G$ есть только один правильный остов T. Для дерева $T \in \boldsymbol{T}_2(n)$ его нумерация и, следовательно, нумерация графа $T+G$, однозначно определяется исходным упорядоченным корневым деревом с помощью преобразования f_2. Если удалить нумерацию вершин и добавленные номера рёбер $\Delta(a)$, то из множества графов $\boldsymbol{T}_2(n))+\boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1)$ получится равномощное множество графов, которое обозначим $\boldsymbol{G}_2^{\,2}(n,m)$. Очевидно, $\boldsymbol{G}_2^{\,2}(n,m) \subset \boldsymbol{G}_1^{\,2}(n,m)$. Аналогично тому, как в подразделе 2.3 $N_1(n,m) \geq N(\boldsymbol{G}_1^{\,2}(n,m)) = N(\boldsymbol{T}_1(n))N(\boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1))$, имеем $N_2(n,m) \geq N(\boldsymbol{G}_2^{\,2}(n,m)) = N(\boldsymbol{T}_2(n))N(\boldsymbol{G}_1^{\,1}(n,m\text{-}n\text{+}1)) = C_{n-1}n^{m-n+1} > 2^n n^{m-n+1}$.

Отсюда следует, что для хранения графов из класса $\boldsymbol{G}_2(n,m)$ требуется память $L_2(n,m) = \lfloor lb(N_2(n,m)\text{-}1)\rfloor\text{+}1 \geq \lfloor lb(2^n n^{m-n+1}\text{-}1)\rfloor\text{+}1$. Поскольку $2^n n^{m-n+1}\text{-}1 \geq 2^{n-1}n^{m-n+1}$ при $n \geq 1$ и $m\text{-}n\text{+}1 \geq 0$, а $\lfloor x\rfloor \geq x\text{-}1$, имеем $L_2(n,m) \geq \lfloor lb2^{n-1}n^{m-n+1}\rfloor\text{+}1 \geq (n\text{-}1)\text{+}(m\text{-}n\text{+}1)lbn\text{-}1\text{+}1 = (n\text{-}1)\text{+}(m\text{-}n\text{+}1)lbn$, т.е. $L_2(n,m) = \boldsymbol{\Omega}(n\text{+}(m\text{-}n\text{+}1)logn)$.

4. Графы без кратных рёбер и петель

В графах без кратных рёбер и петель число рёбер ограничено сверху: $m \leq n(n\text{-}1)/2$. Для класса всех таких графов с n вершинами полученные выше оценки для графов с n вершинами и m рёбрами корректируются.

Обозначим: $\boldsymbol{G}_1(n)$ – класс нумерованных и $\boldsymbol{G}_2(n)$ – класс ненумерованных неориентированных упорядоченных корневых связных графов с n вершинами без кратных рёбер и петель, $n\text{^} = n(n\text{-}1)/2$, $N_1(n) = N_1(\boldsymbol{G}_1(n))$, $N_2(n) = N_2(\boldsymbol{G}_2(n))$, $L_1(n) = \lfloor lb(N_1(n))\text{-}1\rfloor\text{+}1$, $L_2(n) = \lfloor lb(N_2(n))\text{-}1\rfloor\text{+}1$.

Функция $l_1(n,m)$ монотонно неубывающая по переменной n, поэтому память размером $l_1(n) = l_1(n,n\text{^})$ достаточна для хранения описания любого графа из $\boldsymbol{G}_1(n)$. Хотя функция $l_2(n,m)$, вообще говоря, не является монотонно неубывающей по n, нетрудно убедиться, что при $m = n\text{^}$ она монотонно неубывающая при $n \geq 2$. Поэтому при $n \geq 2$ память размером $l_2(n) = l_2(n,n\text{^})$ достаточна для хранения описания любого графа из $\boldsymbol{G}_2(n)$.

Для $n \geq 3$ имеем $l_2(n) = l_2(n,n\text{^}) = (n\text{^}\text{-}n\text{+}1)lbn\text{+}n \leq n\text{^}lbn = l_1(n,n\text{^}) = l_1(n)$. Также $l_1(n) = \boldsymbol{O}(n^2 logn)$. Также $N_1(n) \geq N_2(n)$ и, следовательно, $L_1(n) \geq L_2(n)$.

Покажем, что $L_2(n) = \boldsymbol{\Omega}(n^2 logn)$. Для этого рассмотрим класс $\boldsymbol{G}_2^{\,1}(n) \subset \boldsymbol{G}_2(n)$ полных графов (степени всех вершин одинаковы и равны n-1), в каждом из которых есть простой цикл рёбер длины n с нумерацией рёбер по циклу (1,2),(1,2),...(1,2). Для такого графа можно определить естественную нумерацию вершин: вершина, достижимая из корня по последовательности k рёбер с номерами 1, получает номер k. Это значит, что $\forall a\text{=}0..n\text{-}2\ \delta(a,1) = a\text{+}1$, $\delta(n\text{-}1,1) = 0$, $\forall a\text{=}1..n\text{-}1\ \delta(a,2) = a\text{-}1$, $\delta(0,2) = n\text{-}1$.

При отсутствии кратных рёбер слабый изоморфизм графов совпадает с сильным изоморфизмом. Граф из $\boldsymbol{G}_2^1(n)$ однозначно определяется отображением в каждой вершине a номера ребра $i > 2$ в номер конца ребра, т.е. функцией $\delta(a,i)$. Поскольку кратных рёбер нет, эта функция инъективна по номеру ребра i: $\forall a{=}0..n{-}1\ \forall i,j{=}1..n{-}1\ i \neq j \Rightarrow \delta(a,i) \neq \delta(a,j)$. В каждой вершине имеем $(n{-}3)!$ различных упорядочиваний рёбер с номерами больше 2, а всего получаем $N_2(n) = N_2(\boldsymbol{G}_2(n)) \geq N(\boldsymbol{G}_2^1(n)) = ((n{-}3)!)^n$ графов. Тем самым, $L_2(n) = \lfloor lb(N_2(n))\text{-}1 \rfloor + 1 \geq lb(N_2(n))\text{-}1 = lb(((n{-}3)!)^n)\text{-}1 = nlb((n{-}3)!)\text{-}1 \geq nlb((2\pi)^{0.5}(n{-}3)^{n-3+0.5}/e^{n-3})\text{-}1 > n^2lbn$ при $n > 5$. Отсюда $L_2(n) = \boldsymbol{\Omega}(n^2logn)$.

Итак, память, необходимая и достаточная для хранения графа из класса (нумерованных или не нумерованных) упорядоченных корневых связных графов с n вершинами без кратных рёбер и петель, имеет размер $\boldsymbol{\Theta}(n^2logn)$.

5. Память для хранения последовательности O(n) рёбер

В этом разделе мы исследуем вопрос о размере памяти для хранения последовательности рёбер длиной $\boldsymbol{O}(n)$. Мы будем предполагать, что граф нумерованный из класса $\boldsymbol{G}_1(n,m)$ или ненумерованный из класса $\boldsymbol{G}_2(n,m)$. Во втором случае будем применять естественную нумерацию вершин, основанную на обходе графа в ширину и описанную в подразделе 3.1. Длину последовательности рёбер обозначим через h. Будем считать, что $n \geq 3$. Будем предполагать, что для некоторой константы $C > 0$ имеет место $0 < h \leq Cn$.

Последовательность рёбер будем представлять как битовую последовательность $Ребро_1(1)$, $Ребро_1(2)$, ... $Ребро_1(h)$, 1. Здесь $Ребро_1(i)$ – это битовая последовательность $D(a_i),D(b_i\text{-}1),D(c_i\text{-}1),D(d_i)$, представляющая четыре числа: a_i – это номер одной из вершин, которой инцидентно i-ое ребро, b_i – это номер этого ребра в вершине a_i, $c_i = \nu(a_i,b_i)$, $d_i = \delta(a_i,b_i)$. Здесь для числа x используется представление $D(x)$, поскольку число вершин n и максимальная степень вершины заранее неизвестны.

Поскольку $a_i \leq n\text{-}1$ и $n \geq 3$, имеем $D(a_i) \leq 2r(n\text{-}1)+1 = 2(\lfloor lb(n\text{-}1) \rfloor + 1)+1 < 2(lb(n\text{-}1)+1)+1 < 2(lbn+1)+1 = 2lbn+3$. Такую же оценку имеем для $D(d_i)$.

Поскольку $n \geq 3$, в связном графе $1 < \Delta$. Имеем $D(b_i\text{-}1) \leq 2r(\Delta\text{-}1)+1 \leq 2(\lfloor lb(\Delta\text{-}1) \rfloor + 1)+1 < 2(lb(\Delta\text{-}1)+1)+1 < 2(lb\Delta+1)+1 = 2lb\Delta+3$. Такую же оценку имеем для $D(c_i\text{-}1)$.

Тем самым представление последовательности рёбер занимает память размером $q(h) \leq 2h(2lbn+3+2lb\Delta+3)+1 = 4hlbn\Delta+12h+1 \leq 4Cnlbn\Delta + 12Cn + 1$.

При $n \geq 3$ и $\Delta > 1$ имеем $lbn\Delta \geq 1$. Поэтому $q(h) \leq 4Cnlbn\Delta + 12Cnlbn\Delta + nlbn\Delta = (16C+1)nlbn\Delta = \boldsymbol{O}(nlogn\Delta)$.

Так как $\Delta \leq 2m$, имеем $q(h) \leq (16C+1)n(lbnm+1) < (32C+2)nlbnm = \boldsymbol{O}(nlognm)$.

Обозначим верхнюю границу через $Q(n,m) = (32C+2)nlbnm$ и сравним ее с размером памяти, необходимым для описания графа: $L_1(n,m)$ для нумерованного графа и $L_2(n,m)$ для ненумерованного графа.

Поскольку $L_1(n,m) \ge mlbn$ и $L_2(n,m) \ge n+(m-n+1)lbn$, имеем:

$Q(n,m)/L_1(n,m) \le (32C+2)nlb(nm)/(m-1)lbn$ и

$Q(n,m)/L_2(n,m) \le (32C+2)nlb(nm)/((n-1)+(m-n+1)lbn)$.

При фиксированном (но произвольном) $n > 1$ и $m \to \infty$ правые части неравенств стремятся к нулю. Это значит, что размер памяти, необходимый для хранения неориентированного упорядоченного корневого (нумерованного или ненумерованного) графа с n вершинами и достаточно большим по сравнению с n числом m рёбер, больше по порядку, чем размер памяти, достаточный для хранения произвольной последовательности рёбер этого графа длины $\boldsymbol{O}(n)$.

Это, конечно, не означает, что всегда $Q(n,m) \le mlbn$ и $Q(n,m) \le n+(m-n+1)lbn$. При малом по сравнению с n числе рёбер, например, $m = n$ и $n \ge 3$ имеем: $Q(n,n) = (32C+2)nlb(n \cdot n) > (n-1)lbn > (n-1)+(n-n+1)lbn$.

Для более детального анализа можно рассмотреть граф с n вершинами, состоящий только из одного простого цикла длиной n. Сравним размеры $l_1(n,n)$ и $l_2(n,n)$ описаний этого графа с размером $q(n)$ описания последовательности рёбер этого цикла. Имеем:

$l_1(n,n) < 2lbn + 2nlbn + 4n + 2n + 3 = 2(n+1)lbn + 6n + 3$,

$l_1(n,n) = 2\lfloor lb(n-1) \rfloor + 2n\lfloor lb(n-1) \rfloor + 4n + 2n + 3 = 2(n+1)\lfloor lb(n-1) \rfloor + 4n + 2n + 3 >$

$> 2(n+1)(lb(n-1)-1) + 4n + 2n + 3 = 2(n+1)lb(n-1) + 4n + 1$,

$l_2(n,n) < 2(n-n+1)(lbn+2) + 9n - 6 = 2lbn + 9n - 2$.

Для нижней оценки $q(n)$ будем учитывать только часть описания последовательности, а именно номера вершин $D(a_i)$ и $D(d_i)$. Каждая вершина инцидентна двум рёбрам из последовательности всех рёбер.

Поэтому $q(n) > 2\Sigma\{D(i) \mid i=0..n-1\} = 6 + 2\Sigma\{2(\lfloor lbi \rfloor+1)+1 \mid i=1..n-1\} >$

$> 2\Sigma\{2(lbi-1+1)+1 \mid i=1..n-1\} = 4\Sigma\{lbi \mid i=1..n-1\} + n(n-1) = 4lb((n-1)!) + n(n-1) >$

$> 4lb((2\pi)^{1/2}(n-1)^{n-1+1/2}/e^{n-1}) + n(n-1) > 4(n-1)lb((n-1)/e) + n(n-1)$.

Тогда $q(n) / l_1(n,n) > (4(n-1)lb((n-1)/e) + n(n-1)) / (2(n+1)lbn + 6n + 3)$ и

$l_1(n,n) / l_2(n,n) > (2(n+1)lb(n-1) + 4n + 1) / (2lbn + 9n - 2)$.

При $n \to \infty$ правые части неравенств стремятся к бесконечности, т.е. $q(n)$ по порядку больше, чем $l_1(n,n)$, а $l_1(n,n)$ по порядку больше, чем $l_2(n,n)$.

Для графов с n вершинами без кратных рёбер и петель, поскольку $\Delta \le n-1$, имеем $q(h) = \boldsymbol{O}(nlogn\Delta) = \boldsymbol{O}(nlogn^2) = \boldsymbol{O}(nlogn) = \boldsymbol{o}(n^2logn)$, т.е. меньше по порядку, чем размер памяти, необходимый для хранения графа.

6. Память для хранения корневого поддерева

Если нужно хранить не произвольное множество рёбер, а поддерево с корнем в корне графа, то требуется меньше памяти. При естественной ориентации дерева от корня к листьям ребро $a \to b$ задаётся его номером в вершине a.

Для представления такого дерева мы будем использовать естественную нумерацию вершин дерева, основанную на его обходе в ширину. Процедура

нумерации является упрощённой версией процедуры нумерации корневого упорядоченного графа, описанной в подразделе 3.1. Корень получает номер 0. Выбирается первое ребро, инцидентное корню, и его конец получает номер 1. Далее выбирается следующее ребро, инцидентное корню, и его конец получает номер 2. И так далее. Когда исчерпаются все рёбра, инцидентные корню, аналогичная процедура выполняется для вершины 1. В общем, на каждом шаге у нас определены номера вершин 0,...,a` и перебраны все рёбра дерева, инцидентные вершинам 0,...,$a$-1, где $a < a$`, а для вершины a перебраны рёбра с номерами 1,...,i-1. Если $i \le \Delta(a)$, то для вершины a рассматривается ребро с номером i. Конец этого ребра получает номер a`+1. Когда все рёбра, инцидентные вершине с номером $a$, будут перебраны, т.е. когда $i > \Delta(a)$, переходим к следующей нумерованной вершине $a$+1, если $a < a$`. Если же $a = a$`, то процедура нумерации вершин закончена.

Располагаем описания вершин дерева в порядке этой нумерации, а для каждой вершины располагаем номера рёбер дерева, выходящих из этой вершины в естественной ориентации дерева от корня, в порядке возрастания номеров рёбер. Номер x выходящего ребра задаётся в формате $D(x)$. Оно занимает $2r(x)+1 \le 2r(\Delta)+1 = 2(\lfloor lb\Delta \rfloor+1)+1 < 2lb\Delta+3$ бит. Конец списка рёбер дерева, выходящих из данной вершины, задаётся кодом 10 или кодом 11 для последней вершины в списке вершин. Поскольку в дереве число рёбер не больше n-1, число вершин не больше n, а $\Delta \le 2m$, такое представление дерева имеет размер не больше $(n-1)(2lb\Delta+3)+2n = \boldsymbol{O}(nlog\Delta) = \boldsymbol{O}(nlogm) = \boldsymbol{o}(n+(m-n+1)logn)$ при любом (но фиксированном) n и $m \to \infty$.

Для графов с n вершинами без кратных рёбер и петель $\Delta \le n$-1, поэтому представление дерева имеет размер $\boldsymbol{O}(nlog\Delta) = \boldsymbol{O}(nlogn) = \boldsymbol{o}(n^2logn)$, т.е. меньше по порядку, чем размер памяти, необходимый для хранения графа.

7. Заключение

Мы исследовали размер памяти необходимый и достаточный для хранения неориентированного упорядоченного корневого графа. В разделе 2 изучался класс $\boldsymbol{G}_1(n,m)$ нумерованных, а в разделе 3 – класс $\boldsymbol{G}_2(n,m)$ ненумерованных графов с n вершинами и m рёбрами, в разделе 4 – классы $\boldsymbol{G}_1(n)$ нумерованных и $\boldsymbol{G}_2(n)$ ненумерованных графов без кратных рёбер и петель с n вершинами. Соответствующие оценки равны $\boldsymbol{\Theta}(mlogn)$, $\boldsymbol{\Theta}(n+(m-n+1)logn)$ и $\boldsymbol{\Theta}(n^2logn)$.

Эти результаты мы предполагаем, в частности, использовать в последующих статьях, посвященных решению задач на неориентированных графах с помощью коллектива автоматов. Алгоритмы работы таких автоматов существенно зависят от размера памяти каждого автомата, определяемой числом состояний и числом входных и выходных символов. Если память автомата ограничена сверху константой, это конечный автомат (робот). Если память автомата зависит от размера графа, т.е. числа его вершин n и числа его рёбер m, то мы разделяем автоматы на неограниченные автоматы и

полуроботы в зависимости от того, помещается в память автомата описание любого графа из рассматриваемого класса или нет. Тем самым, указанные выше оценки размера памяти определяют границу между неограниченным автоматом и полуроботом на соответствующих классах графов.

В разделе 5 показано, что размер памяти, достаточный для хранения последовательности $\boldsymbol{O}(n)$ рёбер графа, равен $\boldsymbol{O}(nlogn\Delta) = \boldsymbol{O}(nlognm)$ на классах $\boldsymbol{G}_1(n,m)$ и $\boldsymbol{G}_2(n,m)$, и $\boldsymbol{O}(nlogn\Delta) = \boldsymbol{O}(nlogn)$ на классах $\boldsymbol{G}_1(n)$ и $\boldsymbol{G}_2(n)$, что меньше по порядку, чем размер памяти, необходимый для хранения графа соответствующего класса. Тем самым, автомат, с памятью такого размера является полуроботом на соответствующих классах графов. В разделе 6 показано, что для хранения поддерева с корнем в корне графа достаточно ещё меньше памяти: $\boldsymbol{O}(nlog\Delta) = \boldsymbol{O}(nlogm)$ на классах $\boldsymbol{G}_1(n,m)$ и $\boldsymbol{G}_2(n,m)$, и $\boldsymbol{O}(nlog\Delta) = \boldsymbol{O}(nlogn)$ на классах $\boldsymbol{G}_1(n)$ и $\boldsymbol{G}_2(n)$.

Список литературы

[1]. О.Оре. Теория графов. М., Наука, 1968.
[2]. F. Harary and E. M. Palmer. Graphical Enumeration, Academic Press, NY, 1973.
[3]. The On-Line Encyclopedia of Integer Sequences. Catalan numbers. https://oeis.org/A000108. Дата доступа 10.02.2017.
[4]. J. H. Conway and R. K. Guy. The Book of Numbers, New York: Springer-Verlag, 1995.
[5]. F. Bergeron, G. Labelle and P. Leroux. Combinatorial Species and Tree-like Structures, EMA vol.67, Cambridge, 1998.
[6]. R. Dutton, and R. Brigham. Computationally Efficient. Bounds for the Catalan Numbers. Europ. J. Combinatorics, vol.7, 1986.
[7]. Решение математики онлайн, http://math24.biz/. Дата доступа10.02.2017.

Size of the memory for storage of ordered rooted graph

I.B. Burdonov <igor@ispras.ru>
A.S. Kossatchev <kos@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The paper considers boundaries of memory necessary and sufficient for storage of undirected ordered rooted connected graphs, both numbered and unnumbered. The introduction contains the basic definitions and the problem statement. A graph is rooted if one of its vertices is marked as a root. A graph is ordered if for each of its vertices all the incident edges are ordered (numbered). A graph is numbered if all its vertices are numbered with different integer numbers (from 0 to n-1, where n is the number of vertices). Two undirected ordered graphs G and $G`$ are weakly isomorphic if there exists a one-to-one correspondence between their vertices, for which corresponding vertices has the same degrees (numbers of incident edges) and two edges having corresponding ends and the same numbers in these ends, also have the other ends corresponding. Isomorphism of rooted graphs should also correspond their roots. Isomorphism of numbered graphs should also correspond the vertices with the same numbers. Graphs are considered up to weak isomorphism. It is shown that the memory necessary and sufficient for storage of any graph has the size $\Theta(mlogn)$ for numbered graphs, $\Theta(n+(m-n+1)logn)$ for unnumbered graphs with the number of vertices n and the number of edges m, and $\Theta(n^2logn)$ for graphs without multiple edges and loops with the number of vertices n. It is also shown that the memory sufficient for storage of an edge sequence of length $\boldsymbol{O}(n)$ or a spanning tree, has the $\boldsymbol{O}(nlog(n\Delta))$ or $\boldsymbol{O}(nlog\Delta)$ size, respectively, where Δ is the maximum vertex degree.

Keywords: undirected graphs; ordered graph; labeled graph; rooted graph; graph representation; graph enumeration.

DOI: 10.15514/ISPRAS-2017-29(2)-1

For citation: Burdonov I.B., Kossatchev A.S. Size of the memory for storage of ordered rooted graph. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. 7-26. DOI: 10.15514/ISPRAS-2017-29(2)-1

References

[1]. O.Ore. Theory of graphs. AMS, Providence, Rhode Island, 1962.
[2]. F. Harary and E. M. Palmer, Graphical Enumeration, Academic Press, NY, 1973.
[3]. The On-Line Encyclopedia of Integer Sequences. Catalan numbers. https://oeis.org/A000108. Accessed 10.02.2017.
[4]. J. H. Conway and R. K. Guy. The Book of Numbers, New York: Springer-Verlag, 1995.
[5]. F. Bergeron, G. Labelle and P. Leroux. Combinatorial Species and Tree-like Structures, EMA vol.67, Cambridge, 1998.
[6]. R. Dutton, and R. Brigham. Computationally Efficient. Bounds for the Catalan Numbers. Europ. J. Combinatorics, vol.7, 1986.
[7]. The solution of mathematics online, http://math24.biz/. Accessed 10.02.2017.

Общий подход к решению задач на графах коллективом автоматов

И.Б. Бурдонов <igor@ispras.ru>
А.С. Косачев <kos@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Предложен общий подход к решению задач на неориентированном упорядоченном корневом связном графе коллективом автоматов, расположенных в вершинах графа и обменивающихся сообщениями по рёбрам графа. Автоматы считаются полуроботами, т.е. размер их памяти может расти вместе с ростом числа n вершин и числа m рёбер графа, но описание графа может не помещаться в памяти автомата. В разделе 2 классифицируются модели коллектива автоматов на графе в зависимости от размера памяти автомата, времени срабатывания автомата и ёмкости ребра (числа сообщений, одновременно перемещающихся по ребру). Выбрана модель максимального распараллеливания, в которой время срабатывания автомата считается нулевым, а ёмкость ребра неограниченной. Это позволяет получать нижние оценки сложности алгоритмов решения задач. Раздел 3 определяет правила оценки алгоритмов. В разделе 4 описаны базовые процедуры обработки сообщений и проводится классификация используемых сообщений в зависимости от маршрутов, проходимых сообщениями, и способа размножения или слияния сообщений. На основе этих процедур предлагается строить алгоритмы решения задач на графах, что демонстрируется в следующих разделах статьи. В разделах 5-9 предложены алгоритм построения остова графа, алгоритм универсального «стопора», определяющего конец работы любого алгоритма по отсутствию в графе сообщений, используемых этим алгоритмом, алгоритм построения дерева кратчайших путей, алгоритм нумерации вершин графа, и алгоритм сбора полуроботами информации о графе в неограниченной памяти автомата корня. Предложенные алгоритмы имеют линейную сложность от n и m, и используют число сообщений, линейно зависящее от n и m. В разделе 10 рассматривается задача поиска максимального пути в нумерованном графе, которая относится к классу *NP*. За счёт неограниченного (экспоненциального) числа сообщений алгоритм решения этой задачи имеет линейную сложность. В заключении подводятся итоги и намечаются направления дальнейших исследований: решение других задач на графах и расширение подхода на ориентированные графы, а также недетерминированные и динамические графы.

Ключевые слова: неориентированные графы; задачи на графах; асинхронные распределённые системы; распределенные алгоритмы.

DOI: 10.15514/ISPRAS-2017-29(2)-2

Для цитирования: Бурдонов И.Б., Косачев А.С. Общий подход к решению задач на графах коллективом автоматов. Труды ИСП РАН, vol. 29, issue 2, 2017, pp.27-76. DOI: 10.15514/ISPRAS-2017-29(2)-2

1. Введение

Необходимость в решении различных задач на графах возникает во многих приложениях: в информатике и программировании, в коммуникационных и транспортных системах, схемотехнике, экономике, логистике, физике, химии и др. Примером таких задач могут служить задачи поиска максимального пути в графе, построение минимального остовного дерева во взвешенном графе, построение максимального независимого множества вершин, определение всех мостов в неориентированном графе, проверка наличия эйлерового пути или цикла, проверка наличия гамильтонового пути или цикла и построение их, проверка планарности графа и раскраска планарного графа, построение наибольшего ациклического подграфа ориентированного графа и др.

В мировой литературе, как в теоретическом, так и в практическом планах описано много различных алгоритмов решения актуальных задач на графах с оценками времени работы в зависимости от числа вершин, рёбер и других параметров графа. Однако в большинстве своём эти алгоритмы имеют большую вычислительную сложность, поэтому в последние годы растёт интерес к параллельным и распределённым компьютерным алгоритмам. Такие алгоритмы позволяют существенно уменьшить время решения задач и требуют меньше памяти в каждом из используемых компьютеров. Известные и апробированные алгоритмы часто плохо распараллеливаются или возможность такого распараллеливания требует дополнительных исследований. В первом случае необходим поиск новых алгоритмов, а во втором случае – адаптация имеющихся алгоритмов.

Моделью распределённых алгоритмов для решения задач на графах служит коллектив автоматов, которые как-то «привязаны» к графу и взаимодействуют между собой с помощью передачи сообщений. В зависимости от способа «привязки» к графу различают две альтернативные базовые метамодели. В обеих метамоделях автоматы находятся в вершинах графа, а различаются эти метамодели тем, как используются рёбра графа. В первой метамодели автоматы мобильны и могут перемещаться по рёбрам графа из одной вершины в другую, а для взаимодействия автоматов используется локальная память вершин, и/или обмен сообщениями по некоторой дополнительной сети связи, независимой от графа. Во второй метамодели автоматы неподвижно находятся в вершинах, а роль каналов передачи сообщений играют рёбра графа, по которым эти сообщения передаются. Если граф ориентированный, то автоматы (в первой метамодели) или сообщения (во второй метамодели) перемещаются по рёбрам только в направлении их ориентации; в неориентированном графе рёбра могут проходиться в обоих направлениях.

Для того чтобы из вершины по инцидентному ей ребру (выходящему ребру в ориентированном графе) мог перемещаться автомат или сообщение, автомат в вершине должен это ребро указать. Для этого предполагается, что в неориентированном графе все рёбра, инцидентные вершине, а в ориентированном графе все рёбра, выходящие из вершины, перенумерованы. При перемещении по ребру указывается его номер. В ориентированном графе ребро имеет только один номер – в той вершине, из которой оно выходит, а в неориентированном графе ребро имеет два номера – по одному в каждой инцидентной ему вершине. Такой граф с нумерацией рёбер называется *упорядоченным*. В дальнейшем предполагается, что граф упорядочен.

Мы будем считать, что в графе выделена одна вершина, эту вершину будем называть *корнем* графа, а граф – *корневым*. С корня и начинается решение той или иной задачи на графе: автомат, находящийся в корне, получает извне графа (по «фиктивному» ребру, входящему в корень) сообщение, инициирующее решение задачи. После окончания решения задачи автомат корня также вовне (по «фиктивному» ребру, выходящему из корня) посылает сообщение с «ответом». Для того чтобы коллектив автоматов мог решать задачу на всём графе, такой граф, очевидно, должен быть связным.

Ранее мы изучали задачу обхода графа коллективом автоматов, когда должно быть пройдено каждое ребро графа: каким-нибудь автоматом в первой метамодели ([1],[2],[3],[4],[5]) или каким-нибудь сообщением во второй метамодели ([6],[7],[8],[9],[10],[11]). Для второй метамодели также решалась задача вычисления функции от мультимножества значений, записанных в вершинах графа ([7],[8],[10],[11]). Данная статья открывает серию статей, посвящённых решению коллективом автоматов других задач на графах, которые не сводятся к обходу графа, хотя могут использовать алгоритмы обхода. Мы будем исходить из второй метамодели: автоматы неподвижно «сидят» в вершинах графа, а сообщения пересылаются по рёбрам графа.

Распределённая система предполагается асинхронной. Это означает, что время перемещения сообщения по ребру может быть разным как для разных рёбер, так и в разные моменты времени для одного и того же ребра. Будем считать, чтоб время перемещения сообщения по ребру ограничено сверху.

Методы решения задач на графах коллективом автоматов существенно зависят от 1) типа графа, 2) типа автоматов, 3) типа решаемой задачи.

Граф может быть неориентированным, ориентированным и смешанным. В неориентированном графе предполагается использовать возможность посылки ответного сообщения по тому же ребру, по которому было получено прямое сообщение. Это позволяет существенно уменьшить время решения задачи. Этот же способ можно использовать в смешанном графе для его неориентированных рёбер. В ориентированном графе для ответного сообщения приходится искать обратный путь из конца ребра в его начало. Для этого могут использоваться прямой (ориентированный от корня) и обратный (ориентированный к корню) остовы графа ([6],[7],[8],[10],[11]).

Граф может быть детерминированным или недетерминированным. В недетерминированном графе несколько рёбер, выходящих из одной вершины, могут иметь один номер; множество таких рёбер называется *дельта-ребром*. При посылке сообщения выбор ребра из дельта-ребра с данным номером выполняется недетерминированным образом. Для того чтобы гарантировать возможность решения задачи на недетерминированном графе приходится налагать те или иные ограничения на недетерминизм графа (для первой метамодели см. [3],[5]). Примером такого ограничения может служить справедливый недетерминизм, гарантирующий передачу сообщения по данному ребру из данного дельта-ребра после конечного (не обязательно ограниченного) числа попыток.

Другой подход к обходу недетерминированного графа – это изменение самой цели обхода: вместо того, чтобы проходить по всем рёбрам, требуется пройти по всем дельта-рёбрам, то есть хотя бы по одному ребру в каждом дельта-ребре [12]. Это имеет практический смысл при тестировании программной реализации модели системы: в каждом состоянии системы мы пробуем подать на неё каждое тестовое воздействие хотя бы по одному разу. Если граф ориентирован, то для обхода всех его дельта-рёбер требуется дельта-связность графа (в ориентированном графе сильно-дельта-связность [12]). В частности, достаточно наличия в графе детерминированного остова (в ориентированном графе два остова: ориентированный от корня и ориентированный к корню).

Также граф может быть динамически меняющимся: его ребро может исчезать, появляться или менять один из своих концов. Для решения задач на динамическом графе приходится использовать какие-то ограничения на «скорость» изменения рёбер, чтобы гарантировать доставку сообщений в нужные вершины. Например, некоторые рёбра «долгоживущие», т.е. такое ребро не меняется в течение времени, которое достаточно для передачи по нему хотя бы одного сообщения ([9],[10],[11]).

Кроме этого, существенной характеристикой графа является наличие или отсутствие нумерации его вершин. Граф *нумерованный*, если все его вершины помечены различными идентификаторами, в частности, пронумерованы целыми числами 0, 1, 2, ... Нумерация графа даёт возможность различать вершины графа по их номерам, что существенно влияет на алгоритмы решения задач на графе. Во многих случаях для решения задачи полезно сначала выполнить нумерацию графа с помощью подходящего алгоритма, а уже потом решать задачу на нумерованном графе.

Специфической характеристикой асинхронной распределённой системы является *ёмкость ребра* – максимальное число сообщений, одновременно передаваемых по ребру, или их суммарный размер.

Автоматы характеризуются размером памяти и временем срабатывания. По размеру памяти автоматы делятся на *роботы*, *полуроботы* и *неограниченные автоматы* в зависимости от размера их памяти, который определяется числом состояний, входных и выходных символов. Роботы – это конечные автоматы,

память которых ограничена и не зависит от размера графа. Память неограниченного автомата зависит от размера графа и достаточно велика, чтобы в ней могло размещаться описание всего графа. Память полуробота тоже растёт вместе с ростом размера графа, однако описание графа может не помещаться в память одного автомата, хотя обычно предполагается, что это описание помещается в суммарную память коллектива автоматов.

В [13] доказано, что для неориентированных упорядоченных корневых графов «граница» между неограниченным автоматом и полуроботом – это память размером $\Theta(m\log n)$ для нумерованных графов и $\Theta((m-n+1)\log n+n)$ для ненумерованных графов, где m – число рёбер, а n – число вершин графа. Поскольку $(m-n+1)\log n+n < m\log n$ при $n \geq 3$, для того чтобы автомат был полуроботом на классах нумерованных и ненумерованных графов, достаточно, чтобы память автомата имела размер $\boldsymbol{o}((m-n+1)\log n+n)$. В частности, $\boldsymbol{O}(n\log nm) = \boldsymbol{o}((m-n+1)\log n+n)$ для фиксированного (но произвольного) $n \geq 1$ и $m \to \infty$. Заметим, что полуробот на классе графов может быть неограниченным автоматом на подклассе. Тривиальный пример – автомат с памятью $\Theta(n\log nm)$ на подклассе деревьев, где $m = n-1$ и поэтому $(m-n+1)\log n+n = n = \boldsymbol{o}(n\log nm)$.

Для ориентированных графов нужны дополнительные исследования, чтобы определить «границу» между неограниченными автоматами и полуроботами.

Время срабатывания автомата – это время, за которое автомат принимает одно или несколько сообщений, изменяет своё состояние и посылает одно или несколько сообщений. В асинхронных системах обычно не учитывают время срабатывания автомата, измеряя сложность алгоритма временем перемещения сообщений. Важно также, ограничено или нет число сообщений, которые автомат принимает и/или посылает за одно срабатывание.

Понятно, что как сама возможность решения задачи на графе, так и оценка требуемых для этого ресурсов (время, память и число сообщений) зависят от типа автоматов.

Наконец, методы решения задач на графах, естественно, зависят от самих этих задач. Предполагается, что удастся провести классификацию задач в зависимости от того, какие сообщения между автоматами требуются для решения задачи. Для одних задач достаточно, чтобы из корня графа сообщение дошло до каждой вершины, а ответное сообщение вернулось в корень. Для других задач может потребоваться передача сообщений от каждой вершины в каждую вершину и обратно. Могут быть и такие задачи, которые требуют многократной передачи сообщений между вершинами графа. Для некоторых задач нужны дополнительные ограничения на графы или дополнительные возможности для автоматов и системы обмена сообщениями.

В анонсируемой этим введением серии статей мы предполагаем не только предложить алгоритмы решения тех или иных конкретных задач на графах коллективом автоматов, но и провести классификацию и систематизацию

методов решения этих задач с целью выработки общей методологии и базового набора распределённых алгоритмов решения типовых подзадач.

В данной статье мы ограничиваемся случаем неориентированных детерминированных статических графов.

В разделе 2 формально определяется вторая метамодель коллектива автоматов на графе, и обсуждаются различные модели в рамках этой метамодели в зависимости от размера памяти автомата, времени срабатывания и ёмкости рёбер. Выбирается модель с максимальным параллелизмом для полуроботов, которая используется в дальнейшем. Раздел 3 посвящён правилам оценки предлагаемых далее алгоритмов как функции от числа вершин и рёбер графа. В разделе 4 предложены базовые процедуры обработки сообщений, и дана классификация используемых сообщений в зависимости от маршрутов, проходимых сообщениями, и способа размножения или слияния сообщений. Вводимые здесь классы сообщений используются в предлагаемых далее распределённых алгоритмах. В разделе 5 описаны алгоритмы построения остова графа. В разделе 6 предложен алгоритм универсального «стопора», определяющего конец работы любого алгоритма по отсутствию в графе сообщений, используемых этим алгоритмом, а также процедура очистки графа от сообщений указанных типов. В разделе 7 представлен алгоритм построения дерева кратчайших путей. В разделе 8 описаны алгоритмы нумерации графа. В разделе 9 предлагается алгоритм сбора информации о графе в памяти неограниченного автомата корня, автоматы остальных вершин – полуроботы. Раздел 10 посвящён задаче поиска максимального пути в нумерованном графе, относящейся к классу *NP*. В заключении подводятся итоги и намечаются направления дальнейших исследований.

2. Классификация моделей и выбор модели

Как сказано во введении, в данной статье используется вторая метамодель асинхронной распределённой системы: автоматы «привязаны» к вершинам графа, а сообщения посылаются по рёбрам графа. Сообщение понимается как набор *параметров* сообщения, а состояние автомата – как набор значений *переменных*. Допуская вольность речи, мы будем часто для краткости говорить «вершина» вместо «автомат вершины».

Граф неориентированный, статический, упорядоченный, детерминированный, корневой и связный. Обозначения: n – число вершин графа, m – число рёбер графа, D – длина максимального пути в графе, D_0 – длина максимального пути от корня, d – диаметр графа, т.е. максимальное расстояние между вершинами, где расстояние между вершинами – это длина кратчайшего пути между вершинами, d_0 – эксцентриситет корня, т.е. максимальное расстояние от корня до вершины, Δ – максимальная степень вершины.

Очевидно, $n \le m+1$, $d_0 \le d \le 2d_0$, $D_0 \le D \le 2D_0$, $d_0 \le D_0$, $d \le D \le n\text{-}1$ и $\Delta \le 2m$. Различие между n, d и D демонстрируют граф-звезда $S_{n\text{-}1}$, в котором $d = D = 2$

при $n > 2$, и полный граф K_n, в котором $d = 1$ и $D = n\text{-}1$. При $d = 0$ граф состоит из одной вершины с петлями, поэтому $n = 1$ и $D = 0$. При $d = 1$ любые две различные вершины связаны ребром, поэтому $D = n\text{-}1$. Для любых d, D и n, удовлетворяющих условию $2 \le d \le D \le n\text{-}1$, существует граф с параметрами d, D и n (см. Рис. 1). В графе без кратных рёбер и петель $m \le n(n\text{-}1)/2$ и $\Delta \le n\text{-}1$.

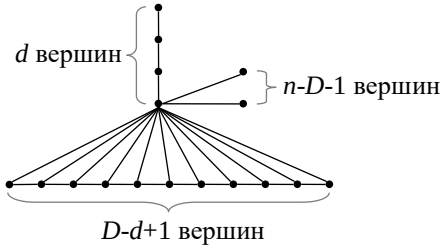


Рис. 1. Граф с n вершинами, диаметром d ≥ 1 и длиной D максимального пути.
Fig. 1. A graph with n vertices, diameter d ≥ 1, and length D of the maximal path.

Ребро графа будет пониматься как дуплексный канал передачи сообщений. При этом сообщения, перемещаемые по ребру в одном направлении, не генерируются самим ребром, не искажаются, не пропадают и не обгоняют друг друга. Иными словами, ребру *ab* соответствуют две очереди сообщений: посланных по этому ребру вершиной *a*, но ещё не принятых с него вершиной *b*, и, наоборот, посланных вершиной *b*, но ещё не принятых вершиной *a*.

Будем исходить из следующих предположений: 1) время перемещения сообщения по ребру ограничено сверху 1 тактом, 2) время срабатывания (автомата) вершины ограничено сверху 1 тактом, 3) за одно срабатывание вершина принимает не более одного сообщения и посылает не более одного сообщения. Третье предположение нуждается в пояснении. Сообщение может быть принято, если оно «дошло» по ребру до вершины, т.е. истекло время перемещения этого сообщения по ребру. Однако, поскольку сообщения, перемещаемые по ребру из вершины *a* в вершину *b*, образуют очередь, вершина *b* может принять только сообщение из головы этой очереди. Если есть несколько рёбер, по которым вершина может принять сообщения, то выбирается одно из них недетерминированным образом. Вместе с принимаемым сообщением вершина получает номер ребра, по которому это сообщение принимается. Если нет сообщений, которые вершина могла бы принять, она получает фиктивное пустое сообщение ε с нулевым номером ребра (рёбра в вершине нумеруются, начиная с 1). Если вершина не посылает сообщение, то указывается пустое сообщение ε с нулевым номером ребра.

Предполагается, что вершина «знает» (или может «узнать») свою степень, которая в дальнейшем хранится в переменной вершины. В некоторых алгоритмах граф предполагается *нумерованным*: его вершинам приписаны уникальные идентификаторы, в частности числа от 0 до n-1. Если граф

нумерованный, то предполагается, что вершина «знает» (или может «узнать») свой номер. Если граф ненумерованный, его можно пронумеровать, используя алгоритм нумерации из раздела 8. В любом случае будем предполагать, что для нумерованного графа номер вершины хранится в переменной вершины.

Ёмкостью ребра будем называть максимальное число сообщений, которые могут одновременно находиться на ребре (посланы, но не приняты) в одном направлении. Если ребро *ab* имеет ограниченную ёмкость *k* и в направлении от вершины *a* к вершине *b* по нему перемещается *k* сообщений, то вершина *a* не может посылать сообщение по ребру *ab* до тех пор, пока вершина *b* не примет с ребра хотя бы одно сообщение. Для того чтобы вершина смогла «узнать», можно или нельзя посылать по ребру сообщение, при ограниченной ёмкости ребра нужен сигнал «*освобождение ребра*» ϕ с параметром «номер ребра». Если ёмкость ребра неограниченная, сообщение по ребру всегда можно посылать, и такого сигнала не требуется.

Мы исходим из того, что вершина, принимая сообщение, размещает его в своей памяти, а для посылки сообщения вершина сначала формирует его в своей памяти. Тем самым, нужно учитывать не только суммарный размер переменных вершины, который мы будем называть *размером автомата* вершины, но и его сумму с максимальным размером сообщения, которую мы будем называть *полным размером автомата*.

Как указано во введении, в зависимости от их полного размера автоматы делятся на неограниченные автоматы, роботы и полуроботы. Рассмотрим различные модели в рамках данной метамодели в зависимости от полного размера автомата, времени срабатывания и ёмкости ребра (см. табл. 1).

Если время срабатывания больше нуля, то это может приводить к тому, что на ребрах, инцидентных вершине, скапливаются сообщения, которые уже завершили перемещение по этим рёбрам, но вершина не успевает их принять. Поэтому нужно учитывать такие задержки, и оценка времени работы становится достаточно сложной.

Если ёмкость ребра ограничена числом *k*, то это может приводить к тому, что вершина, инцидентная ребру, не может посылать сообщения по нему, если ребро полностью занято (на нём уже находится *k* сообщений), пока ребро не освободится. Поэтому при оценке времени работы нужно учитывать такие задержки в пересылке сообщений, и оценка становится достаточно сложной (в качестве примера, можно посмотреть оценку времени работы алгоритма в [8]).

Табл. 1. Классификация моделей
Tab. 1. Classification of models

Модель	Полный размер автомата	Время срабатывания	Ёмкость ребра
1	неограниченный автомат	> 0	не ограничена
2	неограниченный автомат	= 0	ограничена
3	неограниченный автомат	> 0	ограничена
4	неограниченный автомат	= 0	не ограничена

5	робот	> 0	не ограничена
6	робот	= 0	ограничена
7	робот	> 0	ограничена
8	робот	= 0	не ограничена
9	полуробот	> 0	не ограничена
10	полуробот	= 0	ограничена
11	полуробот	> 0	ограничена
12	полуробот	= 0	не ограничена

Формально, автомат в вершине – это набор (Mes,S,Tr,s_0), где Mes – множество всех непустых сообщений, которые автомат может принимать или посылать, S – множество состояний, Tr – множество переходов, s_0 – начальное состояние. Переход имеет вид s—(i,x),(j,y)→s`, $s$—($i$,$x$),(0,ε)→$s$`, s—(i,ϕ),(j,y)→s`, $s$—($i$,ϕ),(0,ε)→$s$`, s—(0,ε),(j,y)→s`, или $s$—(0,ε),(0,ε)→$s$`, где s∈S – состояние, в котором выполняется переход, ε – пустое сообщение, ϕ – сигнал «освобождение ребра», i – номер ребра, по которому принимается непустое сообщение или сигнал ϕ, x∈Mes – принимаемое сообщение, j – номер ребра, по которому посылается непустое сообщение, y∈Mes – посылаемое сообщение, s`∈S – состояние, в которое автомат переходит.

Если полный размер автомата ограничен (робот или полуробот) и ёмкость ребра тоже ограниченная, то возможны состояния, где вершина не принимает сообщений, поскольку не может послать сообщения. Вершина ожидает только сигналов освобождения рёбер, т.е. в таких состояниях определены только переходы вида s—(i,ϕ),(i,y)→s` или $s$—($i$,ϕ),(0,ε)→$s$`. В остальных состояниях вершина v принимает любое сообщение, т.е. её автомат «всюду определён по приёму сообщений»: в каждом таком состоянии определены переходы по всем возможным парам (i,x), (i,ϕ), (0,ε), где i=1..Δ_v, x∈Mes и Δ_v – степень вершины v.

2.1. Неограниченный автомат

Случай неограниченного автомата (модели 1-4) не очень интересен: решение любой задачи можно свести к следующей процедуре. Сначала информация о графе собирается в корне графа, а затем автомат корня решает задачу. В разделе 9 мы опишем алгоритм выполнения процедуры сбора информации.

2.2. Робот

В литературе имеется ряд работ по обходу графа роботом. Для неориентированного графа известен алгоритм Тэрри, основанный на обходе в глубину (DFS), с оценкой времени работы 2m, и алгоритм, основанный на обходе в ширину (BFS), с оценкой m+$\boldsymbol{O}$(n^2) ([14]). Для ориентированного графа наилучшая полученная оценка равна $\boldsymbol{O}$(nm+$n^2 loglogn$) или, выражая через D_0, $\boldsymbol{O}$($D_0 m$+$D_0 nloglogn$) ([14],[15]). При повторном обходе графа, когда автоматы могут использовать результаты первого обхода, оценка

$\boldsymbol{O}(nm+n^2log^*n)$ или $\boldsymbol{O}(D_0m+D_0nlog^*n)$, где log^* – итерированный логарифм (число логарифмирований, после которых результат меньше или равен 1) [16]. Если имеются два робота, передвигающихся по графу и обменивающихся между собой сообщениями по дополнительной сети связи (первая метамодель), то время обхода уменьшается до минимального по порядку $\boldsymbol{O}(nm)$ или, выражая через D_0, $\boldsymbol{O}(D_0m)$ ([14]).

Исследование графов роботами имеет целый ряд специфических особенностей, вызываемых ограниченной памятью робота. В настоящей статье мы не исследуем решение задач коллективом роботов.

2.3. Полуробот

Как и в общем случае, для полироботов ненулевое время срабатывания и/или ограниченная ёмкость ребра приводят к задержкам в пересылке сообщений, что усложняет оценку времени работы алгоритмов. Но в случае полироботов (как и роботов) ограниченная ёмкость ребра может также приводить к *тупику* (deadlock). Дело в том, что, если ребро полностью заполнено, и вершина не может по нему послать сообщение, то, поскольку полный размер автомата ограничен (хотя и зависит от размера графа для полуробота), может оказаться, что вершина приостановит приём сообщений. На цикле в графе может возникнуть тупик, когда каждая вершина цикла «хочет» послать сообщение по «выходящему» ребру цикла, но не может этого сделать, и поэтому не принимает сообщение по «входящему» ребру цикла. Алгоритмы решения задач на графе должны учитывать возможность тупика и не допускать тупика, что, конечно, налагает дополнительные требования к построению алгоритмов.

2.4. Выбор модели

Для серии статей, открываемой данной статьёй, мы выбираем модель 12: ёмкость ребра неограниченная, автомат – полуробот с нулевым временем срабатывания. Время перемещения сообщения по ребру ограничено сверху 1 тактом. Такая модель даёт возможность максимального распараллеливания, поскольку не возникает задержек и тупиков из-за ограниченной ёмкости ребра или ненулевого времени срабатывания. Тем самым, будет учитываться только время перемещения сообщений по рёбрам. Оценки сложности алгоритмов решения задач в такой модели являются нижними границами сложности аналогичных алгоритмов в других моделях с ограниченными автоматами (модели 5-11). Алгоритм в модели 12 либо будет работать медленнее в моделях 5-11, либо не будет работать из-за тупиков или размера памяти в моделях 5-8 для роботов. Однако любой алгоритм в моделях 5-11 будет работать и не большее время в модели 12.

Отметим, что при нулевом времени срабатывания не имеет смысла принимать или посылать больше одного сообщения за одно срабатывание, поскольку это

не уменьшает времени работы, но приводит к увеличению полного размера автомата за счёт хранения в его памяти сразу нескольких сообщений.

Наша модель отличается от хорошо известной модели распределённых вычислений LOCAL [17] асинхронностью и наличием корня. В модели LOCAL все автоматы в вершинах работают синхронно и начинают работать одновременно, поэтому сложность алгоритма в этой модели – это число срабатываний. В нашей модели работа начинается с корня, другая вершина включается в работу при получении ею первого сообщения. Кроме того, при оценке сложности алгоритма мы считаем концом работы алгоритма не момент времени, когда задача решена, а момент времени, когда об этом «узнал» корень и сообщил вовне. Асинхронность моделируется не разными временами срабатываний автоматов, а разными и, вообще говоря, переменными временами перемещения сообщений по рёбрам, хотя и ограниченными сверху 1 тактом. Это приводит к тому, что по ребру может передаваться не одно сообщение, а последовательность сообщений, посланных с одного конца ребра и ещё не принятых другим концом.

Для нашей модели, очевидно, получается нижняя оценка $2d_0$+1 сложности любого алгоритма, не имеющего предварительных знаний о графе (вершина знает только свою степень и, для нумерованного графа, свой номер) и решающего глобальную задачу на графе. Задача глобальна, если она не может быть решена без исследования каждого ребра графа. Такое исследование требует, очевидно, времени d_0+1 в наихудшем случае, когда по каждому ребру сообщение двигается ровно 1 такт. Для того чтобы сообщить корню о завершении работы, требуется ещё не менее d_0 тактов в наихудшем случае.

Следует, однако, учитывать, что неограниченная ёмкость ребра и нулевое время срабатывания в некоторых алгоритмах может приводить к неограниченному росту числа сообщений, одновременно циркулирующих в графе и, более того, одновременно перемещающихся по одному ребру. Иными словами, решение задачи на графе может использовать неограниченные ресурсы памяти за счёт сообщений. В этом смысле наша модель сближается с недетерминированной машиной Тьюринга, которая неограниченно (экспоненциально) копируется по дереву вычислений. Поэтому оценки сложности предлагаемых нами алгоритмов решения некоторых задач могут оказаться полиномиальными, хотя эти задачи относятся к классу *NP*. Примером может служить задача о поиске максимального пути в графе (*Longest Path Problem* [18]), которой посвящен раздел 10 данной статьи.

3. Оценка алгоритмов

Обозначения в оценках алгоритмов: M – максимальный размер сообщения, A – размер автомата (сумма размеров переменных), $F = M+A$ – полный размер автомата, T – время работы алгоритма, N – максимальное число сообщений, одновременно передаваемых по одному ребру в одном направлении, и E – максимальный суммарный размер таких сообщений. Очевидно, $E \le NM$.

Для оценки размера памяти от двух переменных n и m будем считать, что n любое, но фиксированное, достаточно большое число, а $m \to \infty$. Будем писать:

$f(n,m) = \boldsymbol{o}_{m\to\infty}(g(n,m))$, если $\exists n_0\ \forall n \ge n_0\ \forall C > 0\ \exists m_0\ \forall m \ge m_0\ f(n,m) \le Cg(n,m)$, что в нашем случае эквивалентно $\exists n_0\ \forall n \ge n_0\ lim_{m\to\infty}(f(n,m)/g(n,m)) = 0$.

Тем не менее, там, где это возможно, будем применять оценки памяти, когда независимо $n \to \infty$ и $m \to \infty$ при условии связности графа $m \ge n$-1. Будем писать:

$f(n,m) = \boldsymbol{o}_{n,m\to\infty}(g(n,m))$, если $\forall C > 0\ \exists n_0\ \forall n \ge n_0\ \forall m \ge n\text{-}1\ f(n,m) \le Cg(n,m)$.

Согласно [13], если $F = \boldsymbol{o}_{m\to\infty}(mlogn)$, то автомат полуробот на классе нумерованных графов, а если $F = \boldsymbol{o}_{m\to\infty}(n+(m\text{-}n+1)logn)$, то автомат полуробот на классе ненумерованных графов. Класс ненумерованных графов можно рассматривать как подкласс класса нумерованных графов, а полуробот на подклассе является полуроботом на классе. Далее для краткости будем обозначать: $\boldsymbol{o}_{n,m\to\infty} = \boldsymbol{o}_{n,m\to\infty}(n+(m\text{-}n+1)logn)$, $\boldsymbol{o}_{m\to\infty} = \boldsymbol{o}_{m\to\infty}(n+(m\text{-}n+1)logn)$.

Конечно, на подклассе графов, где $m = \boldsymbol{O}(n)$, в частности, для деревьев $m = n$-1, уже не верно, что $\boldsymbol{O}(nlognm) = \boldsymbol{o}_{m\to\infty}$, поскольку m не может бесконечно расти (для деревьев m не меняется) при фиксированном n. На таких подклассах могут существовать алгоритмы с меньшей памятью автоматов. Однако нас интересуют алгоритмы, работающие на классе всех графов, и они могут иметь «лишнюю» память на подклассах.

Для корректного использования некоторых наших оценок достаточно, чтобы на рассматриваемом классе графов число рёбер m могло расти быстрее, чем число вершин n, т.е. на некоторой последовательности графов $m/n \to \infty$. Обозначая $h(n) = m/n$, имеем $m = nh(n)$. Если $lim_{n\to\infty}h(n) = \infty$ влечёт $lim_{n\to\infty}(f(n,m)/g(n,m)) = 0$, то будем писать $f(n,m) = \boldsymbol{o}_{m/n\to\infty}(g(n,m))$. Пример: если $m = nlogn$, то $m = \boldsymbol{o}_{m/n\to\infty}(n^2)$. Обозначим $\boldsymbol{o}_{m/n\to\infty} = \boldsymbol{o}_{m/n\to\infty}(n+(m\text{-}n+1)logn)$. Автоматы с памятью $\boldsymbol{o}_{m/n\to\infty}$ являются полуроботами на классе ненумерованных графов.

На подклассе графов без кратных рёбер и петель с n вершинами оценки зависят только от n, а m ограничено сверху по порядку n^2. Согласно [13] на этом подклассе автомат полуробот, если $F = \boldsymbol{o}(n^2logn)$. Для краткости обозначим $\boldsymbol{o}_{n\to\infty} = \boldsymbol{o}(n^2logn)$. Если F имеет эту оценку на подклассе графов без кратных рёбер и петель, будем указывать её в квадратных скобках $[\boldsymbol{o}_{n\to\infty}]$.

Докажем следующие основные отношения:

1) $logm = \boldsymbol{o}_{n,m\to\infty}$, 2) $nlognm = \boldsymbol{o}_{m\to\infty}$ и $n^2logm = \boldsymbol{o}_{m\to\infty}$,

3) $m = \boldsymbol{o}_{m/n\to\infty}$, 4) $nlogn = \boldsymbol{o}_{n\to\infty}$ и $n^2 = \boldsymbol{o}_{n\to\infty}$.

1) Пусть $C > 0$. Тогда для $n \ge 1/C$+3 имеем $n \ge e$, что для $k \ge 0$ влечёт $n^k \ge e^k = (e^{kC})^{1/C}$ и, поскольку $e^x \ge 1+x$, имеем $n^k \ge (e^{kC})^{1/C} \ge (1+\kappa C)^{1/C} \ge (1+k/(n\text{-}3))^{1/C} \ge (1+k/(n\text{-}1))^{1/C} = ((n\text{-}1+k)/(n\text{-}1))^{1/C}$. Отсюда $klogn \ge (1/C)log(n\text{-}1+k)\text{-}(1/C)log(n\text{-}1)$, что влечёт $n+klogn\text{-}(1/C)log(n\text{-}1+k) \ge n\text{-}(1/C)log(n\text{-}1)$. Поскольку в связном графе $m\text{-}n+1 \ge 0$, выберем $k = m\text{-}n+1$. Поэтому $n+(m\text{-}n+1)logn\text{-}(1/C)logm \ge$

$\geq n\text{-}(1/C)log(n\text{-}1)$. Поскольку $lim_{n\to\infty}(n/((1/C)log(n\text{-}1))) = \infty$, найдётся такое $n_0 \geq 1/C+3$, что для каждого $n \geq n_0$ будет $n\text{-}(1/C)log(n\text{-}1) \geq 0$. Тем самым, для $n \geq n_0$ будет $n+(m\text{-}n+1)logn\text{-}(1/C)logm \geq 0$, что влечёт $logm \leq C(n+(m\text{-}n+1)logn)$. Тем самым, $logm = \boldsymbol{o}_{n,m\to\infty}$.

2) $nlognm = \boldsymbol{o}_{m\to\infty}$ доказано в [13]. При любом, но фиксированном $n \geq 2$, имеем $lim_{m\to\infty}((n^2logm)/(n+(m\text{-}n+1)logn)) = 0$. Следовательно, $n^2logm = \boldsymbol{o}_{m\to\infty}$.

3) Пусть $m = nh(n)$ и $lim_{n\to\infty}h(n) = \infty$. Тогда $lim_{n\to\infty}(m/(n+(m\text{-}n+1)logn)) =$
$= lim_{n\to\infty}(m/(n+mlogn\text{-}nlogn+logn)) = lim_{n\to\infty}(h(n)/(1+h(n)logn\text{-}logn+(logn)/n)) =$
$= lim_{n\to\infty}(h(n)/(1+h(n)logn\text{-}logn+0)) = lim_{n\to\infty}(h(n)/(1+h(n)logn\text{-}logn)) =$
$= lim_{n\to\infty}((1/logn)/(1/(h(n)logn)+1\text{-}1/h(n))) = 0/(0+1\text{-}0) = 0$. Поэтому $m = \boldsymbol{o}_{m/n\to\infty}$.
4) $lim_{n\to\infty}((nlogn)/(n^2logn)) = lim_{n\to\infty}(1/n) = 0$. Поэтому $nlogn = \boldsymbol{o}_{n\to\infty}$.
$lim_{n\to\infty}((n^2)/(n^2logn)) = lim_{n\to\infty}(1/logn) = 0$. Поэтому $n^2 = \boldsymbol{o}_{n\to\infty}$.

В дальнейшем при оценках мы будем также учитывать, что $d_0 \leq d \leq D \leq n\text{-}1$, $d_0 \leq D_0 \leq D$, $\Delta \leq 2m$, а для графа без кратных рёбер и петель $\Delta \leq n\text{-}1$.

Когда алгоритм $\mathcal{A}$ использует результат работы другого алгоритма $\mathcal{B}$, то иногда даётся оценка только алгоритма $\mathcal{A}$ без учёта ресурсов, используемых алгоритмом $\mathcal{B}$. Если нужна суммарная оценка, то величины M, A, F, T и E суммируются, а для величины N берётся максимум.

4. Классификация сообщений

В предлагаемых алгоритмах будут использоваться некоторые базовые процедуры обработки сообщений в вершинах. Каждой процедуре соответствует класс сообщения, где под классом сообщения понимается не функциональное предназначение сообщения, а способ его передачи по графу: маршрут, который проходит сообщение, и способ «размножения» или «слияния» сообщений. Те параметры сообщения и переменные вершины, которые существенны для обработки сообщений данного класса, будем называть, соответственно, базовыми параметрами и базовыми переменными. Временем работы алгоритма базовой процедуры будет считаться время распространения сообщения от его создания до уничтожения.

Для любого класса базовым параметром является тип сообщения. Сообщение данного типа, попадая в вершину, далее либо 1) уничтожается, после чего, возможно, из вершины посылается одно или несколько сообщений уже другого типа, либо 2) пересылается вершиной дальше по одному или нескольким ребрам, быть может, с модификацией содержимого сообщения, но не его типа. Если принимается одно сообщение, а потом посылаются сообщения того же типа по нескольким ребрам, то это будем называть *размножением* сообщения. Если принимается несколько сообщений одного типа, быть может за несколько срабатываний, а потом посылается сообщение того же типа по одному ребру, то это будем называть *слиянием* сообщений.

Сообщение данного типа однозначно определяется маршрутом, который оно прошло: для любого маршрута одновременно существует не более одного сообщения этого типа, которое прошло этот маршрут. При размножении получаются сообщения одного типа, маршруты которых имеют одинаковый префикс, заканчивающийся в вершине, где произошло размножение. При слиянии нескольких сообщений одного типа маршрут каждого сообщения, кроме одного, заканчивается в вершине, где произошло слияние.

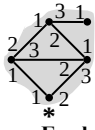
Таким образом, класс сообщения определяется способом размножения или слияния сообщений и маршрутом, которое сообщение проходит до уничтожения; такой маршрут будем называть маршрутом класса. Мы определим девять базовых классов сообщений, иллюстрируемых Рис. 2.

Формально *маршрутом* называется чередующаяся последовательность вершин и ребер, начинающаяся и заканчивающаяся вершиной, в которой каждое i-ое ребро инцидентно предыдущей i-ой и последующей i+1-ой вершинам. Номер i-ой вершины маршрута будем обозначать v_i, номер i-ого ребра в i-ой вершине будем называть *прямым номером* и обозначать f_i, а его номер в i+1-ой вершине будем называть *обратным номером* и обозначать r_i.

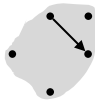
Сообщение может накапливать в себе пройденный им маршрут в виде последовательности $P = \langle v_1,f_1,r_1,v_2,f_2,r_2,\ldots v_k,f_k,r_k,v_{k+1},f_{k+1}\rangle$ номеров пройденных вершин и прямых и обратных номеров рёбер, а сообщение передаётся из вершины v_{k+1} по ребру f_{k+1}. Когда такое сообщение создаётся вершиной v_1 и посылается по ребру f_1, в сообщении параметр $P = \langle v_1,f_1\rangle$. Когда вершина v_i по ребру r_{i-1} получает сообщение с параметром $P = \langle v_1,f_1,r_1,v_2,f_2,r_2,\ldots v_{i-1},f_{i-1}\rangle$ и посылает сообщение далее по ребру f_i, в посылаемом сообщении параметр $P := P\cdot\langle r_{i-1},v_i,f_i\rangle$. Если какие-то номера не нужно накапливать: номера вершин, прямые номера рёбер или обратные номера рёбер, – то в конец P не добавляется соответствующий номер, и последовательность P не содержит эти номера. Будем называть вариант накопления *x-накоплением*, а накопленную последовательность P – *x-вектором маршрута*, где x определяется регулярным выражением $x = v\,?\,f\,?\,r\,?$ (знак «?» означает число повторений предыдущего символа 0 или 1 раз). По умолчанию накопления нет (x пусто).

В данной статье *хордой подграфа* будем называть ребро, оба конца которого принадлежат подграфу, но само ребро не принадлежит подграфу. *Путём* называется маршрут без самопересечения, т.е. такой маршрут длины k, что $\forall i,j=1..k+1\ \ (i\neq j \Rightarrow v_i\neq v_j)$. Иногда такой маршрут называют простым путём. *Терминальной вершиной* будем называть вершину степени 1. *Путём с хордой* назовем маршрут, который состоит из пути, продолженного хордой этого пути, т.е. имеет вид $P\cdot\langle e,v\rangle$, где P – путь, e – хорда пути P, инцидентная конечной вершине пути и вершине v. *Путём с ребром* назовем маршрут, являющийся путём или путём с хордой. Очевидно, длина пути не превосходит D, а если начало маршрута – корень, то D_0. Максимальная длина пути с хордой на 1 больше. Для корневого дерева *листом* или *листовой вершиной* дерева называют терминальную вершину дерева, отличную от корня, а

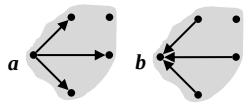
внутренней вершиной дерева – его вершину, не являющуюся листом или корнем дерева. Корнем остова графа, будем считать корень графа.



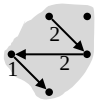
Граф
(корень обозначен *)



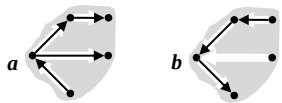
1. Одиночная передача



2. Вопрос / Ответ
a) ***Вопрос***, ***b***) ***Ответ***.



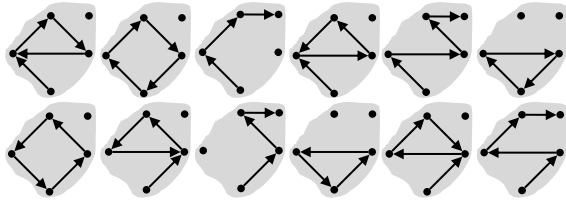
3. Рассылка по запомненному маршруту.
f-вектор <2,2,1>.



4. Рассылка по отмеченным (белым) рёбрам:
a) по прямому остову,
b) по обратному пути.

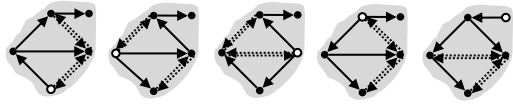
5. Рассылка против отмеченных (белых) рёбер:
a) против прямого пути,
b) против обратного остова.
Пунктиром показаны пересылки с уничтожением сообщения.

6. Сбор по обратному дереву (его рёбра белые)**.**
ЧислоРёбер = число входящих обратных рёбер.



7. Рассылка до самопересечения.
Показаны все маршруты из корня.

8. Рассылка без повторения.
Маршруты из корня.
Пунктиром показаны сообщения, повторно приходящие в вершину.



9. Множественная рассылка.
Показаны рассылки из всех инициаторов (инициаторы выделены белым центром).
Пунктиром показаны сообщения, повторно приходящие в вершину.

Рис. 2. Базовые классы сообщений
Fig. 2. Basic message classes

Размеченным графом назовём такой граф, в некоторых вершинах которого отмечены некоторые инцидентные им рёбра со следующими ограничениями: 1) каждое ребро отмечено не более чем в одной из инцидентных ему вершин,

и отмеченное ребро считается условно ориентированным от этой вершины; 2) условно ориентированные отмеченные ребра образуют ациклический граф. Очевидно, длина пути по отмеченным рёбрам не превосходит D, а если один из концов пути корень, то D_0. Будем говорить, что отмеченное ребро *выходит из вершины*, если оно отмечено в этой вершине, и *входит в вершину*, если оно инцидентно этой вершине, но не отмечено в ней.

Нам понадобятся два важных частных случая размеченного графа, когда отмеченное множество рёбер порождает лес деревьев, и в каждом дереве выделен его корень. Если все рёбра леса отмечены так, что каждое дерево условно ориентировано от корня (к корню) дерева, то такой размеченный граф будем называть *прямым* (*обратным*) *лесом*. Если лес состоит из одного дерева, являющегося остовом графа, то прямой (обратный) лес будем называть *прямым* (*обратным*) *остовом*. Рёбра прямого (обратного) леса будем называть *прямыми* (*обратными*) *рёбрами*, а путь из прямых (обратных) рёбер будем называть *прямым* (*обратным*) *путём*.

При описании классов сообщений *специальный критерий* понимается как некоторое условие, основанное на значениях переменных вершины.

В дальнейшем мы будем описывать алгоритмы, составленные как из «кирпичиков» из нескольких алгоритмов, в том числе, из базовых процедур, быть может, с какими-то их модификациями. Там, где будет возникать коллизия совпадающих имён переменных или типов сообщений из разных комбинируемых алгоритмов или при разных модификациях одного и того же алгоритма, будем предполагать систематическое переименование этих имён и типов, не оговаривая этого специально.

4.1. Одиночная передача

Сообщение этого класса двигается по одному ребру, после чего уничтожается.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$.

Вершина может создать сразу несколько сообщений этого типа и одновременно послать их по нескольким инцидентным рёбрам: по всем или по тем, которые удовлетворяют тому или иному специальному критерию. Также сообщение этого типа может одновременно создаваться в нескольких вершинах. В любом случае после создания всех таких сообщений по каждому ребру в каждом направлении проходит не более одного сообщения.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

4.2. Вопрос/Ответ

В этой процедуре используются два типа сообщений: ***Вопрос*** и ***Ответ***.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *СчётчикОтветов* – изменяется от *Степень* до 0, имеет размер $\boldsymbol{O}(log\Delta)$.

Вершина посылает ***Вопрос*** по каждому инцидентному ей ребру, после чего ожидает получения сообщений ***Ответ*** по каждому такому ребру. Предполагается, что по каждому ребру должен придти ровно один ***Ответ***. Сначала *СчётчикОтветов* = *Степень*, а потом счётчик уменьшается на 1 при получении сообщения ***Ответ***. Когда счётчик становится равным 0, он снова инициализируется *СчётчикРёбер* := *Степень* для дальнейшего использования. Получив ***Вопрос*** по некоторому ребру, вершина посылает по этому же ребру в обратном направлении ***Ответ***. По каждому ребру в каждом направлении проходит не более одного ***Вопроса*** и не более одного ***Ответа***; причём ***Ответ*** посылается по ребру только после получения по этому ребру ***Вопроса***.

Возможны модификации этого алгоритма, когда ***Вопрос*** посылается только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию, и вместо *Степень* используется число таких рёбер.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2$. $N \le 2$. $E = \boldsymbol{O}(1)$.

4.3. Пересылка по запомненному маршруту

Сообщение содержит в себе маршрут, по которому оно должно пройти, в виде *f*-вектора маршрута. Сообщение не размножается, а уничтожается тогда, когда проходит запомненный в нём маршрут.

Базовые параметры: 1) *Тип* – тип сообщения размером $\boldsymbol{O}(1)$, 2) *Маршрут* – *f*-вектор $P = \langle f_1, f_2, ... f_k \rangle$ ещё не пройденной части маршрута размером $\boldsymbol{O}(k log\Delta)$. Базовых переменных нет.

Пока *f*-вектор имеет вид $P = \langle f_1, f_2, ... f_k \rangle$, где $k > 0$, из сообщения удаляется номер ребра f_1, т.е. $P := \langle f_2, ... f_k \rangle$, и сообщение посылается по ребру f_1. Если $k = 0$, то сообщение уничтожается.

Оценка. Тогда $M = \boldsymbol{O}(k log\Delta)$. $A = \boldsymbol{O}(1)$. $F = \boldsymbol{O}(k log\Delta)$. Для $k = \boldsymbol{O}(D)$ имеем $F = \boldsymbol{O}(D log\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le k$. Сообщение не размножается, поэтому $N \le 1$. $E = \boldsymbol{O}(k log\Delta)$.

4.4. Рассылка по отмеченным рёбрам

Граф размеченный. Маршрут этого класса является путём, состоящим из отмеченных рёбер. Ребро проходится по его условной ориентации: из вершины, в которой ребро отмечено, в вершину, в которой оно не отмечено.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовых переменных нет, не считая отметок рёбер.

Рассылка сообщений начинается с одной или нескольких вершин-инициаторов. Сообщение двигается по отмеченным рёбрам. Обычно при размножении создается столько сообщений, сколько из вершины выходит отмеченных рёбер и сообщения посылаются по этим рёбрам. Возможно также, что сообщение посылается не по всем таким рёбрам, а только по тем, которые удовлетворяют тому или иному специальному критерию. Обычно сообщение

уничтожается в стоке ациклического подграфа отмеченных рёбер, т.е. в такой вершине, из которой не выходят отмеченные рёбра. Возможно также, что сообщение уничтожается раньше по тому или иному специальному критерию.

Нам понадобятся два важных частных случаях: рассылка по прямому лесу и пересылка по обратному пути. Если подграф отмеченных рёбер – это прямой лес, то рассылка обычно начинается в корне дерева этого леса; в каждой внутренней вершине степени больше 2 происходит размножение сообщения. Число сообщений равно числу листьев деревьев прямого леса. Если же подграф отмеченных рёбер – это обратный лес, то размножения не происходит, поскольку из каждой вершины выходит не более одного обратного ребра. Сообщение двигается по обратным рёбрам до корня дерева.

Оценка. $M = \boldsymbol{O}(1)$. Не считая отметок рёбер, $A = \boldsymbol{O}(1)$ и $F = \boldsymbol{O}(1) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le D$ ($T \le D_0$ при рассылке от корня или если корень единственный сток подграфа отмеченных рёбер). Для прямого леса, когда инициаторы – это корни деревьев, $N \le 1$ и $E = \boldsymbol{O}(1)$, для обратного леса, когда инициаторы – это листья деревьев, $N \le n$-1 и $E = \boldsymbol{O}(n)$.

4.5. Рассылка против отмеченных рёбер

Граф размеченный. Маршрут этого класса состоит из отмеченных рёбер, но проходимых против их условной ориентации: из вершины, в которой ребро не отмечено, в вершину, в которой оно отмечено. Точнее, сообщение пересылается и по рёбрам, не отмеченным в их другом конце, но после такой пересылки сообщение сразу уничтожается. Не считая таких пересылок, сообщение двигается по ациклическому графу отмеченных рёбер в направлении, обратном условной ориентации этих рёбер.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные (не считая отметок рёбер): *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$.

Рассылка сообщений начинается с одной или нескольких вершин-инициаторов. Инициатор рассылает сообщение по всем инцидентным ему рёбрам, не отмеченным в нём. Вершина, получив сообщение по некоторому ребру, проверяет, отмечено ли в ней это ребро. Если ребро не отмечено, сообщение уничтожается. Если ребро отмечено, сообщение пересылается дальше по всем инцидентным вершине неотмеченным рёбрам. Возможно также, что сообщения посылаются только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию. Сообщение уничтожается после того, как оно, пройдя по ребру, попало в вершину, где это ребро не отмечено. В частности, всегда уничтожается сообщение, посланное из истока ациклического подграфа отмеченных рёбер. Возможно также, что сообщение уничтожается раньше по тому или иному специальному критерию. Например, если вершина «знает», что она является истоком, то она может не посылать дальше сообщение, а сразу уничтожать его. Отмеченное ребро будет пройдено один раз в одном направлении – против его условной ориентации.

Не отмеченное ребро будет пройдено по одному разу в каждом направлении (кроме рёбер, инцидентных истокам, если истоки «знают», что они истоки).

Наиболее важные частные случаи размеченного графа – прямой и обратный лес. В случае обратного леса инициатором является обычно корень дерева. Сообщение двигается по рёбрам дерева от корня до листьев с размножением. Если вершина «знает», что она лист дерева, сообщение сразу уничтожается. В случае прямого леса инициатор – это, как правило, лист дерева. Сообщение двигается по рёбрам дерева от листа до корня без размножения. Если вершина «знает», что она корень дерева, она сразу уничтожает сообщение.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le D+1$ или $T \le D$, если истоки «знают», что они истоки. При рассылке от корня или к корню, соответственно, $T \le D_0+1$ или $T \le D_0$. Для обратного леса, когда инициаторы – это корни деревьев, $N \le 1$ и $E = \boldsymbol{O}(1)$, для прямого леса, когда инициаторы – это листья деревьев, $N \le n\text{-}1$ и $E = \boldsymbol{O}(n)$.

4.6. Сбор по обратному дереву

Граф размеченный, подграф отмеченных рёбер является обратным лесом.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *ОбратноеРебро* – номер выходящего обратного ребра (в корнях деревьев равен 0) размером $\boldsymbol{O}(log\Delta)$, *ЧислоРёбер* – число входящих обратных рёбер размером $\boldsymbol{O}(log\Delta)$, *СчётчикРёбер* – изменяется от *ЧислоРёбер* до 0, имеет размер $\boldsymbol{O}(log\Delta)$.

Это единственная процедура обработки со слиянием сообщений: сообщение данного типа посылается из вершины по выходящему обратному ребру только тогда, когда получены сообщения этого типа по всем входящим обратным рёбрам. Возможно и дополнительное условие посылки сообщения. Сначала *СчётчикРёбер* = *ЧислоРёбер*, потом счётчик уменьшается на 1 при получении сообщения по входящему обратному ребру. Когда (не в корне) *СчётчикРёбер* становится равным 0, посылается сообщение по выходящему обратному ребру, а *СчётчикРёбер* := *ЧислоРёбер* для дальнейшего использования.

Сбор по обратному дереву обычно начинается в листьях обратных деревьев, по каждому обратному ребру проходит ровно одно сообщение данного типа.

Возможна модификация этого алгоритма, когда ожидаются сообщения (не обязательно этого типа) не по множеству входящих обратных рёбер, а по некоторому его надмножеству, не содержащему выходящего обратного ребра. Например, по всем инцидентным рёбрам, кроме выходящего обратного ребра. В этом случае переменная *ЧислоРёбер* инициализируется не числом входящих обратных рёбер, а числом рёбер в нужном надмножестве, например, *Степень*-1. Сообщение проходит одно ребро и далее путь по дереву. По каждому ребру леса проходит не более одного сообщения, а по другому ребру – не более одного сообщения в каждом направлении.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le D$ и $T \le D_0$, если корень дерева – это корень графа. Для модифицированной процедуры, соответственно, $T \le D+1$ и $T \le D_0+1$. В любом случае $N \le 1$. $E = \boldsymbol{O}(1)$.

4.7. Рассылка до самопересечения

Граф нумерованный, номер вершины меняется от 0 до n-1 и имеет размер $\boldsymbol{O}(logn)$. Маршрут класса является путём с хордой или путём, заканчивающимся в терминальной вершине графа. Применяется v-накопление.

Базовые параметры: 1) *Тип* – тип сообщения размером $\boldsymbol{O}(1)$, 2) *Маршрут* – v-вектор пути $P = <v_1,v_2,...v_k,v_{k+1}>$ размером $\boldsymbol{O}((k+1)logn)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *Номер* – номер вершины размером $\boldsymbol{O}(logn)$.

Когда сообщение создаётся в вершине v_1, оно рассылается по всем инцидентным ей рёбрам и содержит v-вектор $P = <v_1>$. Пусть сообщение с v-вектором $P = <v_1,v_2,...v_k,v_{k+1}>$, принимается в вершине v по ребру r. Если это «новая» вершина, т.е. $\forall i=1..k+1 \quad v \ne v_i$, сообщение далее посылается по каждому ребру, инцидентному вершине v, кроме ребра r. В сообщении указывается v-вектор $P := P\cdot<v>$. Если степень вершины больше 2, происходит размножение сообщения. Сообщение уничтожается, когда оно проходит хорду пути, т.е. образуется цикл: $\exists i=1..k+1 \; v = v_i$, или когда вершина v терминальная.

Рассылка до самопересечения начинается в одной вершине. По каждому маршруту с началом в этой вершине, являющемуся путём с хордой или путём, заканчивающимся в терминальной вершине, проходит ровно одно сообщение.

Возможны модификации алгоритма, когда сообщение посылается только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию.

Оценка. $M = \boldsymbol{O}(Dlogn)$. $A = \boldsymbol{O}(log\Delta+logn)$. $F = \boldsymbol{O}(Dlogn+log\Delta) = \boldsymbol{o}_{m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le d+1$, а при рассылке от корня $T \le d_0+1$. Число N сообщений на ребре $a\to b$ не больше числа путей, начинающихся в инициаторе и заканчивающихся ребром $a\to b$. На классе всех графов N и E экспоненциально зависят от n.

4.8. Рассылка без повторения

Маршрут этого класса является путём с ребром.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *Было* – булевский признак, равный ***true***, если в вершине было сообщение этого типа.

Когда вершина получает сообщение первый раз (*Было* = ***false***), оно пересылается по каждому ребру, инцидентному вершине, кроме того, по которому оно пришло в вершину, и *Было* := ***true***. Повторные (*Было* = ***true***) принимаемые сообщения игнорируются. Если степень вершины больше 2, сообщение размножается. Сообщение уничтожается, если оно повторное (*Было* = ***true***), или когда сообщение попадает в терминальную вершину графа.

Рассылка без повторения начинается в одной вершине, обычно в корне графа, из которой сообщения рассылаются по всем инцидентным рёбрам. В связном графе для каждой вершины b будет ровно одно инцидентное ей ребро ab, по которому сообщение данного типа первый раз приходит в вершину b. Если это ребро условно ориентировать как $a \to b$, то множество так ориентированных рёбер образует прямой остов графа (ориентированный от корня). Если ребро условно ориентируется как $b \to a$, то множество так ориентированных рёбер образует обратный остов графа (ориентированный к корню). По каждому ребру остова пройдёт ровно одно сообщение в направлении $a \to b$. По каждой хорде остова пройдут ровно два сообщения, по одному в каждом направлении.

Возможны модификации алгоритма, когда сообщение посылается только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. Если сообщение рассылается по всем рёбрам, то $T \le d+1$ ($T \le d_0+1$ при рассылке от корня), а если не по всем, то $T \le D+1$ ($T \le D_0+1$ при рассылке от корня). $N \le 1$. $E = \boldsymbol{O}(1)$.

4.9. Множественная рассылка

Граф нумерованный, номер вершины меняется от 0 до n-1 и имеет размер $\boldsymbol{O}(logn)$. Множественная рассылка – это рассылка без повторения, которая параллельно ведётся, начиная с нескольких начальных вершин, которые мы назовём *инициаторами*. Для того чтобы различать сообщения от разных инициаторов, вершина хранит множество номеров инициаторов, сообщения от которых были в вершине. Обычно цель рассылки – доставить информацию в некоторые *конечные* вершины (не инициаторы), в частности, в корень, в которых сообщение не посылается дальше, а уничтожается. Время доставки информации в конечные вершины, если они есть, будем считать временем T, а общее время существования сообщений обозначим T*. Если после доставки информации в конечные вершины нужно, не дожидаясь времени T*, удалить оставшиеся в графе сообщения, можно воспользоваться универсальной процедурой очистки, описываемой ниже в подразделе 6.3.

Базовые параметры: *Тип* – тип сообщения размером $\boldsymbol{O}(1)$, *Инициатор* – номер инициатора размером $\boldsymbol{O}(logn)$. Базовые переменные: *Степень* – степень вершины размером $\boldsymbol{O}(log\Delta)$, *Номер* – номер вершины размером $\boldsymbol{O}(logn)$, *Инициаторы* – множество номеров инициаторов размером $\boldsymbol{O}(nlogn)$.

Если вершина не конечная, то первое полученное ею сообщение от данного инициатора она пересылает по каждому инцидентному вершине ребру, кроме того, по которому оно получено, а номер инициатора добавляет в переменную *Инициаторы*. Повторные сообщения от того же инициатора, а в терминальной и конечной вершине любые сообщения, уничтожаются. По ребру в одном направлении пройдёт не более одного сообщения от каждого инициатора.

Возможны модификации алгоритма, когда сообщение посылается только по тем рёбрам, которые удовлетворяют тому или иному специальному критерию.

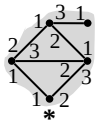
Оценка. $M = \boldsymbol{O}(logn)$. $A = \boldsymbol{O}(nlogn+log\Delta)$. $F = \boldsymbol{O}(nlogn+log\Delta) = \boldsymbol{o}_{m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. Пусть k число конечных вершин. Если $k > 0$, то $T \le d$, а если сообщение посылается не по всем рёбрам, то $T \le D$. Если только корень конечная вершина, то $T \le d_0$ и $T \le D_0$, соответственно. $T^* \le d+1$, а если сообщение посылается не по всем рёбрам, то $T^* \le D+1$. $N \le n-k$. $E = \boldsymbol{O}((n-k)logn)$.

Модификация для векторов: Множественная рассылка для случая, когда номер вершины – это f-вектор пути от корня до вершины по прямому остову, рассматривается ниже в подразделе 5.5. Здесь мы приведём только оценку.

Оценка. $M = \boldsymbol{O}(D_0log\Delta)$. $A = \boldsymbol{O}(nlog\Delta)$, $F = \boldsymbol{O}(nlog\Delta) = \boldsymbol{o}_{m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. Оценка времени T и числа сообщений N такие же. $E = \boldsymbol{O}((n-k)D_0log\Delta)$.

5. Построение остова графа

В этом разделе описывается процедура построения остова графа (см. Рис. 3), которая в дальнейшем будет использоваться либо как предварительный этап, либо как прототип алгоритмов решения задач на графе.



Граф
(корень обозначен *)

Сообщение ***Вперёд*** класса «Рассылка без повторения». Пунктиром показаны сообщения, повторно приходящие в вершину.

Сообщение ***Назад*** класса «Сбор по обратному дереву».

Рис. 3. Построение остова графа.
Fig. 3. Construction of the spanning tree.

5.1. Построение обратного остова

Используется сообщение ***Вперёд*** класса «Рассылка без повторения». Вершина имеет дополнительную переменную *ОбратноеРебро* размером $\boldsymbol{O}(log\Delta)$, предназначенную для хранения номера выходящего обратного ребра.

В начале процедуры корень создаёт сообщение ***Вперёд*** и посылает его по всем инцидентным рёбрам. Когда некорневая вершина первый раз (переменная *Было* = ***false***) получает сообщение ***Вперёд***, переменная *ОбратноеРебро* инициализируется номером ребра, по которому получено сообщение ***Вперёд***. Остальное регулируется правилами обработки сообщения класса «Рассылка без повторения». Остов будет построен через время d_0, но ещё не более 1 такта сообщения могут двигаться по графу (по хордам остова). Для определения конца построения остова, применяется процедура из следующего подраздела.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

5.2. Определение конца построения обратного остова

Если нужно не только построить обратный остов, но и определить момент завершения построения, то дополнительно используется сообщение ***Назад*** класса «Сбор по обратному дереву», где *ЧислоРёбер* понимается как число инцидентных вершине рёбер, кроме выходящего обратного ребра.

Переменные *ЧислоРёбер* и *СчётчикРёбер* в корне инициализируются степенью корня в начале процедуры построения обратного остова, а в некорневой вершине – степенью вершины минус 1, когда вершина первый раз (переменная *Было* = ***false***) получит сообщение ***Вперёд***. Некорневая вершина ожидает получения по каждому инцидентному ей ребру сообщения ***Вперёд*** или ***Назад***, что требует времени не более d_0+1, после чего посылает по выходящему обратному ребру сообщение ***Назад***. Сообщение ***Назад*** проходит путь к корню, но из-за различного и переменного времени прохождения сообщения по рёбрам графа этот путь может быть не кратчайшим. Поэтому сообщение ***Назад*** может пройти путь длиной не d_0, а D_0.

Процедура заканчивается, когда корень получит сообщение ***Вперёд*** или ***Назад*** по всем инцидентным корню рёбрам. По каждому ребру остова пройдёт сообщение ***Вперёд*** в прямом направлении и сообщение ***Назад*** в обратном направлении, а по каждой хорде остова в каждом направлении пройдёт по одному сообщению ***Вперёд***.

Оценка суммарно с построением обратного остова. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+1+D_0$. $N \le 1$. $E = \boldsymbol{O}(1)$.

5.3. Установка счётчиков входящих обратных рёбер

Одновременно с построением обратного остова и определением конца построения можно инициализировать в каждой вершине переменную *ЧислоОбратныхРёбер* размером $\boldsymbol{O}(log\Delta)$. Для этого, во-первых, инициализируется переменная *ЧислоОбратныхРёбер* := 0 в корне в начале работы, а в некорневой вершине при получении первого (переменная *Было* = ***false***) сообщения ***Вперёд***. Во-вторых, при получении сообщения ***Назад***, которое приходит по входящему обратному ребру, вершина увеличивает *ЧислоОбратныхРёбер* на 1. При посылке сообщения ***Назад*** по выходящему обратному ребру вершина присваивает *СчётчикРёбер* := *ЧислоРёбер* := *ЧислоОбратныхРёбер*. Корень делает это в конце работы. Переменные *СчётчикРёбер* и *ЧислоРёбер* могут впоследствии использоваться в базовой процедуре «Сбор по обратному дереву», когда *ЧислоРёбер* понимается как число входящих обратных рёбер.

Оценка суммарно с построением обратного остова и определением конца построения. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+1+D_0$. $N \le 1$. $E = \boldsymbol{O}(1)$.

5.4. Отметка прямых рёбер

Одновременно с построением обратного остова с определением конца построения можно построить и прямой остов, отметив рёбра остова в других их концах. Для этого в каждой вершине нужна переменная *ШкалаРёбер* – битовая шкала размером $\boldsymbol{O}(\Delta)$, содержащая по одному биту на каждое инцидентное ребро. Шкала инициализируется нулями в корне в начале работы, а в некорневой вершине при получении первого (переменная *Было* = ***false***) сообщения ***Вперёд***. При получении по ребру i сообщения ***Назад*** в i-ый бит шкалы записывается 1.

Оценка суммарно с построением остова. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta+\Delta)$. $F = \boldsymbol{O}(log\Delta+\Delta) = \boldsymbol{o}_{m/n\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+1+D_0$. $N \le 1$. $E = \boldsymbol{O}(1)$.

5.5. Быстрое построение остова

Можно уменьшить оценку времени при построении остова до оптимальной, если использовать сообщение ***Назад*** класса «Множественная рассылка», когда только корень конечная вершина. Для этого класса граф должен быть нумерованным, но мы будем нумеровать вершины одновременно с построением остова. Мы применим способ нумерации вершин, аналогичный тому, который использовался в [6],[8] для ориентированного графа. Для этого в качестве номера вершины возьмём f-вектор прямого пути от корня до этой вершины, который назовём *вектором вершины*. Он имеет размер $\boldsymbol{O}(D_0 log\Delta)$. Сообщение ***Вперёд*** посылается с f-накоплением, и вершина получает свой вектор из первого полученного ею сообщения ***Вперёд***. Как и раньше, при получении первого сообщения ***Вперёд*** по ребру i вершина запоминает i как номер выходящего обратного ребра.

Добавляется ещё одно сообщение ***Остов*** класса «Одиночная передача» без дополнительных параметров, которое при передаче из вершины a по ребру $a\to b$ сообщает вершине b, что это ребро является ребром остова. Сообщение посылается в ответ на первое сообщение ***Вперёд***, полученное вершиной a. При получении сообщения ***Остов*** (а не ***Назад*** как раньше) по ребру j вершина увеличивает *ЧислоОбратныхРёбер* на 1 и, если нужно отметить прямые рёбра, то в j-ый бит переменной *ШкалаРёбер* записывает 1.

Сообщение ***Назад*** содержит вектор создавшего его инициатора и число входящих обратных рёбер инициатора как дополнительный параметр. Сообщение создаётся в вершине, когда по каждому инцидентному ей ребру будет получено сообщение ***Вперёд*** или ***Остов***, т.е. когда полностью определится *ЧислоОбратныхРёбер* и, если нужно, *ШкалаРёбер*.

Поскольку корень конечная вершина, он не пересылает дальше сообщение ***Назад***, но запоминает вектор инициатора p и число q_p входящих в него обратных рёбер. Заметим, что входящее обратное ребро совпадает с прямым выходящим ребром с точностью до условной ориентации. Поэтому q_p равно числу выходящих прямых рёбер. Остов построен, когда множество P

запомненных векторов инициаторов содержит вектора всех вершин, что определяется условием «*векторной замкнутости*»: 1) префикс-замкнутость: $\forall p \in P$ любой префикс p принадлежит P, 2) постфикс-замкнутость: вектор вершины продолжается в P каждым прямым ребром, выходящим из неё: $\forall p \in P$ $\boldsymbol{card}(\{ f \mid p \cdot \langle f \rangle \in P \}) = q_p$ (здесь $\boldsymbol{card}(X)$ мощность множества X).

Пусть вершина a получит первое сообщение ***Вперёд*** через время r. Очевидно, $r \le d_0$. Затем, если $r = d_0$, то не более чем через 1 такт вершина a получит по каждому инцидентному ей ребру сообщение ***Вперёд***, и сообщение ***Назад*** будет создано через время не более $r+1 = d_0+1$. Если $r \le d_0-1$, то не более чем через 2 такта вершина a получит по каждому инцидентному ей ребру сообщение ***Вперёд*** или ***Остов***, и сообщение ***Назад*** будет создано через время не более $r+2 \le d_0+1$. Сообщение ***Назад*** двигается до корня не более d_0 тактов. Поэтому $T \le 2d_0+1$.

Размер вектора вершины равен $\boldsymbol{O}(D_0 log\Delta)$, поэтому переменная *Инициаторы* имеет размер $\boldsymbol{O}(nD_0 log\Delta) = \boldsymbol{o}_{m\to\infty}$. Однако на классе графов без кратных рёбер и петель, где $\Delta \le n-1$, этот размер может достигать по порядку размера описания графа $n^2 logn$. Например, если в графе есть путь от корня длиной $n-1$ по рёбрам с максимальными номерами $n-1$, то может получиться так, что k-ая вершина пути получит свой вектор размером порядка $(k-1)log(n-1)$ как вектор префикса пути длиной $k-1$, а сумма размеров векторов по всем инициаторам будет порядка $n^2 logn$.

Размер памяти автомата можно уменьшить, если в переменной *Инициаторы* хранить не список векторов, а прямое дерево [13], в котором пометить инициаторы. Это дерево будем называть *деревом инициаторов*. Оно задаётся как список описаний вершин в порядке обхода дерева в ширину. Описание вершины – это список выходящих из вершины прямых номеров рёбер дерева. Прямой номер ребра, имеющий двоичное представление $x_1,x_2,...x_r$ хранится в виде последовательности $0,x_1,0,x_2,...\ 0,x_r,1$ размером $\boldsymbol{O}(log\Delta)$. Конец описания вершины задаётся кодом 100, конец списка описаний вершин задаётся кодом 110 (в [13] это коды 10 и 11, соответственно). Если вершина – это запомненный инициатор, то код 100 заменяется кодом 101. Тогда переменная *Инициаторы* имеет размер $\boldsymbol{O}(n+nlog\Delta) = \boldsymbol{O}(nlog\Delta)$. В корне после кода 101 дополнительно помещается число входящих обратных рёбер инициатора размером $\boldsymbol{O}(log\Delta)$; суммарно по всем инициаторам добавляется $\boldsymbol{O}(nlog\Delta)$ бит.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta)$. Если прямые рёбра не отмечаются, то $A = \boldsymbol{O}(nlog\Delta)$, $F = \boldsymbol{O}(nlog\Delta) = \boldsymbol{o}_{m\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$, а если отмечаются (5.4), то $A = \boldsymbol{O}(nlog\Delta+\Delta)$, $F = \boldsymbol{O}(nlog\Delta+\Delta) = \boldsymbol{o}_{m/n\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. По ребру в одном направлении одновременно перемещаются не более одного сообщения ***Вперёд*** или ***Остов*** и не более одного сообщения ***Назад*** от каждого инициатора (числом $n-1$), поэтому $N \le n$. $E = \boldsymbol{O}(nD_0 log\Delta)$.

6. Стопор и очистка

В этом разделе мы предложим универсальную процедуру обнаружения завершения работы любого алгоритма, использующего сообщения конечного числа типов, которую мы назвали *стопором*, а также процедуру *очистки* графа от сообщений указанных типов.

6.1. Стопор

Любой алгоритм $\mathcal{A}$, использующий сообщения конечного числа типов, завершает свою работу не позже того момента, когда в графе исчезнут все сообщения используемых им типов. Предполагается, что работа алгоритма $\mathcal{A}$ инициируется корнем, т.е. сообщения указанных типов генерирует только корень, а любая некорневая вершина посылает сообщение одного из указанных типов только в таком срабатывании, в котором она получает сообщение одного из указанных типов. Если нужно отслеживать наличие или отсутствие в графе сообщений любого из типов $t_1,t_2,...t_k$, то множество этих типов $t = \{t_1,t_2,...t_k\}$ понимается как обобщённый тип, и считается, что сообщение имеет тип t, если оно имеет тип $t_i \in t$.

Стопор используется корнем в цикле: корень начинает процедуру стопора и, когда она заканчивается и оказывается, что в графе ещё есть сообщение указанного типа, начинается новая процедура стопора, и так далее до тех пор, пока не окажется, что сообщений указанного типа нет.

Стопор строится как модификация процедуры (не быстрого) построения обратного остова с определением конца построения (раздел 5). В сообщение ***Вперёд*** добавлен один параметр: тип учитываемого сообщения *Утип* размером $\boldsymbol{O}(1)$. Задача сообщений ***Вперёд*** – известить все вершины о начале процедуры стопора. Сообщение ***Назад*** содержит дополнительный булевский параметр *Найдено*, который сообщает о том, обнаружено или нет сообщение типа *Утип*. Задача сообщений ***Назад*** – доставить в корень дизъюнкцию их параметров *Найдено*.

Вершина находится в одном из двух режимов работы: *Рабочий* и *Учёт*. Режим работы хранится в переменной *Режим* размера $\boldsymbol{O}(1)$. Также в вершине имеется дополнительная булевская переменная *Найдено*. Корень переходит из режима *Рабочий* в режим *Учёт*, когда он начинает процедуру стопора. Можно считать, что это происходит сразу после начала работы алгоритма $\mathcal{A}$, когда корень генерирует первое сообщение типа *Утип*, или при завершении очередной процедуры стопора, если в графе ещё остались сообщения типа *Утип* и должна начаться следующая процедура стопора. В этот момент корень инициализирует переменную *Найдено* := ***false***, создаёт сообщение ***Вперёд*** и рассылает его по всем инцидентным корню рёбрам.

Некорневая вершина в режиме *Рабочий*, получив сообщение ***Вперёд***, переходит в режим *Учёт* и присваивает своей переменной *Найдено* := ***false***. Когда вершина в режиме *Учёт* посылает сообщение типа *Утип*, она

присваивает своей переменной *Найдено* := ***true***. Когда вершина получает сообщение ***Назад*** с параметром *Найдено* = ***true***, она присваивает своей переменной *Найдено* := ***true***. Когда вершина посылает сообщение ***Назад***, параметр *Найдено* делается равным значению переменной *Найдено*, и вершина переходит в режим *Рабочий*. Когда процедура завершается, корень переходит в режим *Рабочий*. Если при этом в корне *Найдено* = ***true***, то корень начинает новую процедуру стопора, снова переходя в режим *Учёт*.

Утверждение 1: Если в начале процедуры стопора в графе нет сообщений типа *Ytun*, и во время процедуры корень не генерирует таких сообщений, то при окончании процедуры стопора в корне *Найдено* = ***false***.

Доказательство: Действительно, при этих условиях во время процедуры стопора в графе не могут возникнуть сообщения типа *Ytun*, поскольку некорневые вершины не генерируют такие сообщения, а только пересылают их. Поэтому в каждой вершине переменная *Найдено* = ***false*** в течение всей процедуры стопора. А тогда в конце процедуры стопора в корне *Найдено* = ***false***, что и требовалось доказать.

Утверждение 2: Если при окончании процедуры стопора в корне *Найдено* = ***false***, то в графе нет сообщений типа *Ytun*.

Доказательство: Допустим, это не так, и при окончании процедуры стопора в графе на некотором ребре $a \to b$ есть сообщение M_1 типа *Ytun*. Поскольку при окончании процедуры стопора в корне *Найдено* = ***false***, в каждой вершине также *Найдено* = ***false***. Сообщения типа *Ytun* не генерируются во время процедуры стопора. Следовательно, сообщение M_1 послано из вершины a до её перехода в режим *Учёт* и всё ещё находится на ребре, когда вершина b выходит из режима *Учёт*. Но вершина b выходит из режима *Учёт* только после получения по ребру $a \to b$ сообщения M_2 типа ***Вперёд*** или ***Назад***, а такое сообщение посылается только в режиме *Учёт*. Следовательно, M_2 посылается из a позже M_1, а принимается в b раньше, чего быть не может. Мы пришли к противоречию, и утверждение доказано.

Оценка. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+D_0+1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

Для определения конца работы алгоритма $\mathcal{A}$, использующего сообщения типа *Ytun*, процедура стопора запускается в цикле до тех пор, пока она не завершится с переменной в корне *Найдено* = ***false***. Поэтому в тот момент времени, когда в графе реально исчезают сообщения типа *Ytun*, уже может быть запущена процедура стопора, которая может дать *Найдено* = ***true***. Но зато следующая процедура стопора даст гарантированно *Найдено* = ***false***. Поэтому время может удвоиться: $T \le 2(d_0+D_0+1)$.

6.2. Быстрый стопор

Процедуру стопора можно реализовать как модификацию процедуры быстрого построения остова (подраздел 5.5). Вершина находится в режиме

Учёт не более 2 тактов между получением первого сообщения ***Вперёд*** и посылкой сообщения ***Назад***: за эти 2 такта по каждому инцидентному вершине ребру придёт сообщение ***Вперёд*** или ***Остов***. В доказательстве утверждения 2 сообщение M_2 – это сообщение типа ***Вперёд*** или ***Остов***.

После выполнения процедуры быстрого стопора в графе могут остаться сообщения ***Назад***, поскольку они рассылаются множественной рассылкой. Для того чтобы эти сообщения не перепутывались с сообщениями ***Назад*** следующей процедуры быстрого стопора, будем разделять эти процедуры на чётные и нечётные. Чётность процедуры является дополнительным параметром сообщений ***Вперёд*** и ***Назад***. Когда вершина первый раз получает сообщение ***Вперёд***, она запоминает чётность процедуры и далее игнорирует все принимаемые сообщения ***Назад*** предыдущей чётности. Когда вершина посылает сообщение ***Назад***, она помещает в него текущую чётность, а сама готова к приёму сообщения ***Вперёд*** следующей чётности, считая его первым сообщением ***Вперёд*** следующей процедуры. Важно отметить, что процедура быстрого стопора не заканчивается, пока по каждому ребру в каждом направлении не пройдёт сообщение ***Вперёд*** или ***Остов***. Эти сообщения «очищают» рёбра от сообщений ***Назад*** предыдущей чётности. Поэтому, когда закончится процедура с данной чётностью, в графе не останется сообщений ***Назад*** предыдущей чётности.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta)$. $A = \boldsymbol{O}(nlog\Delta)$. $F = \boldsymbol{O}(nlog\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le n$. $E = \boldsymbol{O}(nD_0 log\Delta)$.

6.3. Очистка и быстрая очистка

Процедуры стопора и быстрого стопора легко модифицировать в процедуры очистки и быстрой очистки, которые вместо того, чтобы определять, есть ли в графе сообщения указанного типа, удаляют все такие сообщения. Вершина после перехода в режим *Учёт* не посылает сообщений типа *Ytun*. Правда, если применяется процедура быстрой очистки, то она использует сообщения класса «Множественная рассылка», и эти её сообщения могут остаться в графе при завершении очистки, хотя сообщений типа *Ytun* уже не будет.

7. Дерево кратчайших путей

Дерево кратчайших путей – это такой остов графа, что расстояние от корня до любой вершины в остове и в графе совпадают. Высота этого дерева равна d_0, что позволяет впоследствии использовать его для рассылки от корня до каждой вершины (если отмечены прямые рёбра, т.е. дерево прямое) и обратно (если отмечены обратные рёбра, т.е. дерево обратное) за время не более $2d_0$.

7.1. Построение обратного дерева кратчайших путей

Для того чтобы построить обратное дерево кратчайших путей, модифицируем алгоритм построения остова, описанный в подразделе 5.1. Сообщение ***Вперёд***

будет накапливать длину пройденного им маршрута в дополнительном параметре *Длина* размером $\boldsymbol{O}(logD_0)$. Некорневая вершина сохраняет в своей дополнительной переменной *Расстояние* размером $\boldsymbol{O}(logD_0)$ минимум из параметров *Длина* сообщений ***Вперёд***, полученных этой вершиной, а в переменной *ОбратноеРебро* номер ребра, по которому получено сообщение ***Вперёд*** с минимальным значением параметра *Длина*.

Корень посылает сообщение ***Вперёд*** с параметром *Длина* = 1. Некорневая вершина, получив сообщение ***Вперёд*** первый раз, инициализирует *Расстояние* := *Длина*, *ОбратноеРебро* := *f*, где *f* – номер ребра, по которому получено сообщение. Сообщение ***Вперёд*** посылается дальше по всем инцидентным вершине рёбрам, кроме ребра *f*, с параметром *Длина*, увеличенным на 1. Повторное сообщение ***Вперёд*** игнорируется только в том случае, когда *Длина* ≥ *Расстояние*. Если же *Длина* < *Расстояние*, то выполняются присваивания *Расстояние* := *Длина* и *ОбратноеРебро* := *f*, где *f* – номер ребра, по которому получено сообщение. Сообщение ***Вперёд*** посылается дальше по всем инцидентным вершине рёбрам, кроме ребра *f*, с параметром *Длина*, увеличенным на 1.

Для определения конца построения используется универсальный стопор – процедура стопора из раздела 6 для типа сообщения ***Вперёд***.

Покажем, что через d_0 тактов в графе не останется сообщений ***Вперёд***, и будет построено дерево кратчайших путей. Действительно, пусть расстояние от корня до вершины равно *r*. Тогда не более чем через *r* тактов вершина получит сообщение, прошедшее от корня до этой вершины кратчайший путь. Тем самым, после *r* тактов вершина перестанет посылать сообщение ***Вперёд***. Поэтому сообщения ***Вперёд*** исчезнут из графа не позже чем через d_0 тактов. Очевидно, что если в каждой некорневой вершине выбрать в качестве обратного ребра любое инцидентное вершине ребро, лежащее на кратчайшем пути от корня до вершины, то получится дерево кратчайших путей.

Оценка без учёта стопора. По сравнению с алгоритмом 5.1 размер сообщения и памяти автомата увеличиваются на $\boldsymbol{O}(logD_0)$. $M = \boldsymbol{O}(logD_0)$. $A = \boldsymbol{O}(logD_0+log\Delta)$. $F = \boldsymbol{O}(logD_0+log\Delta) = \boldsymbol{o}_{m,n\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0$. Вершина может получить несколько подряд сообщений ***Вперёд*** со строго уменьшающимися значениями параметра *Длина* и переслать их дальше. Для пересылаемых дальше сообщений *Длина* ≤ D_0. Поэтому $N \le D_0$, $E = \boldsymbol{O}(D_0logD_0)$.

7.2. Установка счётчиков входящих обратных рёбер

В подразделе 5.3 счётчики входящих обратных рёбер устанавливались одновременно с построением обратного остова и определением конца построения. Здесь мы применим аналогичный алгоритм, но только после определения конца построения обратного дерева кратчайших путей. Вместо сообщения ***Вперёд*** класса «Рассылка без повторения» используется сообщение ***Вперёд*1** класса «Рассылка против отмеченных рёбер», где

отмеченное ребро – это ребро обратного дерева кратчайших путей. Сообщение ***Вперёд*1** отличается от сообщения ***Вперёд*** не только своим классом, т.е. способом рассылки, но также тем, что не происходит инициализации переменной *ОбратноеРебро*, поскольку она уже установлена при построении дерева кратчайших путей. Соответственно, сообщение ***Назад*** создаётся и посылается по выходящему обратному ребру тогда, когда вершина получит по каждому инцидентному ей ребру сообщение ***Вперёд*1** (вместо ***Вперёд***) или сообщение ***Назад***.

Оценка без учёта построения дерева кратчайших путей. $M = \boldsymbol{O}(1)$. $A = \boldsymbol{O}(log\Delta)$. $F = \boldsymbol{O}(log\Delta) = \boldsymbol{o}_{n,m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

Можно совместить установку счётчиков с процедурой стопора для определения конца построения дерева кратчайших путей. Переименуем сообщения ***Вперёд*** в стопоре на ***Вперёд*2**. Первое сообщение ***Вперёд*2**, приходящее в вершину, инициирует сброс счётчика в ноль и посылку сообщения ***Остов*** по текущему выходящему обратному ребру. Получая сообщение ***Остов***, вершина увеличивает счётчик на 1. Если очередная процедура стопора не обнаруживает сообщений ***Вперёд***, то счётчики будут установлены правильно. Это даёт суммарную оценку верхней границы времени построения дерева кратчайших путей с установкой счётчиков не $d_0+2(d_0+D_0+1)+2d_0+1 = 5d_0+2D_0+3$, а $d_0+2(d_0+D_0+1) = 3d_0+2D_0+1$.

7.3. Отметка прямых рёбер

После определения конца построения обратного дерева кратчайших путей можно построить и прямой остов, т.е. отметить рёбра остова в других их концах. Это можно совместить с установкой счётчиков входящих обратных рёбер. Алгоритм аналогичен описанному в подразделе 5.4. Переменная *ШкалаРёбер* – инициализируется нулями при получении сообщения ***Вперёд*1** по выходящему обратному ребру (в корне с самого начала). При получении по ребру i сообщения ***Назад*** в i-ый бит шкалы записывается 1.

Оценка без учёта построения дерева кратчайших путей. $M = \boldsymbol{O}(1)$, $A = \boldsymbol{O}(log\Delta+\Delta)$, $F = \boldsymbol{O}(log\Delta+\Delta) = \boldsymbol{o}_{m/n\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le 1$. $E = \boldsymbol{O}(1)$.

Можно совместить отметку прямых рёбер с процедурой стопора для определения конца построения дерева кратчайших путей аналогично тому, как это делается в подразделе 7.2. Получая сообщение ***Остов*** по некоторому ребру, вершина отмечает это ребро в шкале рёбер.

8. Нумерация графа

В этом разделе мы предложим несколько алгоритмов нумерации графа. Вершина имеет дополнительную переменную *Номер*, которая инициализируется в процедуре нумерации графа. Вершины можно пронумеровать произвольными уникальными идентификаторами, например, векторами вершин. Однако отметим, что минимальная память, требуемая для

хранения идентификаторам вершины, получается тогда, когда вершины пронумерованы числами от 0 до n-1.

8.1. Нумерация векторами

Если на нумерацию графа налагается единственное ограничение: разные вершины имеют разные номера, – то в качестве номера вершины можно использовать вектора вершин. Процедура нумерация векторами является модификацией процедуры построения обратного остова с определением конца построения (подраздел 5.2). Сообщение ***Вперёд*** выполняется с f-накоплением и поэтому имеет размер не $\boldsymbol{O}(1)$ как в подразделе 5.2, а $\boldsymbol{O}(D_0 log\Delta)$. Когда такое сообщение первый раз (переменная *Было* = ***false***) попадает в вершину, f-вектор пути запоминается в переменной *Номер* вершины.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta)$, $A = \boldsymbol{O}(D_0 log\Delta)$. $F = \boldsymbol{O}(D_0 log\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0+D_0+1$. $N \le 1$. $E = \boldsymbol{O}(D_0 log\Delta)$.

Модификация 1. Если предварительно построен обратный остов с установкой счётчиков входящих обратных рёбер и отметкой прямых рёбер, то можно предложить другой алгоритм нумерации векторами. Используются сообщения ***Вперёд*** класса «Рассылка по отмеченным рёбрам» с f-накоплением размером $\boldsymbol{O}(D_0 log\Delta)$, где отмеченными рёбрами считаются прямые рёбра остова, и сообщение уничтожается в листе остова, и сообщение ***Назад*** класса «Сбор по обратному дереву», где *ЧислоРёбер* понимается как число входящих обратных рёбер. Вершина, получая сообщение ***Вперёд***, запоминает f-вектор пути как свой вектор вершины. Листовая вершина остова, кроме этого, создаёт сообщение ***Назад*** без дополнительных параметров. Процедура заканчивается, когда в корне становится *СчётчикРёбер* = 0, после чего для дальнейшего использования *СчётчикРёбер* := *ЧислоРёбер*. Сообщение ***Вперёд*** проходит путь от корня до листа остова длиной не более D_0, после чего сообщение ***Назад*** проходит постфикс этого пути в обратном направлении.

Оценка без учёта ресурсов для предварительного построения остова. $M = \boldsymbol{O}(D_0 log\Delta)$. $A = \boldsymbol{O}(D_0 log\Delta)$. $F = \boldsymbol{O}(D_0 log\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2D_0$, а если остов – это дерево кратчайших путей, то $T \le 2d_0$. $N \le 1$. $E = \boldsymbol{O}(D_0 log\Delta)$.

Модификация 2: Алгоритм модификации 1 можно применить и тогда, когда предварительно построен обратный остов с установкой счётчиков входящих обратных рёбер, но без отметки прямых рёбер. Для этого достаточно использовать сообщение ***Вперёд*** с f-накоплением не класса «Рассылка по отмеченным рёбрам», а класса «Рассылка против отмеченных рёбер», где под отмеченными рёбрами понимаются рёбра обратного остова.

Поскольку установлены счётчики входящих обратных рёбер, вершина «знает», что является листом, если в ней счётчик равен нулю, и в этом случае не посылает сообщение ***Вперёд*** дальше. Поэтому сообщение ***Вперёд*** проходит путь от корня до листа остова длиной не более D_0. После этого сообщение ***Назад*** проходит постфикс этого пути в обратном направлении.

Оценка без учёта ресурсов для предварительного построения остова. $M = \boldsymbol{O}(D_0 log\Delta)$. $A = \boldsymbol{O}(D_0 log\Delta)$. $F = \boldsymbol{O}(D_0 log\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2D_0$, а если остов – это дерево кратчайших путей, то $T \le 2d_0$. $N \le 1$. $E = \boldsymbol{O}(D_0 log\Delta)$.

8.2. Быстрая нумерация векторами

Процедуру нумерации векторами можно построить как модификацию процедуры быстрого построения остова (подраздел 5.5).

Оценка. $M = \boldsymbol{O}(D_0 log\Delta)$. $A = \boldsymbol{O}(nlog\Delta)$. $F = \boldsymbol{O}(nlog\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le n$. $E = \boldsymbol{O}(nD_0 log\Delta)$.

8.3. Нумерация по алгоритму Тэрри

Для того чтобы пронумеровать граф числами от 0 до n-1, можно воспользоваться хорошо известным алгоритмом Тэрри обхода неориентированного графа [14]. Сообщение обходит граф, проходя по каждому ребру ровно 2 раза. Более строго, здесь используются сообщения трёх типов: ***Вперёд*** класса «Рассылка без повторения» при пересылке по ещё не пройденным рёбрам, ***ОткатПоХорде*** класса «Одиночная передача» при возвращении по хорде остова, и ***Назад*** класса «Рассылка по отмеченным рёбрам», где отмеченное ребро – это выходящее обратное ребро, и сообщение уничтожается сразу после прохода этого ребра. См. Рис. 4.

Дополнительные переменные вершины: *МойНомер* размером $\boldsymbol{O}(logn)$, *Ребро* размером $\boldsymbol{O}(log\Delta)$, в которой хранится номер ребра, по которому последний раз из вершины посылалось сообщение ***Вперёд***, и *ШкалаРёбер* размером $\boldsymbol{O}(\Delta)$, в которой отмечаются пройденные рёбра: как при посылке, так и при получении по ребру сообщения ***Вперёд***.

Переменные *Ребро* и *ШкалаРёбер* инициализируются нулями: в корне – в начале работы, а в некорневой вершине – при получении первого (*Было* = ***false***) сообщения ***Вперёд***. Затем некорневая вершина дополнительно присваивает: *ОбратноеРебро* := f, где f – ребро, по которому получено сообщение ***Вперёд***, *ШкалаРёбер*(f) := 1. Если все рёбра, инцидентные некорневой вершине, пройдены, что проверяется по переменной *ШкалаРёбер*, то посылается сообщение ***Назад*** по выходящему обратному ребру. Если все рёбра пройдены в корне, обход закончен. Если не все рёбра, инцидентные вершине, пройдены, то посылается сообщение ***Вперёд*** (с использованием специального критерия) по одному ребру – следующему после *Ребро* непройденному ребру f. *Ребро* := f и *ШкалаРёбер*(f) := 1. Когда вершина получает сообщение ***Вперёд*** повторно (*Было* = ***true***), то это сообщение приходит по хорде. Вершина отмечает эту хорду в переменной *ШкалаРёбер* и в обратном направлении по хорде посылает сообщение ***ОткатПоХорде***. Когда вершина получает сообщение ***Назад*** или ***ОткатПоХорде***, она проверяет непройденность рёбер и, если нужно, посылает сообщение ***Назад*** или ***Вперёд*** как описано выше.

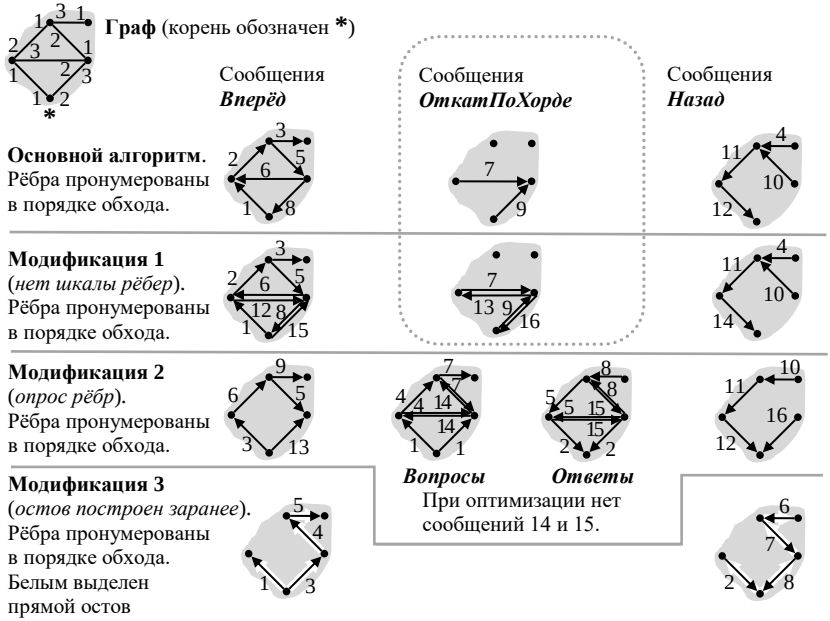


Рис. 4. Нумерация графа по алгоритму Тэрри.
Fig. 4. Numbering of the graph by the Terry algorithm.

Для нумерации вершин достаточно, чтобы сообщение каждого из этих трёх типов содержало параметр *Номер* размера $\boldsymbol{O}(\log n)$, который должен быть присвоен очередной вершине. Нумерация начинается с того, что корень присваивает себе *МойНомер* := 0, и посылает сообщение ***Вперёд*** с параметром *Номер* = 1. Когда некорневая вершина получает сообщение ***Вперёд*** первый раз (*Было* = ***false***), она получает свой номер из сообщения ***Вперёд***: *МойНомер* := *Номер*, а *Номер* + 1 становится параметром посылаемого далее сообщения ***Вперёд*** или ***Назад***. В остальных случаях параметр *Номер* из принятого сообщения без изменения переписывается в посылаемое сообщение.

Оценка. $M = \boldsymbol{O}(\log n)$. $A = \boldsymbol{O}(\log n+\Delta)$. $F = \boldsymbol{O}(\log n+\Delta) = \boldsymbol{o}_{m/n\to\infty}\ [\boldsymbol{o}_{n\to\infty}]$. $T \le 2m$. $N \le 1$, поскольку алгоритм последовательный. $E = \boldsymbol{O}(\log n)$.

Модификация 1. Можно сократить размер автомата за счёт увеличения времени работы. Достаточно просто отказаться от шкалы рёбер: номер ребра, по которому посылается сообщение ***Вперёд***, – это следующий номер после *Ребро*, отличный от *ОбратноеРебро*. По хорде остова сообщения будут проходить теперь не 2, а 4 раза: из каждого конца хорды туда и обратно.

Оценка. $M = \boldsymbol{O}(\log n)$, $A = \boldsymbol{O}(\log n\Delta)$, $F = \boldsymbol{O}(\log n\Delta) = \boldsymbol{o}_{m\to\infty}\ [\boldsymbol{o}_{n\to\infty}]$. Время увеличится на удвоенное число хорд остова: $T \le 4m-2n+2$. $N \le 1$. $E = \boldsymbol{O}(\log n)$.

Модификация 2. Можно, наоборот, уменьшить время работы алгоритма, сохраняя шкалу рёбер и размер автомата $\boldsymbol{O}(log n+\Delta)$. Для этого корень вначале, а другая вершина при получении сообщения ***Вперёд*** выполняет опрос инцидентных рёбер, чтобы установить, какие из них являются хордами остова. Не считая опросов, выполняться будет обход остова, а не всего графа. В каждую некорневую вершину придёт ровно одно сообщение ***Вперёд***.

Переменная *ШкалаРёбер* хранит «0» для прямых выходящих рёбер и (до опроса) неопрошенных рёбер. С самого начала в корне *ШкалаРёбер* нулевая.

Опрос распараллеливается, используя сообщения ***Вопрос*** и ***Ответ*** класса «Вопрос/Ответ». ***Вопрос*** посылается по всем рёбрам, для которых *ШкалаРёбер* содержит «0». ***Ответ*** содержит булевский параметр *Хорда*, показывающий, является ли ребро хордой остова или нет. Пусть вершина получает ***Вопрос*** по ребру x. Если *Было* = ***false***, то вершина посылает ***Ответ*** с параметром *Хорда* = ***false*** и устанавливает *Было* := ***true***, *ОбратноеРебро* := x, *ШкалаРёбер* инициализируется нулями, а потом *ШкалаРёбер*(x) := 1. В противном случае вершина посылает ***Ответ*** с параметром *Хорда* = ***true***. Вершина, получающая по ребру y сообщение ***Ответ*** с параметром *Хорда* = ***true***, устанавливает *ШкалаРёбер*(y) := 1. Сообщение ***Вперёд*** посылается после получения сообщений ***Ответ*** на все посланные сообщения ***Вопрос***. Сообщение ***Вперёд*** теперь относится к классу «Рассылка по отмеченным рёбрам»: оно посылается только по выходящим прямым рёбрам, но, как и прежде, только по одному ребру – следующему после *Ребро*.

Параметр *Номер* имеется в каждом сообщении. При получении сообщения ***Вперёд*** *МойНомер* := *Номер*, а параметр *Номер* в посылаемом сообщении увеличивается на 1. В остальных случаях параметр *Номер* из принятого сообщения без изменения переписывается в посылаемое сообщение.

Оценка. $M = \boldsymbol{O}(log n)$. $A = \boldsymbol{O}(log n+\Delta)$. $F = \boldsymbol{O}(log n+\Delta) = \boldsymbol{o}_{m/n\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. Обход остова занимает время не более $2(n-1)$, а в каждой из n вершин опрос рёбер занимает не более 2 тактов. $T \le 4n-2$. $N \le 1$. $E = \boldsymbol{O}(log n)$.

Оптимизация. Когда вершина первый раз получает сообщение ***Вопрос*** по ребру x, она отмечает это ребро как хорду: *ШкалаРёбер*(x) := 1. Эта вершина не будет опрашивать это ребро. В наихудшем случае время T не уменьшится, а в общем случае опрос рёбер делают не все, а $k = 1..n$ вершин, и $T = 2n+2k-2$.

Модификация 3. Если предварительно построен обратный остов с установкой счётчиков входящих обратных рёбер и отметкой прямых рёбер, то нумерацию естественно делать обходом не графа, а остова. Используются сообщение ***Вперёд*** класса «Рассылка по отмеченным рёбрам», которое, как и в модификации 2, посылается по одному выходящему прямому ребру, и сообщение ***Назад*** класса «Рассылка по отмеченным рёбрам» при возвращении по выходящему обратному ребру. Оба сообщения имеют дополнительный параметр *Номер*, а вершина имеет дополнительную переменную *Ребро*.

Оценка без учёта предварительного построения остова. $M = \boldsymbol{O}(logn)$. $A = \boldsymbol{O}(logn+log\Delta)$. $F = \boldsymbol{O}(logn\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2(n\text{-}1)$. $N \le 1$. $E = \boldsymbol{O}(logn)$.

8.4. Нумерация через корень

Можно сократить время нумерации числами от 0 до n-1 за счёт распараллеливания так же, как в случае нумерации векторами. Для этого применяется модификация процедуры построения обратного остова с определением конца построения (см. Рис. 5).

Добавляются два новых сообщения: ***ДайНомер*** класса «Рассылка по отмеченным рёбрам» с r-накоплением размером $\boldsymbol{O}(D_0 log\Delta)$, где отмеченными рёбрами считаются выходящие обратные рёбра, и сообщение уничтожается, когда доходит до корня, и ***БериНомер*** класса «Пересылка по запомненному маршруту» с длиной маршрута не более D_0 и дополнительным параметром *Номер* размером $\boldsymbol{O}(logn)$. В вершине имеются дополнительные переменные *МойНомер* размером $\boldsymbol{O}(logn)$ и булевский *ПризнакНумерации*, а в корне – переменная *ТекущийНомер* размером $\boldsymbol{O}(logn)$.

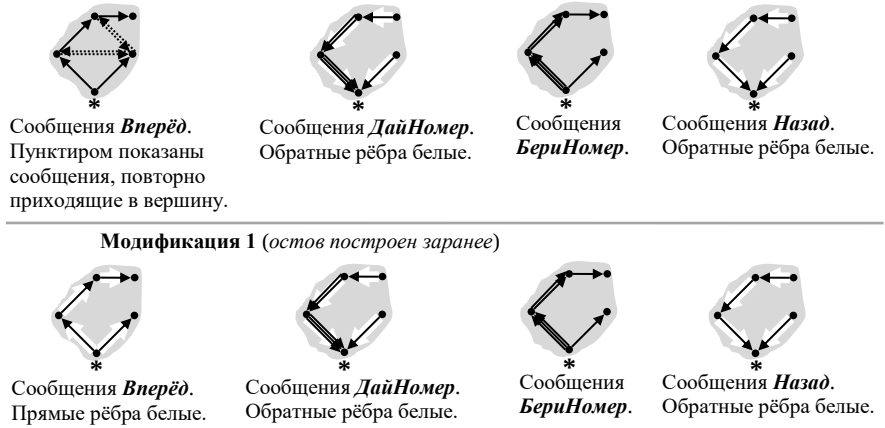


Рис. 5. Нумерация через корень.
Fig. 5. Numbering through the root.

Работа начинается с того, что корень присваивает себе номер 0, т.е. *МойНомер* := 0, инициализирует переменные *ПризнакНумерации* := ***true*** и *ТекущийНомер* := 0, создаёт сообщение ***Вперёд*** и посылает его по всем инцидентным корню рёбрам.

Когда вершина v первый раз (переменная *Было* = ***false***) получает сообщение ***Вперёд***, инициализируется переменная *ПризнакНумерации* := ***false*** и создаётся сообщение ***ДайНомер***. Это сообщение двигается по обратному пути до корня с накоплением r-вектора пути. Корень получает из принятого сообщения ***ДайНомер*** r-вектор $<r_1,...r_k>$ пройденного сообщением пути. Очевидно,

обратная последовательность $<r_k,...r_1>$ – это *f*-вектор пути от корня до вершины *v*. Получая сообщение ***ДайНомер***, корень увеличивает *ТекущийНомер* на 1 и создаёт сообщение ***БериНомер*** класса «пересылка по запомненному маршруту», в котором запомненный маршрут – это *f*-вектор $<r_k,...r_1>$ пути от корня до вершины *v*, и есть дополнительный параметр *Номер* размером $\boldsymbol{O}(log n)$, который устанавливается равным значению переменной *ТекущийНомер*. Когда сообщение ***БериНомер*** пройдёт запомненный в нём путь, конечная вершина этого пути, т.е. вершина *v*, запоминает значение параметра *Номер* в переменной *МойНомер*.

Также изменяется условие создания сообщения ***Назад***: дополнительно требуется, чтобы вершина была уже пронумерована. Это означает, что сообщение ***Назад*** создаётся либо 1) при получении сообщения ***Вперёд*** или ***Назад***, когда *СчётчикРёбер* становится равен нулю, а вершина уже пронумерована, т.е. *ПризнакНумерации* = ***true***, либо 2) в момент нумерации вершины (*ПризнакНумерации* := ***true***) при получении сообщения ***БериНомер***, предназначенного этой вершине (в сообщении *f*-вектор нулевой длины), если *СчётчикРёбер* = 0. В обоих случаях после создания сообщения ***Назад*** *СчётчикРёбер* := *ЧислоРёбер* для дальнейшего использования. Когда в корне переменная *СчётчикРёбер* станет равной нулю, нумерация закончена.

Сообщение ***ДайНомер*** создаётся, когда сообщение ***Вперёд*** проходит путь от корня до вершины *v* за время не более d_0, и само проходит этот путь в обратном направлении от вершины *v* до корня за время не более D_0, после чего сообщение ***БериНомер*** проходит этот путь в прямом направлении от корня до вершины *v* за время не более D_0, а после этого сообщение ***Назад*** двигается по тому же пути от вершины *v* в сторону корня за время не более D_0. Тем самым, время $T \le d_0+3D_0$ при условии $D_0 > 0$. Если $D_0 = 0$, то в связном графе только одна вершина, а все рёбра – петли. В этом случае требуется время для прохождения петель: $T = 1$.

Сообщения ***ДайНомер*** от всех вершин, кроме корня, т.е. от *n*-1 вершины, могут оказаться на одном ребре (а потом сообщения ***БериНомер*** для всех вершин, кроме корня, могут оказаться на одном ребре).

Оценка. $M = \boldsymbol{O}(D_0 log\Delta+logn)$. $A = \boldsymbol{O}(log\Delta+logn)$. $F = \boldsymbol{O}(D_0 log\Delta+logn) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le max\{d_0+3D_0,1\}$. $N \le max\{n-1,1\}$. $E = \boldsymbol{O}(n(D_0 log\Delta+logn))$.

Модификация 1. Эта модификация применяется тогда, когда уже построен обратный остов с установкой счётчиков входящих обратных рёбер и отметкой прямых рёбер. Сообщение ***Вперёд*** теперь будет относиться к классу «Рассылка по отмеченным рёбрам», где отмеченные рёбра – это прямые выходящие рёбра. Сообщение ***Назад*** теперь будет относиться к классу «Сбор по обратному дереву», где *ЧислоРёбер* понимается как число входящих обратных рёбер.

В начале корень создаёт и рассылает по выходящим из корня прямым рёбрам сообщение ***Вперёд***. Сообщения ***ДайНомер*** и ***БериНомер*** такие же, как в описанном выше алгоритме. Некорневая вершина получит сообщение ***Вперёд***

один раз и создаст сообщение ***ДайНомер***. Дополнительное условие посылки сообщения ***Назад*** такое же: вершина должна быть уже пронумерованной.

Процедура заканчивается, когда в корне становится *СчётчикРёбер* = 0, после чего *СчётчикРёбер* := *ЧислоРёбер* для дальнейшего использования. Если $n = 1$, с самого начала в корне *СчётчикРёбер* = 0, поэтому сообщения не посылаются и процедура заканчивается после присвоения корню номера 0.

Оценка без учёта предварительного построения остова. $M = \boldsymbol{O}(D_0 log\Delta + logn)$. $A = \boldsymbol{O}(log\Delta + logn)$. $F = \boldsymbol{O}(D_0 log\Delta + logn) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 4D_0$, а если остов – это дерево кратчайших путей, то $T \le 4d_0$. $N \le n$-1. $E = \boldsymbol{O}(n(D_0 log\Delta + logn))$.

8.5. Быстрая нумерация вершин через корень

Можно уменьшить время нумерации через корень, меняя в оценке D_0 на d_0, если применять модификацию быстрого построения остова (подраздел 5.5). Сообщения ***ДайНомер*** и ***БериНомер*** относятся к классу «Множественная рассылка», как и сообщение ***Назад***. Сообщение ***ДайНомер*** создаётся, как и раньше, когда вершина первый раз получает сообщение ***Вперёд***, и имеет только базовые параметры своего класса. Сообщение ***БериНомер*** создаётся, как и раньше, когда корень получит сообщение ***ДайНомер***, и имеет дополнительный параметр *Номер*. Имеется дополнительное условие создания сообщения ***Назад***: требуется, чтобы вершина была уже пронумерована.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta + logn)$. $A = \boldsymbol{O}(nlog\Delta + logn)$. $F = \boldsymbol{O}(nlog\Delta + logn) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 4d_0$+1. $N \le n$. $E = \boldsymbol{O}(n(D_0 log\Delta + logn))$.

8.6. Быстрая нумерация вершин не через корень

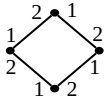
Можно ещё уменьшить время быстрой нумерации, если заметить следующий факт. В алгоритме быстрого построения остова (подраздел 5.5) сообщение ***Назад***, содержащее вектор инициатора и число входящих в него обратных рёбер, рассылается множественной рассылкой. Поэтому не только корень, но каждая вершина получает в переменной *Инициаторы* дерево инициаторов и может определить конец построения такого дерева. После этого вершина может сама вычислить свой номер, применяя один и тот же алгоритм обхода дерева в ширину. Сообщения ***ДайНомер*** и ***БериНомер*** не нужны.

Мы возьмём за основу алгоритм быстрого построения остова, но только поменяем название сообщения ***Назад*** на ***Конец***. Когда вершина определит конец построения дерева инициаторов по условию «векторной замкнутости», она вычисляет свой номер и посылает сообщение ***Назад*** класса «Множественная рассылка», сообщая как дополнительный параметр свой номер. Когда корень получит сообщения ***Назад*** от всех вершин, алгоритм заканчивается. Каждый инициатор посылает сообщение ***Конец*** через время не более d_0+1, до каждой вершины оно идёт не более d тактов. Затем сообщение ***Назад*** от каждой вершины доходит до корня за время не более d_0.

Оценка. $M = \boldsymbol{O}(D_0 log\Delta + logn)$. $A = \boldsymbol{O}(nlog\Delta + logn)$. $F = \boldsymbol{O}(nlog\Delta + logn) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le d_0 + d + d_0 + 1 \le 4d_0 + 1$. $N \le n$. $E = \boldsymbol{O}(n(D_0 log\Delta + logn))$.

9. Сбор информации о графе в памяти автомата корня

В этом разделе мы предложим алгоритм сбора полной информации о графе в памяти неограниченного автомата корня. При этом автоматы остальных вершин будут полуроботами. Прежде всего, отметим, что если граф ненумерованный, то информация о графе может содержать только *fr*-вектора маршрутов. Однако множество таких *fr*-векторов определяет граф не однозначно. Например, неразличимы графы, каждый из которых состоит из простого цикла рёбер длины $k > 0$ с нумерацией рёбер по циклу (1,2),(1,2),...(1,2) (см. Рис. 6).



Множество *fr*-векторов задаётся регулярным выражением (12)* (здесь знак «*» означает конечное число повторений выражения в скобках 0 или более раз) и не зависит от числа вершин в цикле.

Рис. 6. Множество fr-векторов не однозначно определяет граф.
Fig. 6. The set of fr-vectors does not uniquely determine the graph.

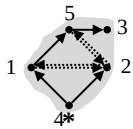
Будем предполагать, что в нумерованном графе номера вершин линейно упорядочены. Алгоритмы нумерации из раздела 8 это гарантируют как для чисел от 0 до n-1, так и для векторов. Алгоритм сбора – это модификация алгоритмов построения обратного остова с определением конца построения или быстрого построения остова. Можно также совместить сбор информации с нумерацией графа по любому из алгоритмов раздела 8.

Информация о графе собирается в виде множества описаний рёбер формата (v,f,r,v`), где $v$ и $v$` – номера вершин, инцидентных ребру, f – номер ребра в вершине v, r – номер ребра в вершине v`. Если вершины пронумерованы числами от 0 до n-1, то размер описания ребра $\boldsymbol{O}(logn\Delta)$, а если используются вектора вершин, то $\boldsymbol{O}(D_0 log\Delta)$. Описание ребра посылается как параметр сообщения ***Ребро***, которое должно дойти до корня. Вершина должна создать все нужные сообщения ***Ребро*** до того, как она создаст сообщение ***Назад***, и все эти сообщения должны идти по одним и тем же путям от вершины до корня. Тогда корень получит все сообщения ***Ребро*** до того, как получит все сообщения ***Назад***, по которым он определяет конец работы.

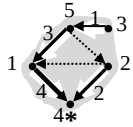
Заметим, что объединять в одном сообщении описания всех рёбер, инцидентных вершине, – плохая идея, так как это увеличивает порядок размера сообщения и, соответственно, полного размера автомата до $\Delta log n\Delta$ или (для векторов) до $\Delta D_0 log\Delta$. На классе всех графов с n вершинами и m рёбрами эти величины имеют порядок $mlognm$ или (для векторов) до $mnlogm$, т.е. автомат является неограниченным.

9.1. Сбор по остову нумерованного графа

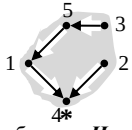
Сбор информации о нумерованном графе можно реализовать как модификацию алгоритма построения обратного остова с определением конца построения (подраздел 5.2, см. также Рис. 7). Сообщение ***Ребро*** имеет класс «Рассылка по отмеченным рёбрам», где отмеченное ребро – это выходящее обратное ребро; сообщение двигается по обратному пути до корня.



Сообщения ***Вперёд***. Пунктиром показаны сообщения, повторно приходящие в вершину.



Сообщения ***Ребро***. Обратные рёбра – белые. Хорды – пунктирные линии. Описание ребра $a \to b$ посылает вершина a. Показано число сообщений, проходящих по ребру.



Сообщения ***Назад***.

Рис. 7. Сбор информации о графе в неограниченной памяти автомата корня.
Fig. 7. Collect information about the graph in unlimited memory of the root automaton.

Описание ребра добавляется также в сообщение ***Вперёд***. Когда это сообщение посылается из вершины v по ребру f, описание ребра в нём имеет формат $(v,f,0,0)$. Когда первое сообщение ***Вперёд*** приходит в некорневую вершину $v`$ по ребру $r$, создаётся сообщение ***Ребро*** с описанием ребра $(v,f,r,v`)$. Если сообщение ***Вперёд*** повторное, то сообщение ***Ребро*** создаётся в том случае, когда $v > v`$ в линейном порядке вершин. Сообщение ***Назад*** посылается из вершины после того, как она пошлёт все свои сообщения ***Ребро***. При получении корнем сообщения ***Вперёд*** можно считать, что корень создаёт сообщение ***Ребро*** и сам же его принимает без реальной пересылки по рёбрам.

В корень поступит одно описание каждого ребра. Так как все сообщения ***Ребро*** посылаются из вершины раньше, чем сообщение ***Назад***, и эти сообщения двигаются по одному и тому же обратному пути, в момент определения конца построения остова все сообщения ***Ребро*** уже достигли корня, и корень получил полную информацию о графе.

В «худшем» случае на одном ребре могут оказаться все сообщения ***Ребро*** и одно сообщение ***Назад***, суммарно m+1.

Оценка. Размер сообщения и памяти некорневой вершины увеличивается на размер описания ребра. $T \le d_0+D_0+1$. $N \le m+1$. Если номер вершины – это число от 0 до n-1, то $M = \boldsymbol{O}(log n\Delta)$, $A = \boldsymbol{O}(log n\Delta)$ и $F = \boldsymbol{O}(log n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$, $E = \boldsymbol{O}((m+1)log n\Delta)$. Если номер вершины – это её вектор, то $M = \boldsymbol{O}(D_0 log\Delta)$, $A = \boldsymbol{O}(D_0 log\Delta)$ и $F = \boldsymbol{O}(D_0 log\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$, $E = \boldsymbol{O}((m+1)D_0 log\Delta)$.

9.2. Быстрый сбор по нумерованному графу

Сбор информации о нумерованном графе можно реализовать как модификацию алгоритма быстрого построения обратного остова (подраздел

5.5). Аналогично алгоритму из предыдущего подраздела описание ребра добавляется в сообщение ***Вперёд***. Когда это сообщение посылается из вершины v по ребру f, описание ребра в нём имеет формат (v,f,0,0). Когда первое сообщение ***Вперёд*** приходит в некорневую вершину $v`$ по ребру $r$, создаётся сообщение ***Ребро*** с описанием ребра ($v$,$f$,$r$,$v`$). Если сообщение ***Вперёд*** повторное, то сообщение ***Ребро*** создаётся в том случае, когда $v > v`$ в линейном порядке вершин. Сообщение ***Назад*** посылается из вершины после того, как она пошлёт все свои сообщения ***Ребро***. При получении корнем сообщения ***Вперёд*** можно считать, что корень создаёт сообщение ***Ребро*** и сам же его принимает без реальной пересылки по рёбрам.

Теперь сообщение ***Ребро***, как и сообщение ***Назад***, имеет класс «Множественная рассылка», когда только корень конечная вершина. На одном ребре могут оказаться одновременно сообщения ***Ребро*** для каждого ребра и сообщения ***Назад*** от каждой вершины, кроме корня, а также одно сообщение ***Вперёд*** или ***Остов***; всего не более $m+(n-1)+1 = m+n$ сообщений.

Но здесь есть проблема с инициаторами сообщений ***Ребро***. Как сказано выше, это сообщение содержит описание только одного ребра, поэтому вершина посылает множество таких сообщений ***Ребро*** с описаниями инцидентных вершине рёбер, и все они должны дойти до корня. Поэтому инициатором сообщения ***Ребро*** с описанием ребра (v,f,r,$v`$) следует считать не вершину $v`$, создавшую это сообщение, а ребро, задаваемое парой (r,$v`$). Эта проблема решается немного по-разному в зависимости от того, пронумерованы вершины числами от 0 до n-1 или векторами.

Если номер вершины – это вектор, то переменная *Инициаторы* (подраздел 5.5) хранит дерево инициаторов-вершин как список описаний вершин в порядке обхода дерева в ширину, где описание вершины – это список выходящих из вершины прямых номеров рёбер дерева. Для добавления инициаторов-рёбер, достаточно в описание вершины добавить (через разделитель) битовую шкалу рёбер, инцидентных вершине. Инициатор-ребро, задаваемый парой (r,$v`$), отмечается «1» в $r$-м бите шкалы рёбер в описании вершины $v`$. Размер переменной *Инициаторы* увеличивается на $\boldsymbol{O}(n+m)$ бит.

Оценка (номер вершины – её вектор). $M = \boldsymbol{O}(D_0 log\Delta)$. Для некорневой вершины $A = \boldsymbol{O}(n+m+nlog\Delta)$ и $F = \boldsymbol{O}(n+m+nlog\Delta) = \boldsymbol{o}_{m/n\to\infty}$ $[\boldsymbol{o}_{n\to\infty}]$. $T \le 2d_0+1$. $N \le m+n$. $E = \boldsymbol{O}((m+n)D_0 log\Delta)$.

Если вершины пронумерованы числами от 0 до n-1, то нет смысла использовать алгоритм, основанный на векторах вершин. В переменной *Инициаторы* хранится список описаний вершин в порядке возрастания их номеров. Описание вершины содержит один бит для отметки вершины-инициатора и битовую шкалу рёбер, инцидентных вершине, для отметки рёбер-инициаторов. Размер переменной равен $\boldsymbol{O}(n+m)$ бит.

Сообщение ***Вперёд*** посылается без f-накопления. Также как описано выше, в сообщение ***Вперёд*** добавляется описание ребра (v,f,0,0), которое используется для создания сообщения ***Ребро*** с описанием ребра (v,f,r,$v`$). Сообщение ***Назад***

содержит номер (а не вектор) создавшего его инициатора и число входящих обратных рёбер инициатора как дополнительный параметр.

Кроме того, меняется процедура определения конца работы, выполняемая в корне. В подразделе 5.5 эта процедура использовала дерево инициаторов с указанием числа входящих обратных рёбер для каждого инициатора. Теперь для описания дерева инициаторов достаточно в описаниях рёбер отметить рёбра остова. Для этого в описание ребра добавляется булевский признак остова, который равен ***true***, если описание ребра создаётся при получении вершиной первого сообщения ***Вперёд***, и равен ***false***, если описание ребра создаётся при получении вершиной повторного сообщения ***Вперёд***. Число входящих обратных рёбер, как и в подразделе 5.5, равно числу полученных вершиной сообщений ***Остов*** и передаётся корню в сообщении ***Назад***.

Вместо условия «векторной замкнутости» у нас будет аналогичное условие «числовой замкнутости»: 1) для каждого ребра остова (v,f,r,$v`$) оба его конца $v$ и $v`$ должны быть вершинами-инициаторами, 2) для каждой вершины-инициатора $v`$ (а также для корня) число рёбер остова вида ($v$,$f$,$r$,$v`$) должно быть равно числу обратных рёбер, входящих в вершину $v`$.

Оценка (номер вершины – число от 0 до n-1). $M = \boldsymbol{O}(log n\Delta)$. Для некорневой вершины $A = \boldsymbol{O}(n+m+log n\Delta) = \boldsymbol{O}(m)$ и $F = \boldsymbol{O}(m) = \boldsymbol{o}_{m/n\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. $T \le 2d_0+1$. $N \le m+n$. $E = \boldsymbol{O}((m+n)log n\Delta)$.

9.3. Сбор по остову с одновременной нумерацией графа

Алгоритм сбора может быть построен как модификация любого не быстрого алгоритма нумерации графа (подразделы 8.1, 8.3, 8.4). Размер сообщения и памяти некорневой вершины увеличивается на размер описания ребра.

Идея алгоритма следующая. Получая номер, вершина опрашивает соседей с помощью сообщений класса «Вопрос/Ответ». ***Вопрос*** посылается из вершины v по ребру f и принимается вершиной $v`$ по ребру $r$, а ***Ответ*** посылается обратно по тому же ребру. Оба сообщения содержат описание ребра: ***Вопрос*** – в формате ($v$,$f$,0,0), ***Ответ*** – в формате ($v$,$f$,0,0), если вершина $v`$ ещё не имеет номера, или в формате (v,f,r,$v`$), если уже имеет. Сообщение ***Ребро*** с описанием ребра ($v$,$f$,$r$,$v`$) создаёт либо вершина $v`$ при получении сообщения ***Вопрос***, если вершина v` пронумерована и $v > v`$ в линейном порядке вершин, либо вершина $v$ при получении сообщения ***Ответ*** с параметром ($v$,$f$,$r$,$v`$), если $v < v`$. Сообщение ***Ребро*** имеет класс «Рассылка по отмеченным рёбрам», где отмеченное ребро – выходящее обратное ребро. Сообщение двигается по обратному пути до корня. Сообщение ***Назад*** создаётся после окончания опроса соседей и посылки всех нужных сообщений ***Ребро***.

Опрос соседей занимает не более 2-х тактов. Опрос начинается, когда вершина получает номер, происходит параллельно в разных вершинах и не тормозит передачу других сообщений, кроме, быть может, сообщения ***Назад***. Поэтому время T увеличивается не более чем на 2 такта. Для некоторых

алгоритмов нумерации эти «лишние» два такта можно удалить, если опрос соседей выполнять с помощью уже имеющихся в алгоритме сообщений.

В основном алгоритме нумерации векторами ***Вопрос*** и ***Ответ*** не нужны. Сообщение ***Ребро*** создаётся при получении либо первичного сообщения ***Вперёд***, либо повторного сообщения ***Вперёд*** от вершины с большим вектором. В модификации 1 предварительно построен обратный остов с отметкой прямых рёбер. Для ребра остова создаётся одно сообщение ***Ребро*** при получении первичного сообщения ***Вперёд***. Вершина опрашивает только хорды (соседей на концах хорд). В модификации 2 предварительно построен обратный остов без отметки прямых рёбер. Для рёбер остова создаётся сообщение ***Ребро*** при получении либо сообщения ***Вперёд*** по выходящему обратному ребру, либо по хорде от вершины с большим вектором.

В алгоритме Тэрри нужно в сообщении ***Вперёд***, посылаемом из вершины v по ребру f, дополнительно указать описание ребра в формате (v,f,0,0). В основном алгоритме сообщение ***Ребро*** создаётся при получении сообщения ***Вперёд***. В модификации 1 нет шкалы рёбер, поэтому сообщение ***Ребро*** создаётся при получении вершиной $v`$ сообщения ***Вперёд*** либо по ребру остова, либо по хорде при условии $v` < v$. В модификации 2 (без оптимизации) уже выполняется опрос соседей. Поэтому достаточно в сообщении ***Вопрос***, посылаемом из вершины v по ребру f и принимаемом вершиной $v`$ по ребру $r$, указать описание ребра в формате ($v$,$f$,0,0), а в сообщении ***Ответ*** с параметром *Хорда* = ***true*** указать описание ребра в формате ($v$,$f$,0,0), если вершина $v`$ ещё не имеет номера, или в формате (v,f,r,$v`$), если вершина $v`$ уже пронумерована. Сообщение ***Ребро*** создаётся при получении вершиной $v`$ по ребру $r$ либо сообщения ***Вперёд***, либо сообщения ***Вопрос***, если вершина $v`$ уже пронумерована и $v > v`$, либо при получении вершиной $v$ сообщения ***Ответ*** с параметрами *Хорда* = ***true***, $r \neq 0$ и $v` > v$. В модификации 3 предварительно построен остов с отметкой прямых рёбер и выполняется обход остова, а не графа. Для рёбер остова создаётся сообщение ***Ребро*** при получении сообщения ***Вперёд***. Хорды (соседи на концах хорд) опрашиваются.

В основном алгоритме нумерации через корень и его модификации 1 нужно делать опрос соседей при получении сообщения ***БериНомер***.

9.4. Быстрый сбор с одновременной нумерацией графа

Алгоритм сбора может быть построен как модификация любого быстрого алгоритма нумерации графа (подразделы 8.2, 8.5, 8.6). Размер сообщения и памяти некорневой вершины увеличивается на размер описания ребра. Как и в предыдущем подразделе используется опрос соседей, который увеличивает время T не более чем на 2 такта. Иногда от этих 2-х тактов можно избавиться.

При быстрой нумерации векторами ***Вопрос*** и ***Ответ*** не нужны. Сообщение ***Ребро*** создаётся при получении либо первичного сообщения ***Вперёд***, либо повторного сообщения ***Вперёд*** от вершины с большим вектором.

В алгоритме быстрой нумерации через корень нужно делать опрос соседей при получении сообщения ***БериНомер***.

В алгоритме быстрой нумерации не через корень опрашиваются только хорды (соседи на их концах). Каждая вершина сама определяет конец построения дерева инициаторов по условию его «векторной замкнутости» и вычисляет свой номер. Также она может вычислить номер n_v любой вершины v по её вектору p_v. Для прямого ребра $a \to b$ вершина b знает его номер r в вершине b (выходящее из b обратное ребро) и свой вектор $p_b = p_a \cdot \langle f \rangle$, где f – номер ребра в вершине a. Вершина b строит описание ребра $a \to b$ в формате (n_a,f,r,n_b) и создаёт сообщение ***Ребро***. Также вершина b по своему вектору p_b находит в дереве инициаторов своё описание, содержащее номера выходящих прямых рёбер. Тем самым, вершина b определяет номера хорд, которые опрашивает.

10. Поиск максимального пути

В этом разделе рассмотрим решение коллективом автоматов задачи поиска максимального пути в графе (*Longest Path Problem*). Как известно, эта задача относится к классу *NP*. Но в нашей модели, допускающей неограниченное число сообщений, она решается за линейное время. Граф предполагается нумерованным числами от 0 до n-1. Сначала рассмотрим задачу поиска и, если нужно, разметки максимального пути, начинающегося в корне графа, а потом общую задачу поиска и разметки максимального пути в графе.

10.1. Максимальный путь от корня

Идея алгоритма заключается в следующем. Рассмотрим все маршруты, начинающиеся в корне и имеющие вид $B \cdot \langle e \rangle$ или C, где B – путь до вершины v, e – хорда пути B, инцидентная вершине v, C – путь, заканчивающийся в терминальной вершине. Максимальный путь от корня – это максимальный среди путей B и C. Используются два типа сообщения: ***Прямо*** класса «Рассылка до самопересечения» с *vfr*-накоплением размером $\boldsymbol{O}(D_0 log n \Delta)$ и ***Обратно*** класса «Рассылка по отмеченным рёбрам», которыми считаются рёбра обратного остова (Рис. 8). Вершина содержит дополнительную булевскую переменную *Было*, которая равна ***false***, если эта вершина ещё не получала сообщения ***Прямо***; вначале *Было* = ***false***.

В начале работы корень присваивает *Было* := ***true***, создаёт сообщение ***Прямо*** и рассылает его по всем инцидентным рёбрам. Получая первое сообщение ***Прямо***, вершина отмечает ребро, по которому принято сообщение, как выходящее обратное ребро. Так строится обратный остов графа. Сообщение ***Обратно*** создаётся при уничтожении сообщения ***Прямо***, когда его маршрут самопересекается или заканчивается в терминальной вершине. В этот момент сообщение ***Прямо*** содержит *vfr*-вектор пути с ребром вида $P_k \cdot \langle f_k \rangle$, где $P_k = \langle v_1,f_1,r_1,v_2,f_2,r_2,...v_k \rangle$, v_1 корень, f_k прямой номер ребра, по которому вершина v_k посылала сообщение. Сообщение получает вершина v_{k+1} по ребру

r_k, причём либо $v_{k+1} = v_i$ для некоторого i=1..k, либо вершина v_{k+1} терминальная. Сообщение ***Обратно*** имеет дополнительный параметр $P = P_k$, если было самопересечение, или $P = P_k \cdot \langle f_k, r_k, v_{k+1} \rangle$, если маршрут закончился в терминальной вершине.

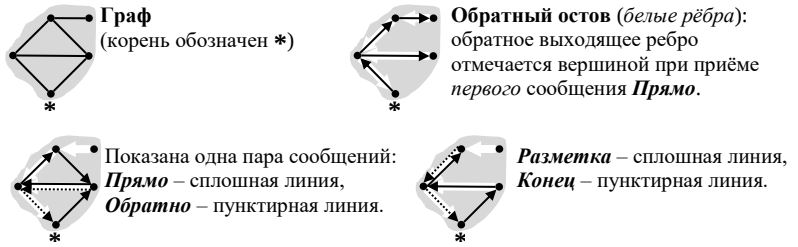


Рис. 8. Поиск максимального пути от корня и его разметка.
Fig. 8. Search for the maximum path from the root and its markup.

В корне хранится *vfr*-вектор P_{max}, вначале пустой. Когда корень получает сообщение ***Обратно*** с параметром P, он сравнивает длину P из сообщения с длиной P_{max}. Если P длиннее, то запоминается $P_{max} := P$.

Конец работы определяется с помощью универсальной процедуры стопора (раздел 6) для типов сообщений ***Прямо*** и ***Обратно***. Корень посылает вовне сообщение, содержащее как параметр $P_{max} = \langle v_1, f_1, r_1, v_2, f_2, r_2, \ldots v_{k_{max}} \rangle$.

Сообщение ***Прямо*** проходит от корня путь с хордой, длиной не более D_0+1, или путь, заканчивающийся в терминальной вершине, длиной не более D_0. Затем сообщение ***Обратно*** проходит путь до корня длиной не более D_0.

Число сообщений на ребре экспоненциально, что и следовало ожидать, поскольку задача поиска максимального пути – класса *NP*.

Оценка без учёта стопора. $M = \boldsymbol{O}(D_0 log n \Delta)$. $A = \boldsymbol{O}(D_0 log n \Delta)$. $F = \boldsymbol{O}(D_0 log n \Delta) = \boldsymbol{o}_{m \to \infty}[\boldsymbol{o}_{n \to \infty}]$. $T \le 2D_0$+1. N и E экспоненциальны.

Модификация 1. Число сообщений ***Обратно*** можно сделать линейным (а не экспоненциальным), если в каждой вершине хранить максимальную длину k_{max} векторов P из проходивших через вершину сообщений ***Обратно***. Полученное вершиной сообщение ***Обратно***, содержашее вектор P длины k, посылается дальше только, если $k > k_{max}$. Поскольку k меняется в диапазоне от 0 до D, число сообщений ***Обратно*** на одном ребре не превосходит D+1.

Модификация 2. Если нужно разметить найденный максимальный путь от корня, то после завершения поиска такого пути корень создаёт сообщение ***Разметка*** класса «Пересылка по запомненному маршруту», в котором параметр *Маршрут* равен $\langle f_1, f_2, \ldots f_{k_{max}-1} \rangle$ и имеет длину не более D_0. Когда вершина v_j, где j=1..k_{max}-1, посылает сообщение ***Разметка*** по ребру f_j, она отмечает выходящее ребро пути в переменной $Fn := f_j$ размером $\boldsymbol{O}(log \Delta)$. Когда вершина v_j, где j=2..k_{max}, принимает сообщение ***Разметка*** по ребру r_{j-1},

она отмечает входящее ребро пути в переменной $Rn := r_{j-1}$ размером $\boldsymbol{O}(log\Delta)$. Когда сообщение ***Разметка*** дойдёт до конца запомненного пути, т.е. до вершины $v_{k_{max}}$, эта вершина создаёт сообщение ***Конец*** класса «Рассылка по отмеченным рёбрам», которыми считаются рёбра обратного остова. Разметка заканчивается, когда корень получает сообщение ***Конец***.

Сообщение ***Разметка*** проходит путь от корня длиной не более D_0, а сообщение ***Конец*** возвращается в корень по пути длиной не более D_0.

Оценка без учёта поиска максимального пути. $M = \boldsymbol{O}(D_0 log n\Delta)$. $A = \boldsymbol{O}(D_0 log n\Delta)$. $F = \boldsymbol{O}(D_0 log n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. $T \le 2D_0$. $N \le 1$ и $E = \boldsymbol{O}(D_0 log n\Delta)$.

Модификация 3. Если в графе построен обратный остов, при получении первого сообщения ***Прямо*** не нужно отмечать выходящее обратное ребро. Выигрыш по времени будет, если остов – дерево кратчайших путей.

Оценка. $M = \boldsymbol{O}(D_0 log n\Delta)$. $A = \boldsymbol{O}(D_0 log n\Delta)$. $F = \boldsymbol{O}(D_0 log n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$. При поиске пути без учёта стопора $T \le D_0+d_0+1$. Для разметки $T \le D_0+d_0$.

Заметим, что такой же выигрыш по времени можно получить и в том случае, когда дерево кратчайших путей не построено, если сообщения ***Обратно*** и ***Конец*** имеют класс «Множественная рассылка». Однако в этом случае память автомата больше: $A = \boldsymbol{O}(n log n\Delta)$, $F = \boldsymbol{O}(n log n\Delta) = \boldsymbol{o}_{m\to\infty}[\boldsymbol{o}_{n\to\infty}]$.

10.2. Максимальный путь в графе

Идея поиска максимального пути в связном графе заключается в следующем. Рассмотрим все маршруты вида $A{\cdot}B{\cdot}\langle e\rangle$ или $A{\cdot}C$, где A – путь от корня до некоторой вершины v, B – путь от вершины v до вершины $v`$, $e$ – хорда пути $B$, инцидентная вершине $v`$, C – путь от вершины $v`$, заканчивающийся в терминальной вершине. Вершину v назовём *инициатором*. Максимальный путь в графе – это максимальный среди путей B и C для всех инициаторов.

Как и при поиске максимального пути от корня, используются два типа сообщения: ***Прямо*** класса «Рассылка до самопересечения» с *vfr*-накоплением и ***Обратно*** класса «Рассылка по отмеченным рёбрам», которыми считаются рёбра обратного остова (Рис. 9). Вершина содержит переменную *Было*.

Сообщение ***Прямо*** может создаваться не только в корне, но и в любой вершине-инициаторе v, поэтому *vfr*-вектор имеет вид $P_k{\cdot}\langle f_k\rangle$, где $P_k = \langle v_{j+1},f_{j+1},r_{j+1},v_{j+2},f_{j+2},r_{j+2},\ldots v_k\rangle$ и $v_{j+1} = v$. Вершина v становится инициатором, когда получает первое (*Было* = ***false***) сообщение ***Прямо*** с *vfr*-вектором $P_j{\cdot}\langle f_j\rangle$, где $P_j = \langle v_1,f_1,r_1,v_2,f_2,r_2,\ldots v_j\rangle$. Вершина v сначала рассылает дальше сообщение ***Прямо*** в соответствии с его классом, а потом создаёт новое сообщение ***Прямо***, которое рассылается по всем инцидентным рёбрам. Новое сообщение ***Прямо***, посылаемое по ребру f_{j+1}, содержит *vfr*-вектор $\langle v_{j+1}\rangle{\cdot}\langle f_{j+1}\rangle$.

Создание и обработка сообщения ***Обратно*** и определение конца работы такие же как в случае поиска максимального пути от корня.

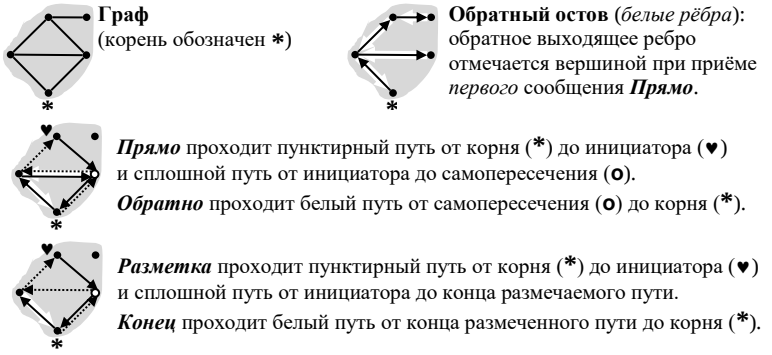


Рис. 9. Поиск максимального пути в графе и его разметка.
Fig. 9. Search for the maximum path in the graph and its markup.

Модификация 1 аналогична модификации 1 из подраздела 10.1. Число сообщений ***Обратно*** на ребре не больше D+1.

Модификация 2 для разметки найденного максимального пути. В сообщение ***Прямо*** добавляется ещё один параметр R, содержащий f-вектор пути от корня до инициатора. Когда инициатор – корень, R пусто. Сообщение ***Прямо*** с инициатором в некорневой вершине v создаётся этой вершиной при получении первого сообщения ***Прямо*** (его инициатор – корень), содержащего vfr-вектор $P_j \cdot \langle f_j \rangle$, где $P_j = \langle v_1,f_1,r_1,v_2,f_2,r_2,\ldots v_j \rangle$. Тогда $R = \langle f_1,\ldots f_j \rangle$. Сообщение ***Обратно*** также имеет дополнительный параметр R, переписываемый из сообщения ***Прямо*** при создании сообщения ***Обратно***.

Корень вместе с переменной $P_{max} = \langle v_{j+1},f_{j+1},r_{j+1},v_{j+2},f_{j+2},r_{j+2},\ldots v_{k_{max}} \rangle$ хранит соответствующую ей переменную $R_{max} = \langle f_1,f_2,\ldots f_j \rangle$, в которую из сообщения ***Обратно*** переписывается параметр R, когда параметр P переписывается в переменную P_{max}. Когда создаётся сообщение ***Разметка***, оно имеет два параметра: R_{max} и $P^f_{max} = \langle f_{j+1},f_{j+2},\ldots f_{k_{max}-1} \rangle$ как подпоследовательность P_{max}, являющаяся f-вектором найденного пути. Сначала сообщение ***Разметка*** двигается по пути с f-вектором R_{max} от корня до начала найденного пути, а потом – по этому пути с f-вектором P^f_{max}, размечая путь в переменных Rn и Fn.

Модификация 3 полностью аналогична модификации 3 в подразделе 10.1.

Оценка. По сравнению с оценками в подразделе 10.1 увеличиваются размеры сообщения M и автомата A за счёт того, что величина D_0 меняется на D: $M = A = F = \boldsymbol{O}(D log n \Delta) = \boldsymbol{o}_{m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$]. Если используется множественная рассылка, то память ещё больше: $A = F = \boldsymbol{O}(n log n \Delta) = \boldsymbol{o}_{m\to\infty}$ [$\boldsymbol{o}_{n\to\infty}$].

Время работы (без учёта стопора) слагается из времени X прохождения пути от корня до инициатора, времени Y прохождения максимального пути и времени Z прохождения пути до корня. При поиске пути $X \le d_0$, $Y \le D$ и $Z \le D_0$, а при разметке $X \le D_0$, $Y \le D$ и $Z \le D_0$. Если используется ранее построенное дерево кратчайших путей или множественная рассылка, то $Z \le d_0$.

При наличии дерева кратчайших путей можно при разметке сделать $X \le d_0$. Для этого надо, чтобы вершина становилась инициатором при получении не первого сообщения ***Прямо***, а того, которое прошло кратчайший путь по дереву от корня. Для этого сообщение ***Прямо*** имеет булевский признак *Кратчайший*, который равен ***true*** при создании сообщения в корне, а затем «сбрасывается» в ***false***, когда сообщение проходит не остовное ребро (что отмечено в его конце) или когда оно создаётся в некорневом инициаторе.

Если используется множественная рассылка для передачи сообщения ***Разметка*** от корня до инициатора, то при разметке также $X \le d_0$. В этом случае в сообщениях не нужен параметр R, описывающий путь от корня до инициатора. Инициатор сам «ловит» адресованное ему сообщение ***Разметка***, поскольку в нём есть *vfr*-вектор пути, начинающийся как раз в инициаторе.

В целом получаем: при поиске пути $T \le d_0+2D+1$, при разметке $T \le d_0+2D$. При использовании дерева кратчайших путей или множественной рассылки: при поиске пути $T \le 2d_0+D+1$, при разметке $T \le 2d_0+D$. При поиске пути N и E экспоненциальны, а при разметке $N \le 1$ и $E = \boldsymbol{O}(D_0 log n \Delta)$.

11. Заключение

В статье предложен общий подход к решению задач на неориентированном упорядоченном связном корневом графе коллективом автоматов, расположенных в вершинах графа и обменивающихся сообщениями по рёбрам графа. Автоматы – полуроботы, т.е. размер их памяти может расти вместе с ростом числа вершин n и числа рёбер m графа, но описание графа, вообще говоря, не помещается в память автомата. Выбрана модель максимального распараллеливания: время срабатывания автомата нулевое, а ёмкость ребра (число сообщений на нём) не ограничена. Это позволяет получать нижние оценки сложности алгоритмов решения задач.

Предложены базовые процедуры обработки сообщений, из которых строятся алгоритмы решения задач на графах. Это продемонстрировано на примерах построения остова графа, универсального «стопора», определяющего конец работы любого алгоритма, построения дерева кратчайших путей и нумерации графа. Также предложен алгоритм сбора в неограниченной памяти автомата корня полной информации о графе полуроботами некорневых вершин. Все эти алгоритмы имеют линейную сложность от n и m, и не более чем линейное число сообщений, одновременно передаваемых по ребру. Также рассмотрена задача поиска максимального пути в графе класса *NP*. За счёт неограниченного (экспоненциального) числа сообщений алгоритм решения этой задачи имеет линейную сложность.

Полученные результаты позволяют надеяться, что предложенная методика может успешно применяться для решения других задач на графах. Мы намереваемся в дальнейшем рассмотреть такие задачи, как построение минимального остовного дерева во взвешенном графе, построение

максимального независимого множества вершин, определение всех мостов в неориентированном графе и др. Также предполагается расширить предлагаемый подход на случай ориентированных графов, недетерминированных и динамических графов.

Список литературы

[1]. И. Бурдонов, А. Косачев. Обход неизвестного графа коллективом автоматов. Труды ИСП РАН, том 26, вып. 2, 2014, стр. 43-86. DOI: 10.15514/ISPRAS-2014-26(2)-2

[2]. И. Бурдонов, А. Косачев. Исследование графа взаимодействующими автоматами. «Вестник Томского государственного университета. Управление, вычислительная техника и информатика», №3, 2014, стр. 67-75.

[3]. И. Бурдонов, А. Косачев. Обход неизвестного графа коллективом автоматов. Недетерминированный случай. Труды ИСП, том 27, вып. 1, 2015, стр. 51-68. DOI: 10.15514/ISPRAS-2015-27(1)-4

[4]. И.Б. Бурдонов, А.С. Косачев., В.В. Кулямин. Исследование графа набором автоматов. "Программирование", 2015, №6, стр. 3-8.

[5]. И.Б. Бурдонов, А.С. Косачев. Исследование графов коллективом двигающихся автоматов. "Программная инженерия", 2016, том 7, № 12, стр. 559-567.

[6]. И.Б.Бурдонов, А.С. Косачев. Построение прямого и обратного остовов автоматами на графе. Труды ИСП РАН, том 26, вып. 6, 2014, стр. 57-62. DOI: 10.15514/ISPRAS-2014-26(6)-4

[7]. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Параллельные вычисления автоматами на прямом и обратном остовах графа. Труды ИСП РАН, том 26, вып. 6, 2014, стр. 63-66. DOI: 10.15514/ISPRAS-2014-26(6)-5

[8]. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. Параллельные вычисления на графе. "Программирование", 2015, №1, стр. 3-20.

[9]. И. Бурдонов, А. Косачев. Мониторинг динамически меняющегося графа. Труды ИСП РАН, том 27, вып. 1, 2015, стр. 69-96. DOI: 10.15514/ISPRAS-2015-27(1)-5

[10]. И. Бурдонов, А. Косачев. Параллельные вычисления на динамически меняющемся графе. Труды ИСП РАН, том 27(2), 2015, стр. 189-220. DOI: 10.15514/ISPRAS-2015-27(2)-12

[11]. И.Б. Бурдонов, А.С. Косачев. Исследование ориентированного графа коллективом неподвижных автоматов. "Программная инженерия", 2017, том 8, № 1, стр. 16-25.

[12]. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай. // Программирование. — 2004. — №1. — С. 2-17.

[13]. И. Бурдонов, А. Косачев. Размер памяти для хранения упорядоченного корневого графа. Труды ИСП РАН, том 29, вып. 2, 2017, стр. xx-xx. DOI: 10.15514/ISPRAS-2017-29(2)-1

[14]. И.Б. Бурдонов, А.С. Косачев. Исследование графа автоматом. "Программная инженерия", 2016, №11, стр. 498-508.

[15]. И.Б.Бурдонов. Обход неизвестного ориентированного графа конечным роботом. "Программирование", 2004, №4, стр.11-34.

[16]. И.Б.Бурдонов. Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом. "Программирование", 2004, №6, стр.6-29.

[17]. David Peleg. Distributed computing — A Locality-sensitive approach. SIAM Monographs on Discrete Mathematics and Applications. 2000, 359 pp.
[18]. Garey, Michael R.; Johnson, David S. (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, ISBN 0-7167-1045-5.

A general approach to solving problems on graphs by collective automata

I.B. Burdonov <igor@ispras.ru>
A.S. Kossatchev <kos@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. We propose a general method to solve graph problems by a set of automata (computational agents) located in vertices of undirected ordered connected rooted graph and communicating by passing messages along graph edges. The automata are semi-robots, i.e., their internal memory size is sufficient to store values depending on the number of vertices and edges of the graph (n and m, correspondingly) but is too small to store the complete graph description. Section 2 presents classification of graph-based distributed computational models depending on internal memory size of vertex automaton, vertex automaton operation time, and edge capacity (the number of messages that are passing along an edge simultaneously). We choose for further use the model of maximum parallelism, having zero automaton operation time and unbounded edge capacity. It allows to obtain lower complexity bounds on distributed graph problems. Section 3 describes algorithm complexity assessment rules. Section 4 presents basic message processing procedures and message classification according to paths passed by them and methods of message processing by vertex automaton. We propose to construct algorithms solving graph problems on the base of the procedures considered, and present some examples in further sections. Sections 5-9 describe distributed algorithms for spanning tree construction, for task termination detection (based on detection of absence of messages used by the task), for shortest paths tree construction, for graph vertices enumeration, for collecting graph structure information in the root automaton memory (if it is unbounded). The algorithms proposed has linear time complexity in n and m, and use linear in n and m number of messages. Section 10 considers construction of maximum weight path in a weighted graph, which is the NP problem. We propose the algorithm that for the sake of using unbounded number of messages can solve this problem in linear time in n and m. The conclusion summarizes the paper and draws directions of further research: constructing algorithms for other graph problems and generalization of the approach to directed, non-deterministic and dynamic graphs.

Keywords: undirected graphs; graph problems; asynchronous distributed systems; distributed algorithms.

DOI: 10.15514/ISPRAS-2017-29(2)-2

For citation: Burdonov I.B., Kossatchev A.S. A general approach to solving problems on graphs by collective automata. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. 27-76. DOI: 10.15514/ISPRAS-2017-29(2)-2

References

[1]. I. Burdonov, A. Kosachev. Graph learning by a set of automata. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 2, 2014, pp. 43-86 (in Russian). DOI: 10.15514/ISPRAS-2014-26(2)-2

[2]. I. Burdonov, A. Kosachev. Analysis of a Graph by a Set of Interacting Automata. «Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naja tehnika i informatika» [Tomsk State University. Journal of Control and Computer Science], №3, 2014, pp. 67-75. (in Russian).

[3]. I. Burdonov, A. Kosachev. Graph learning by a set of automata. The nondeterministic case. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 1, 2015, pp. 51-68, (in Russian). DOI: 10.15514/ISPRAS-2015-27(1)-4

[4]. I.B. Bourdonov, A.S. Kossatchev, V.V. Kulyamin. Analysis of a Graph by a Set of Automata. Programming and Computer Software, Vol. 41, No. 6, 2015, pp. 307-310. DOI: 10.1134/S0361768815060031

[5]. I.B. Burdonov, A.S. Kosachev. Analysis of a Graph by a Set of Moving Automata. "Programmnaja inzhenerija" [Software Engineering], 2016, Vol. 7, № 12, pp. 559-567, (in Russian).

[6]. I.B.Burdonov, A.S. Kosachev. Building direct and back spanning trees by automata on a graph. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 6, 2014, pp. 57-62, (in Russian). DOI: 10.15514/ISPRAS-2014-26(6)-4

[7]. I.B.Burdonov, A.S.Kosachev, V.V.Kuljamin. Parallel calculations by automata on direct and back spanning trees of a graph. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 6, 2014, pp. 63-66 (in Russian). DOI: 10.15514/ISPRAS-2014-26(6)-5

[8]. I.B. Bourdonov, A.S. Kossatchev, V.V. Kulyamin. Parallel Computations on a Graph. Programming and Computer Software, Vol. 41, No. 1, 2015, pp. 1-13. DOI: 10.1134/S0361768815010028

[9]. I. Burdonov, A. Kosachev. Monitoring of dynamically changed graph. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 1, 2015, pp. 69-96 (in Russian). DOI: 10.15514/ISPRAS-2015-27(1)-5

[10]. I. Burdonov, A. Kosachev. Parallel Calculations on Dynamic Graph. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 2, 2015, pp. 189-220 (in Russian). DOI: 10.15514/ISPRAS-2015-27(2)-12

[11]. I.B. Burdonov, A.S. Kosachev. Analysis of an Oriented Graph by a Set of Motionless Automata. "Programmnaja inzhenerija" [Software Engineering], 2017, Vol. 8, № 1, pp. 16-25, (in Russian).

[12]. I. B. Bourdonov, A.S. Kossatchev, V V. Kuliamin. Irredundant Algorithms for Traversing Directed Graphs: The Nondeterministic Case. Programming and Computer Software, Vol. 30, No. 1, 2004, pp. 2-17. DOI: 10.1023/A:1025733107700

[13]. I. Burdonov, A. Kosachev. Size of the memory for storage of ordered rooted graph. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. xx-xx (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-1

[14]. I.B. Burdonov, A.S. KosachevAnalysis of a Graph by an Automaton. "Programmnaja inzhenerija" [Software Engineering], 2016, №11, pp. 498-508, (in Russian).

[15]. I.B. Bourdonov. Traversal of an Unknown Directed Graph by a Finite Robot. Programming and Computer Software, Vol. 30, No. 4, 2004, pp. 188-203. DOI: 10.1023/B:PACS.0000036417.58183.64

[16]. I. B. Bourdonov. Backtracking Problem in the Traversal of an Unknown Directed Graph by a Finite Robot. Programming and Computer Software, Vol. 30, No. 4, 2004, pp. 305-322. DOI: 10.1023/B:PACS.0000049509.66710.3a

[17]. David Peleg. Distributed computing — A Locality-sensitive approach. SIAM Monographs on Discrete Mathematics and Applications. 2000, 359 pp.

[18]. Garey, Michael R.; Johnson, David S. (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, ISBN 0-7167-1045-5.

Развитие ядра операционной системы Linux[1]

Е.М. Новиков <novikov@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Существующие исследования, которые посвящены анализу развития ядра операционной системы Linux, рассматривают ядро вместе с поставляемыми с ним загружаемыми модулями или некоторые конкретные подсистемы ядра. Целью данной работы является оценка развития ядра без загружаемых модулей, для чего предлагается метод определения границы между ними. Оценка развития дается для всех версий ядра операционной системы Linux, выпущенных за последние 7,5 лет. Также приводятся результаты классификации и распределение типовых ошибок, исправленных в ядре, на основе анализа изменений, которые были сделаны в стабильных ветках ядра за последние 2 месяца 2015 года. Полученные результаты могут быть использованы при оценке актуальности и применимости различных методов и инструментов обеспечения качества программных систем.

Ключевые слова: операционная система; монолитное ядро; качество программной системы; анализ изменений.

DOI: 10.15514/ISPRAS-2017-29(2)-3

Для цитирования: Новиков Е.М. Развитие ядра операционной системы Linux. Труды ИСП РАН, том 29, вып. 2, 2017 г., стр. 77-96. DOI: 10.15514/ISPRAS-2017-29(2)-3

1. Введение

В настоящее время ядро операционной системы (далее – ОС) Linux используется повсеместно огромным количеством пользователей на большом количестве аппаратных платформ от встроенных систем до суперкомпьютеров. По своей архитектуре ядро ОС Linux является монолитным – оно целиком загружается в оперативную память во время старта ОС и затем работает в одном адресном пространстве [1]. Как правило, в ядре реализуется основная функциональность, например, планирование процессорного времени, управление памятью и межпроцессным взаимодействием, обработка прерываний, поддержка разнообразных

[1] Исследование выполнено при финансовой поддержке РФФИ, проект «Инкрементальная статическая верификация подсистем монолитного ядра операционных систем» № 16-31-60097.

примитивов синхронизации. Изменить набор функций ядра можно путем динамической загрузки модулей или его статической перекомпиляции с другой конфигурацией и/или для другой целевой архитектуры. В обоих случаях модули становятся частью монолитного ядра.

Для ядра ОС Linux в виде загружаемых модулей могут быть собраны большинство драйверов устройств, файловых систем, сетевых протоколов, аудиокодеков и т.д. Необходимость в конкретных загружаемых модулях определяется аппаратным обеспечением, а также потребностями конкретных пользователей. На типовых персональных компьютерах может быть загружено несколько десятков или сотен модулей. Общее количество загружаемых модулей, которые поставляются вместе с ядром ОС Linux последних версий, составляет порядка 5 тысяч. В данной работе будет показано, что их совокупный размер во много раз превосходит размер ядра без загружаемых модулей.

Для оценки различных аспектов развития ядра ОС Linux постоянно проводятся исследования [2-7]. В данных исследованиях описывается, что происходило на протяжении нескольких лет в прошлом, и отмечается актуальность задач, которые предстоит решить в будущем. Анализируются общие аспекты разработки, такие как время выхода новых версий, размер кода, изменения между различными версиями и авторство данных изменений. В большинстве работ основное внимание уделяется качеству ядра ОС Linux. Изучаются как те ошибки, которые уже были обнаружены и исправлены до публикации соответствующей статьи, так и те ошибки, которые находили инструменты во время ее подготовки.

Насколько известно автору, до сих пор не было предпринято ни одной попытки оценить развитие ядра ОС Linux без загружаемых модулей – рассматривается ядро вместе с поставляемыми с ним загружаемыми модулями или некоторые конкретные подсистемы ядра, например файловые системы (если не оговорено иное, то далее в работе ядро без загружаемых модулей будет называться ядром, как это принято в англоязычной литературе). Это оправдано, например, с точки зрения критичности ошибок, поскольку архитектура ядра является монолитной, а значит, любая ошибка или сбой в одном из загруженных модулей может привести к серьезным последствиям для ядра, других загруженных модулей и пользовательских приложений. Однако на большинстве систем используется относительно малая часть загружаемых модулей. Как следствие, ядро ОС Linux является наиболее критичной частью ядра в совокупности со всеми загружаемыми модулями – ошибки в нем вероятнее всего затронут наибольшее количество пользователей, как на уровне ядра и различных загружаемых модулей, так и на уровне пользовательских приложений.

В данной работе предлагается метод определения границы между ядром и загружаемыми модулями, позволяющий оценить развитие ядра ОС Linux. В разделе 2 приведены общие сведения о развитии ядра вместе с поставляемыми с ним загружаемыми модулями за последние 7,5 лет с целью последующей

оценки развития ядра ОС Linux. Раздел 3 описывает предлагаемый метод и оценивает развитие ядра за последние 7,5 лет. В разделе 4 приведены результаты классификации и распределение типовых ошибок, исправленных в ядре ОС Linux, на основе анализа изменений, которые были сделаны в стабильных ветках ядра за последние 2 месяца 2015 года. В заключении делаются выводы на основе проведенного исследования.

В качестве базы для анализа развития ядра ОС Linux был выбран официальный Git-репозиторий[1], в котором собраны все версии, изменения и стабильные ветки для оригинального ядра и поставляемых с ним загружаемых модулей, начиная с версии 2.6.12, выпущенной 17 июня 2005 года. В рамках данной работы не рассматриваются ни сторонние модификации ядра, например, поддержка выполнения в реальном времени, ни сторонние загружаемые модули, такие как проприетарные драйверы видеокарт.

2. Развитие ядра ОС Linux вместе с поставляемыми с ним загружаемыми модулями за последние 7,5 лет

В табл. 1 указаны общие сведения для всех версий ядра, начиная с 2.6.30 и заканчивая 4.8 – последняя версия ядра на момент написания данной статьи (всего 39 версий). В среднем на разработку одной версии уходил 71[2] день (около 10 недель). Каждая версия в среднем включала 12 тысяч изменений в ядре и загружаемых модулях, что соответствует в среднем 7,5 изменениям в час.

Табл. 1. Общие сведения о развитии ядра ОС Linux вместе с поставляемыми с ним загружаемыми модулями за последние 7,5 лет
Table 1. General information about the development of the Linux kernel, along with the loadable modules for the last 7.5 years

Версия ядра	Дата выпуска	Количество дней на разработку	Количество изменений, тысяч	Среднее количество изменений в час
4.8	2 октября 2016	70	15[3]	8,7
4.7	24 июля 2016	70	13	8,0
4.6	15 мая 2016	63	15	9,7
4.5	13 марта 2016	63	13	8,7
4.4	10 января 2016	70	14	8,4
4.3	1 ноября 2015	63	13	8,8
4.2	30 августа 2015	70	15	8,8
4.1	21 июня 2015	70	13	7,7
4.0	12 апреля 2015	63	11	7,5

[1] git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git.
[2] Здесь и далее выполнено округление до 2 значащих цифр.
[3] Количество изменений в данной ячейке посчитано с помощью команды «git rev-list --count v4.7..v4.8». В других ячейках данного столбца – аналогично для соответствующих версий.

3.19	8 февраля 2015	63	14	9,0
3.18	7 декабря 2014	63	12	8,2
3.17	5 октября 2014	63	13	8,8
3.16	3 августа 2014	56	14	10
3.15	8 июня 2014	70	15	8,9
3.14	30 марта 2014	70	13	8,0
3.13	19 января 2014	77	13	7,2
3.12	3 ноября 2013	62	12	8,0
3.11	2 сентября 2013	64	12	7,7
3.10	30 июня 2013	63	15	9,7
3.9	28 апреля 2013	69	13	7,8
3.8	18 февраля 2013	70	14	8,1
3.7	10 декабря 2012	71	13	7,6
3.6	30 сентября 2012	71	11	6,5
3.5	21 июля 2012	62	12	7,9
3.4	20 мая 2012	63	12	7,8
3.3	18 марта 2012	74	11	6,4
3.2	4 января 2012	72	13	7,3
3.1	24 октября 2011	95	9,4	4,1
3.0	21 июля 2011	64	9,8	6,4
2.6.39	18 мая 2011	65	11	7,1
2.6.38	14 марта 2011	69	10	6,3
2.6.37	4 января 2011	76	12	6,7
2.6.36	20 октября 2010	80	10	5,3
2.6.35	1 августа 2010	77	10	5,7
2.6.34	16 мая 2010	81	10	5,2
2.6.33	24 февраля 2010	84	12	5,8
2.6.32	2 декабря 2009	84	12	5,9
2.6.31	9 сентября 2009	92	12	5,3
2.6.30	9 июня 2009	78	13	6,9

Для всех версий ядра ОС Linux, выпущенных за последние 7,5 лет, с помощью утилиты cloc 1.70[1] для ядра вместе с поставляемыми с ним загружаемыми модулями было посчитано количество файлов (рис. 1) и количество строк кода (рис. 2), включая комментарии и пустые строки, на различных языках программирования. Общее количество файлов увеличилось с 25 тысяч до 45 тысяч – на 83%, а количество строк кода – с 10 миллионов до 20 миллионов – на 95%, что в среднем соответствует приросту на 7,5 файлов и 3,6 тысяч строк кода в день.

Графики наглядно демонстрируют, что основной вклад как в количество файлов, так и в количество строк кода вносили файлы на языке программирования Си: в среднем 33 тысячи (91%) и 15 миллионов (97%) соответственно. Также за счет них происходил и основной рост ядра ОС Linux

[1] https://github.com/AlDanial/cloc.

– на 91% и на 97% соответственно. Файлы на языке ассемблера в среднем вносили вклад 1,3 тысячи файлов (4%) и 0,36 миллионов строк кода (2%). Их вклад в рост – 3% и 2% соответственно. Файлы на других языках программирования, в основном сценарных, описывающих сборку и представляющих различные вспомогательные утилиты, в среднем вносили вклад 2 тысячи файлов (5%) и 0,14 миллионов строк кода (1%) соответственно. Их влияние на рост составило 6% и 1% соответственно.

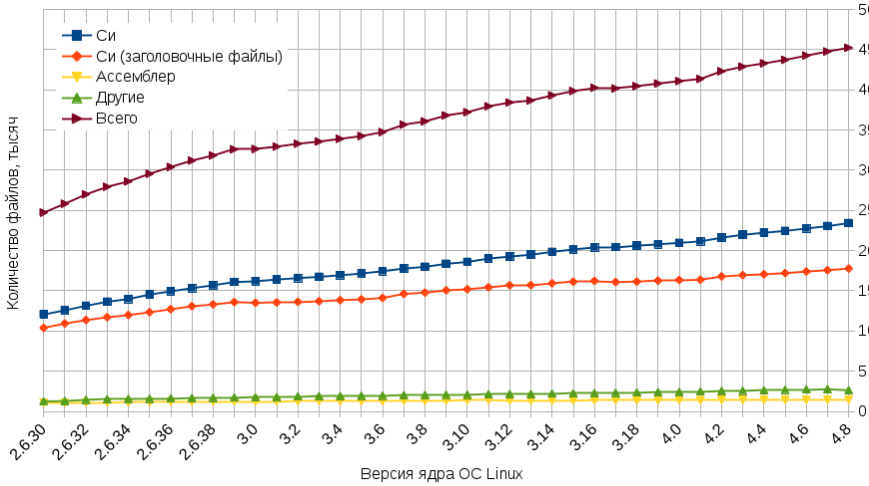


Рис. 1. Количество файлов на различных языках программирования в ядре ОС Linux вместе с поставляемыми с ним загружаемыми модулями
Fig. 1. The number of files in different programming languages in the Linux kernel, along with the loadable modules

3. Развитие ядра ОС Linux за последние 7,5 лет

В данном разделе предлагается метод, позволяющий оценить развитие ядра ОС Linux. Представляются результаты данной оценки с разбиением по основным подсистемам и без него для всех версий ядра, которые были выпущены за последние 7,5 лет.

3.1 Определение границы между ядром ОС Linux и загружаемыми модулями

Основной сложностью при оценке развития ядра ОС Linux является определение границы между ядром и загружаемыми модулями. Данная граница зависит от версии ядра, а также от целевой архитектуры и конфигурационных опций, с которыми собирается ядро.

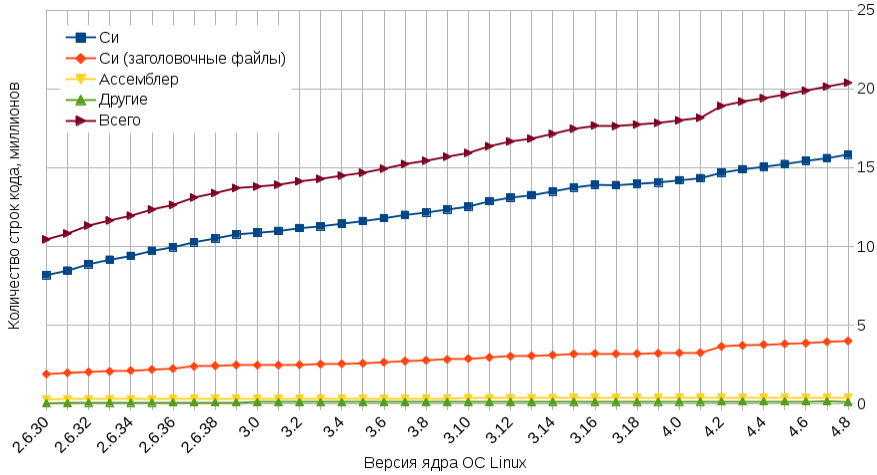


Рис. 2. Количество строк кода на различных языках программирования в ядре ОС Linux вместе с поставляемыми с ним загружаемыми модулями
Fig. 2. Number of lines of code in different programming languages in the Linux kernel, along with the loadable modules

Проводить границу для каждой исследуемой версии ядра предлагается независимо, поскольку между ними могут быть достаточно большие различия, примеры чего приведены в предыдущем разделе.

Для конфигурирования предлагается использовать набор опций, получаемых для стандартной конфигурационной цели *allmodconfig*. В этом случае все, что можно собрать в виде загружаемых модулей, собирается в таком виде. При использовании другого набора опций можно получить другое подмножество исходного кода ядра, например, можно отключить или включить поддержку определенной функциональности, использовать альтернативные реализации или даже включить некоторые загружаемые модули в состав ядра.

Проводить границу предлагается по файлам, а не по строкам кода. Это позволяет включить в ядро хотя бы частично исходный код альтернативных реализаций (активируемых при выборе соответствующих конфигурационных опций), который помещен в файлы с исходным кодом основных реализаций. При этом в состав ядра не будет включен исходный код альтернативных реализаций, который полностью помещен в отдельные файлы. Например, это относится к различным механизмам распределения памяти. Для *allmodconfig* в состав ядра включается файл *mm/slub.c*, соответствующий механизму распределения памяти *SLUB*, но не включаются файлы *mm/slab.c* и *mm/slob.c*, соответствующие механизмам распределения памяти *SLAB* и *SLOB*[1]. Также

[1] Подробнее о различных механизмах распределения памяти в ядре ОС Linux 1)2)можно узнать, например, из слайдов следующей презентации:
http://events.linuxfoundation.org/sites/events/files/slides/slaballocators.pdf.

благодаря такому способу проведения границы при подсчете количества строк кода включаются комментарии и пустые строки, которые традиционно учитываются при оценке развития ядра вместе с загружаемыми модулями.

Предлагается выбирать все файлы с исходным кодом на языке программирования Си и языке ассемблера, которые после компиляции и компоновки включаются в состав файла *vmlinux.o*, представляющего образ ядра ОС Linux без загружаемых модулей. Заголовочные файлы на языке программирования Си, а также файлы на других языках программирования предлагается не учитывать. Заголовочные файлы преимущественно используются для описания программного интерфейса, а не его реализации. Более того, заголовочные файлы включаются как в файлы с исходным кодом ядра, так и в файлы с исходным кодом загружаемых модулей. Аналогично файлы на всех других языках программирования достаточно сложно отнести к ядру и/или загружаемым модулям. Однако в предыдущем разделе показано, что их совокупный вклад в общее количество строк кода незначительный, около 1%, поэтому допустимо исключить их при сравнении.

Собирать ядро ОС Linux предлагается для архитектуры *x86_64*. В следующем подразделе будет показано, что это является наиболее существенным недостатком предложенного метода определения границы между ядром и загружаемыми модулями.

Результаты оценки развития ядра ОС Linux за последние 7,5 лет с использованием предложенного метода представлены в следующих двух подразделах. Данный метод также был использован при определении границы ядра для изменений в стабильных ветках ядра, однако для этого было сделано дополнительное предположение (раздел 4).

3.2 Развитие ядра ОС Linux

На рис. 3 и рис. 4 для ядра ОС Linux представлены количество файлов и количество строк кода, включая комментарии и пустые строки, на языке программирования Си и языке ассемблера для всех версий ядра, которые были выпущены за последние 7,5 лет.

Общее количество файлов увеличилось с 1,1 тысячи до 2,2 тысяч – на 90%, а количество строк кода – с 0,7 миллионов до 1,4 миллионов – на 105%. Это показывает, что ядро ОС Linux развивалось немного интенсивнее ядра вместе с загружаемыми модулями.

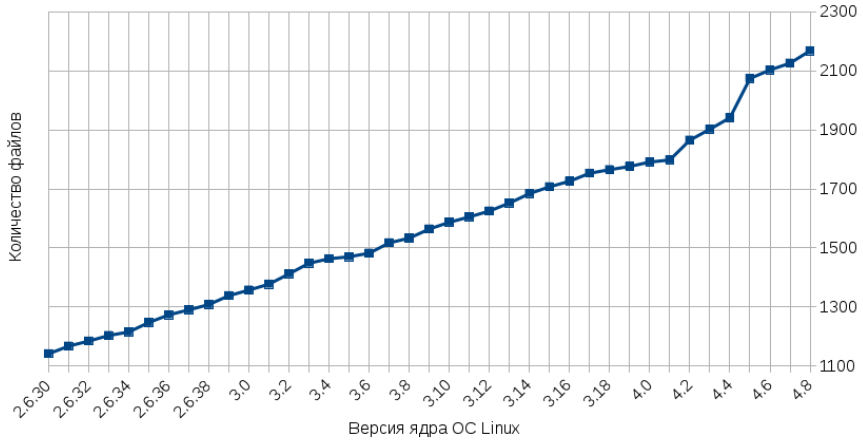
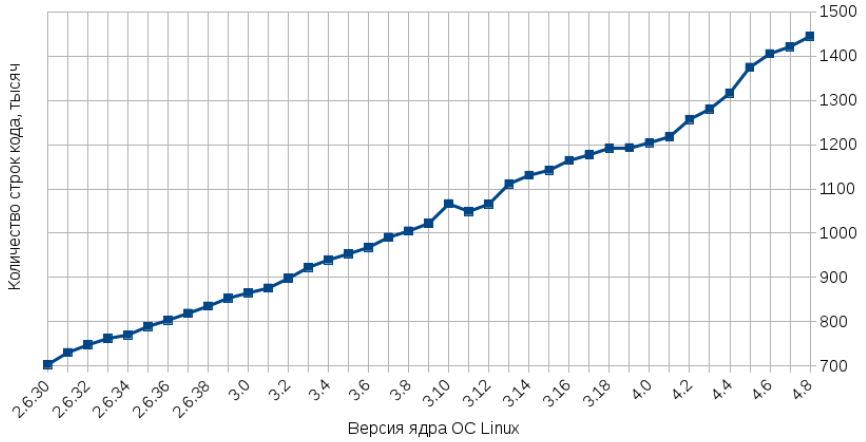


Рис. 3. Количество файлов на языке программирования Си и языке ассемблера в ядре ОС Linux

Fig. 3. The number of files in the C programming language and the assembler language in



the Linux kernel

Рис. 4. Количество строк кода на языке программирования Си и языке ассемблера в ядре ОС Linux

Fig. 4. Number of lines of code in C programming language and assembler language in Linux kernel

Среднее количество файлов на языке программирования Си для ядра ОС Linux составило 1,5 тысяч (97%), на языке ассемблера – 43 (3%). Среднее количество строк кода – 1 миллион (99%) и 6,3 тысячи (1%) соответственно. Таким образом, при использовании предложенного метода определения

границы ядра ОС Linux получилось, что количество файлов и количество строк кода на языке ассемблера в ядре по сравнению с ядром вместе с загружаемыми модулями меньше в 30 и 58 раз соответственно, в то время как для языка программирования Си данные соотношения составляют 21 и 15. Однако следует учесть, что при определении границы ядра не учитывался исходный код для других архитектур, кроме *x86_64*, притом, что файлы на языке ассемблера достаточно редко используются в загружаемых модулях – большинство из них попадает в ядро для соответствующей архитектуры и/или конфигурации.

В среднем по количеству файлов и строк кода на языке программирования Си и языке ассемблера ядро ОС Linux составило по 8,4% относительно ядра вместе с загружаемыми модулями. Таким образом, по размеру загружаемые модули, которые поставляются вместе с ядром, в 11 раз больше ядра.

3.3 Развитие основных подсистем ядра ОС Linux

Помимо оценки развития ядра ОС Linux в целом также полезно оценить развитие его основных подсистем. Основными подсистемами ядра в данной работе считаются следующие 12 подсистем в соответствии со структурой директорий с исходным кодом[1]:

- *arch* – аппаратные платформы;
- *block* – блочный уровень;
- *crypto* – криптографический интерфейс;
- *drivers* – поддержка различных типов устройств;
- *fs* – поддержка файловых систем;
- *init* – общие настройки ядра;
- *ipc* – межпроцессное взаимодействие;
- *kernel* – основная часть ядра;
- *lib* – вспомогательные библиотеки;
- *mm* – управление памятью;
- *net* – сеть;
- *security* – модели безопасности.

Единственной основной подсистемой ядра ОС Linux, в которой содержится достаточно много исходного кода на языке ассемблера, является *arch*. Поскольку в предыдущем подразделе было показано, что в ядре файлов и строк кода на языке ассемблера всего 3% и 1% соответственно, далее они не рассматриваются отдельно от файлов и строк кода на языке программирования Си.

[1] Подсистемы *certs*, *firmware* и *usr* не рассматриваются, поскольку они содержат менее 250 строк кода на языке ассемблера для всех версий ядра ОС Linux.

Аналогично рис. 3 и рис. 4 на рис. 5 и рис. 6 представлены количество файлов и количество строк кода, включая комментарии и пустые строки, на языке программирования Си и языке ассемблера для всех основных подсистем и версий ядра ОС Linux, которые были выпущены за последние 7,5 лет. На графиках не представлены основные подсистемы *init* и *ipc*, поскольку они включают до 13 файлов и до 8,9 тысяч строк кода на языке программирования Си.

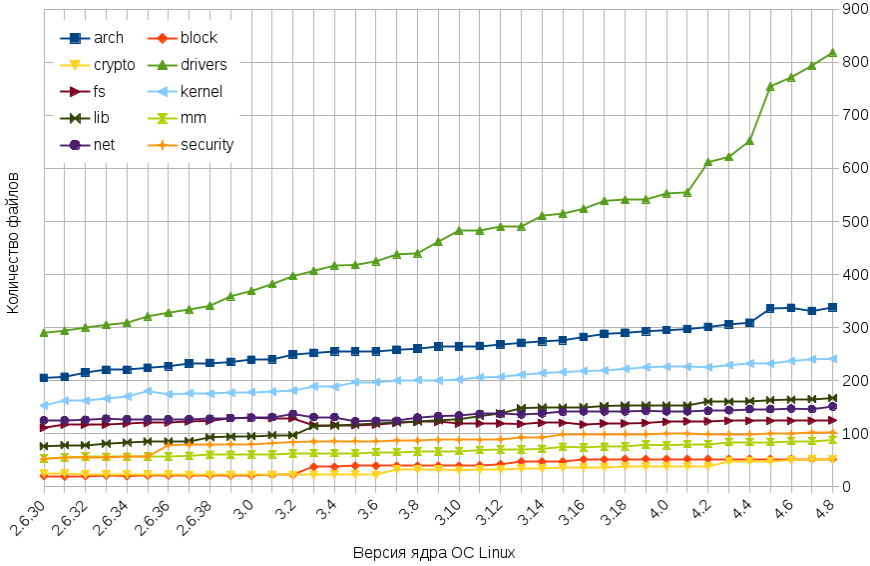


Рис. 5. Количество файлов на языке программирования Си и языке ассемблера в основных подсистемах ядра ОС Linux
Fig. 5. The number of files in the programming language C and assembly language in the core subsystems of the Linux kernel

Графики показывают, что основной вклад в ядро ОС Linux вносила основная подсистема *drivers*. Также преимущественно за счет нее происходил рост ядра. По количеству файлов лидерами также являлись основные подсистемы *arch* и *kernel*, по количеству строк кода – *kernel*, *arch* и *net*.

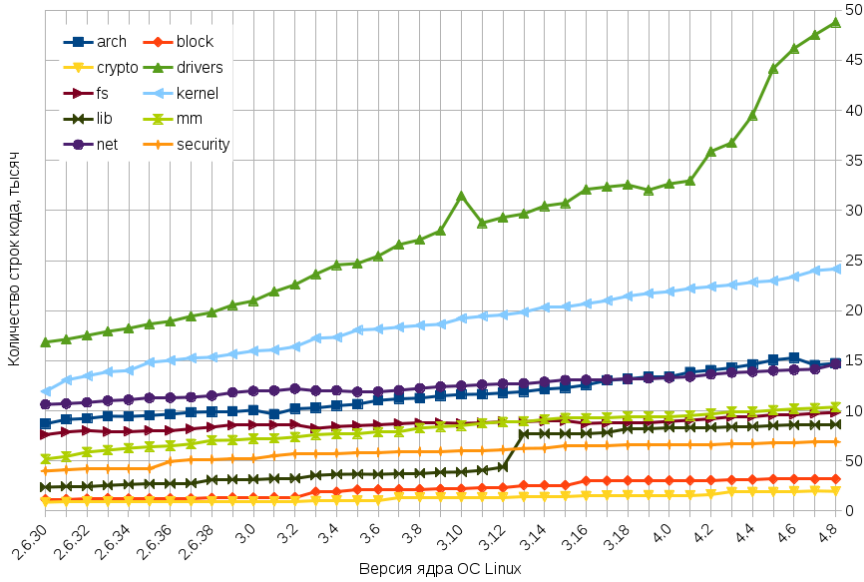


Рис. 6. Количество строк кода на языке программирования Си и языке ассемблера в основных подсистемах ядра ОС Linux

Fig. 6. Number of lines of code in the C programming language and assembler language in the core subsystems of the Linux kernel

4. Классификация и распределение типовых ошибок, исправленных в ядре ОС Linux

В соответствии с доработанной методикой выявления и классификации типовых ошибок [4] в данной работе был проведен анализ изменений, сделанных в стабильных ветках ядра за последние 2 месяца 2015 года.

Оказалось, что в указанный период времени изменения были сделаны только в 11 из 58 стабильных веток ядра. Это объясняется тем, что большая часть стабильных веток ядра более не поддерживается, а также тем, что несколько новых версий ядра еще не были выпущены, а значит, не было создано соответствующих стабильных веток ядра.

Вообще говоря, для каждого изменения может быть своя граница ядра ОС Linux. Для того чтобы не вычислять ее 3210 раз (общее количество изменений в стабильных ветках ядра ОС Linux за последние 2 месяца 2015 года), что требует достаточно много времени, было использовано дополнительное предположение, что для каждой стабильной ветки ядра граница не изменяется. Это предположение оправдано, поскольку в стабильных ветках ядра достаточно редко делают существенные изменения, в том числе переименования, перемещения и удаления файлов. Соответственно, для каждой стабильной ветки граница ядра ОС Linux определялась только для

последнего изменения в ней, а затем использовалась для всех остальных изменений в данной ветке.

Табл. 2. Количество изменений с уникальными заголовками в стабильных ветках ядра ОС Linux
Table 2. Number of changes with unique headers in stable Linux kernel branches

Название стабильной ветки ядра ОС Linux	Общее количество изменений	Количество изменений в ядре ОС Linux	Количество изменений с уникальными заголовками
linux-4.3.y	201	31 (15%)	31 (100%)
linux-4.2.y	647	121 (19%)	94 (78%)
linux-4.1.y	510	98 (19%)	10 (10%)
linux-3.18.y	370	81 (22%)	31 (38%)
linux-3.16.y	412	70 (17%)	20 (29%)
linux-3.14.y	266	60 (23%)	12 (20%)
linux-3.12.y	212	43 (20%)	7 (16%)
linux-3.10.y	184	42 (23%)	2 (5%)
linux-3.4.y	67	11 (16%)	9 (82%)
linux-3.2.y	300	70 (23%)	16 (23%)
linux-2.6.32.y	41	22 (54%)	1 (5%)
Всего	**3210**	**649 (20%)**	**233 (36%)**

Некоторые изменения, которые сделаны в нескольких стабильных ветках, могут быть достаточно сильно похожи. Например, к таким изменениям относятся исправления одних и тех же ошибок. Сами изменения могут отличаться, однако предполагается, что их заголовки одни и те же. В табл. 2 представлено количество изменений с уникальными заголовками в стабильных ветках ядра ОС Linux за последние 2 месяца 2015 года. В среднем таких изменений было 36%, что намного меньше, чем в драйверах – 85% [4]. Это показывает то, что для ядра существенно больше изменений переносится в различные стабильные ветки ядра, что косвенно служит показателем большей значимости ядра, в том числе с точки зрения исправления ошибок.

При построении табл. 3 все изменения с уникальными заголовками были автоматически отфильтрованы с использованием доработанных подходов из [4]. Затем 170 изменений, в которых исправляются типовые ошибки, были проанализированы вручную. Результаты данного анализа представлены в табл. 4.

Табл. 3. Количество изменений, в которых исправляются типовые и нетиповые ошибки, а также в которых не исправляются ошибки, в стабильных ветках ядра ОС Linux (отфильтровано автоматически)
Table 3. The number of changes that fix typical and non-standard errors, as well as in which errors are not fixed, in stable Linux kernel branches (automatically filtered)

Название стабильной ветки ядра ОС Linux	Количество изменений, в которых исправляются типовые ошибки	Количество изменений, в которых исправляются нетиповые ошибки	Количество изменений, в которых не исправляются ошибки
linux-4.3.y	21 (68%)	7 (23%)	3 (10%)
linux-4.2.y	70 (74%)	21 (22%)	3 (3%)
linux-4.1.y	9 (90%)	1 (10%)	–
linux-3.18.y	21 (68%)	6 (19%)	4 (13%)
linux-3.16.y	18 (90%)	1 (5%)	1 (5%)
linux-3.14.y	9 (75%)	2 (17%)	1 (8%)
linux-3.12.y	3 (43%)	1 (14%)	3 (43%)
linux-3.10.y	2 (100%)	–	–
linux-3.4.y	6 (67%)	3 (33%)	–
linux-3.2.y	11 (69%)	3 (19%)	2 (13%)
linux-2.6.32.y	–	–	1 (100%)
Всего	**170 (73%)**	**45 (19%)**	**18 (8%)**

Табл. 4. Количество изменений, в которых исправляются типовые и нетиповые ошибки, а также в которых не исправляются ошибки, в стабильных ветках ядра ОС Linux (проанализировано вручную)
Table 4. The number of changes that fix typical and non-standard errors, as well as in which errors are not fixed, in stable Linux kernel branches (analyzed manually)

Название стабильной ветки ядра ОС Linux	Количество изменений, в которых исправляются типовые ошибки	Количество изменений, в которых исправляются нетиповые ошибки	Количество изменений, в которых не исправляются ошибки
linux-4.3.y	12 (57%)	9 (43%)	–
linux-4.2.y	42 (60%)	24 (34%)	4 (6%)
linux-4.1.y	3 (33%)	3 (33%)	3 (33%)
linux-3.18.y	15 (71%)	5 (24%)	1 (5%)
linux-3.16.y	5 (28%)	13 (72%)	–
linux-3.14.y	6 (67%)	3 (33%)	–
linux-3.12.y	–	3 (100%)	–
linux-3.10.y	–	2 (100%)	–

linux-3.4.y	3 (50%)	3 (50%)	–
linux-3.2.y	7 (64%)	4 (36%)	–
linux-2.6.32.y	–	–	–
Всего	**93 (55%)**	**69 (41%)**	**8 (5%)**

В итоге получилось, что среди всех 233 изменений с уникальными заголовками в стабильных ветках ядра в ядре ОС Linux за последние 2 месяца 2015 года:

- 26 (11%) изменений соответствуют расширению функциональности;
- 114 (49%) – исправлению нетиповых ошибок;
- 93 (40%) – исправлению типовых ошибок.

По сравнению с драйверами [4] количество изменений, в которых:

- расширяется функциональность намного меньше: 11% вместо 21%;
- исправляются нетиповые ошибки – практически столько же: 49% вместо 52%;
- исправляются типовые ошибки – намного больше: 40% вместо 27%.

Таким образом, по сравнению с драйверами для ядра ОС Linux в стабильных ветках ядра исправляется больше ошибок, причем среди них больше типовых ошибок.

Все изменения, в которых исправлялись типовые ошибки, были классифицированы на основе методики из [4]. Предложенная в [4] классификация была изменена и дополнена. Класс *specific* был переименован в *linux* для того, чтобы подчеркнуть, что данные ошибки являются специфичными для ядра ОС Linux. Были изменены несколько существующих подклассов:

- *generic:buffer_overflow* – на *generic:buf overflow*;
- *generic:resource* – на *generic:memory*;
- *generic:null_ptr_deref* – на *generic:null ptr dereference*;
- *generic:int_overflow* – на *generic:int*, дополнительно к этому подклассу помимо переполнений целых чисел были отнесены такие ошибки, как преобразования знаковых целых чисел к беззнаковым;
- *specific:check_params* – на *specific:param*;
- *specific:resource* – на *specific:res*;
- *specific:check_ret_val* – на *specific:ret*;
- *specific:lock* – на *specific:one thread*.

Также было добавлено несколько новых подклассов:

- *generic:incorrect read/write* – общие ошибки чтения/записи в том случае, если конкретный подкласс ошибки, например,

generic:buf overflow или *generic:null ptr dereference*, не удается точно определить;

- *generic:infinite loop* – по сути данные типовые ошибки представляют собой непреднамеренное зависание программы;
- *generic:invalid cast* – некорректное преобразование переменных к типу другого размера;
- *generic:info leak* – ошибки данного подкласса происходят, когда значения в памяти не затираются и могут быть прочитаны кем-то, кто не имеет на это прав;
- *generic:always true/false condition* – использование условия, которое истинно или ложно на всех возможных путях выполнения;
- *generic:dead code* – недостижимый код;
- *generic:err ptr dereference* – некорректное разыменование указателей из определенного диапазона;
- *linux:info leak* – аналогично *generic:info leak*, но ошибки данного подкласса связаны с конкретным программным интерфейсом, например, с *copy_to_user()*.

В табл. 5 представлены результаты классификации и распределение типовых ошибок, исправленных в ядре ОС Linux, на основе анализа изменений, которые были сделаны в стабильных ветках ядра за последние 2 месяца 2015 года. Общее количество представителей всех подклассов типовых ошибок больше на 2 общего количества типовых ошибок из табл. 4, поскольку два изменения исправляли сразу две типовые ошибки.

По сравнению с драйверами для ядра ОС Linux в стабильных ветках исправлялось существенно больше общих ошибок: 47% вместо 29%, а также таких ошибок, как состояния гонки и взаимные блокировки: 32% вместо 20%. При этом ошибок, связанных с использованием специфичного программного интерфейса ядра ОС Linux, наоборот намного меньше: 21% вместо 50%. Также заметны следующие существенные отличия в распределении типовых ошибок по пересекающемуся списку подклассов:

- *generic:buf overflow* – 20% в ядре ОС Linux против 8% в драйверах (данное различие может быть даже больше с учетом того, что ошибки из подкласса *generic:incorrect read/write* могут относиться к данному подклассу);
- *generic:memory* – 13% и 24%;
- *generic:null ptr dereference* – 13% и 30%;
- *generic:syntax* – 4% и 14%;
- *linux:param* – 40% в ядре ОС Linux против 14% в драйверах;
- *linux:context* – 20% и 11%.

Табл. 5. Классификация и распределение типовых ошибок, исправленных в ядре ОС Linux
Table 5. Classification and distribution of typical errors fixed in the Linux kernel

Класс типовых ошибок	Подкласс типовых ошибок	Количество исправленных типовых ошибок	Суммарный процент от общего
generic (45 – 47%)	*buf overflow*	9 (20%)	20%
	memory	6 (13%)	33%
	null ptr dereference	6 (13%)	47%
	incorrect read/write	5 (11%)	58%
	int	5 (11%)	69%
	uninit	2 (4%)	73%
	infinite loop	2 (4%)	78%
	invalid cast	2 (4%)	82%
	info leak	2 (4%)	87%
	syntax	2 (4%)	91%
	always true/false condition	2 (4%)	96%
	dead code	1 (2%)	98%
	err ptr dereference	1 (2%)	100%
sync (30 – 32%)	*race*	28 (93%)	93%
	deadlock	2 (7%)	100%
linux (20 – 21%)	*param*	8 (40%)	40%
	context	4 (20%)	60%
	res	4 (20%)	80%
	info leak	2 (10%)	90%
	ret	1 (5%)	95%
	one thread	1 (5%)	100%

В табл. 6 представлено количество типовых ошибок, исправленных в основных подсистемах ядра ОС Linux. Общее количество исправленных типовых ошибок в данной таблице больше на 1 общего количества типовых ошибок из табл. 4, поскольку одно изменение затронуло сразу две подсистемы.

Табл. 6. Количество типовых ошибок, исправленных в основных подсистемах ядра ОС Linux
Table 6. Number of typical errors fixed in the core subsystems of the Linux kernel

Основная подсистема	Количество исправленных типовых ошибок	Суммарный процент от общего
net	27 (29%)	29%
drivers	22 (23%)	52%

kernel	12 (13%)	65%
arch	8 (9%)	73%
mm	6 (6%)	80%
fs	6 (6%)	86%
block	5 (5%)	91%
lib	3 (3%)	95%
security	2 (2%)	97%
crypto	2 (2%)	99%
ipc	1 (1%)	100%

5. Заключение

Метод определения границы между ядром ОС Linux и загружаемыми модулями, предложенный в данной работе, позволил оценить развитие ядра для всех версий, выпущенных за последние 7,5 лет. В результате были получены следующие наиболее существенные выводы:

- Ядро ОС Linux развивалось немного интенсивнее ядра вместе с поставляемыми с ним загружаемыми модулями. За 7,5 лет общее количество файлов увеличилось на 90% и составило 2,2 тысячи, а количество строк кода – на 105%, до 1,4 миллионов. Для ядра вместе с загружаемыми модулями данные показатели следующие: на 83% (45 тысяч) и на 95% (20 миллионов) соответственно. По размеру загружаемые модули, которые поставляются вместе с ядром, в 11 раз больше ядра.
- Основной вклад в состав и рост ядра ОС Linux вносила основная подсистема *drivers*. За ней следовали основные подсистемы *kernel*, *arch* и *net*.

Также данный метод позволил проанализировать типовые ошибки, исправленные в ядре ОС Linux, на основе изменений, сделанных в стабильных ветках ядра за последние 2 месяца 2015 года. Основные выводы следующие:

- По сравнению с драйверами для ядра ОС Linux существенно больше изменений переносится в различные стабильные ветки ядра, что косвенно служит показателем большей значимости ядра, в том числе с точки зрения исправления ошибок.
- По сравнению с драйверами для ядра ОС Linux количество изменений, в которых расширяется функциональность намного меньше (11% вместо 21%), в которых исправляются нетиповые ошибки – практически столько же (49% вместо 52%), в которых исправляются типовые ошибки – намного больше (40% вместо 27%). Таким образом, по сравнению с драйверами для ядра ОС Linux в стабильных ветках ядра исправляется больше ошибок, причем среди них больше типовых ошибок.
- По сравнению с драйверами для ядра ОС Linux в стабильных ветках

исправлялось существенно больше общих ошибок: 47% вместо 29%, а также таких ошибок, как состояния гонки и взаимные блокировки: 32% вместо 20%. При этом ошибок, связанных с использованием специфичного программного интерфейса ядра ОС Linux, наоборот намного меньше: 21% вместо 50%. Таким образом, для ядра ОС Linux в первую очередь важно искать ошибки первых двух классов, особенно с учетом того, что их последствия могут быть существенно хуже, чем последствия ошибок третьего класса. Для первого класса наибольшее количество ошибок связаны с некорректными чтением/записью и управлением памяти, для второго – с состояниями гонки.

- На половину основных подсистем, *net*, *drivers*, *kernel*, *arch*, *mm* и *fs*, пришлось 86% от общего количества типовых ошибок. В целом это коррелирует с их размером, хотя, например, по количеству строк кода основная подсистема *net* меньше основной подсистемы *drivers* в 2,3 раза в среднем за 7,5 лет в то время, как по количеству исправлений типовых ошибок в 1,2 раз больше.

Данные выводы демонстрируют развитие ядра ОС Linux, а также указывают на наиболее актуальные направления в области обеспечения качества данного проекта. Аналогичные исследования возможно провести и для ядра других ОС, особенно для тех, для которых репозитории с исходным кодом находятся в открытом доступе.

Что касается основного недостатка данного исследования, в первую очередь следует предложить подход к определению исходного кода, который входит в ядро ОС Linux для различных целевых архитектур. Скорее всего, это не повлияет существенно на классификацию и распределение типовых ошибок, но может сильно сказаться на размере некоторых основных подсистем ядра, в первую очередь, *arch*. Также следует оценить размер неучтенного исходного кода альтернативных реализаций, активируемых при выборе соответствующих конфигурационных опций.

Список литературы

[1]. В.Е. Карпов, К.А. Коньков. Основы операционных систем. Курс лекций. Учебное пособие. М.: Интернет-университет информационных технологий, 536 стр., 2005.

[2]. A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. An empirical study of operating systems errors. In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP'01), pp. 73-88, 2001.

[3]. N. Palix, G. Thomas, S. Saha, C. Calves, J. Lawall, G. Muller. Faults in Linux: ten years later. In Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11), pp. 305-318, 2011.

[4]. В.С. Мутилин, Е.М. Новиков, А.В. Хорошилов. Анализ типовых ошибок в драйверах операционной системы Linux. Труды ИСП РАН, т. 22, стр. 349-374, 2012. DOI: 10.15514/ISPRAS-2012-22-19

[5]. L. Lu, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, S. Lu. A study of Linux file system evolution. In Proceedings of the 11th USENIX conference on File and Storage Technologies (FAST'13), pp. 31-44, 2013.
[6]. N. Palix, G. Thomas, S. Saha, C. Calvès, G. Muller, J. Lawall. Faults in Linux 2.6. ACM Transactions on Computer Systems (TOCS), vol. 32, issue 2, 2014.
[7]. J. Corbet, G. Kroah-Hartman. Linux kernel development. How Fast It is Going, Who is Doing It, What They Are Doing and Who is Sponsoring the Work. http://go.linuxfoundation.org/linux-kernel-development-report-2016, 2016.

Evolution of the Linux kernel

E.M. Novikov <novikov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Existing research analyzing evolution of the Linux kernel considers the kernel together with loadable modules delivered with it or some specific subsystems of the kernel. The aim of this paper is to evaluate evolution of the kernel without loadable modules. It proposes a method for determining boundaries between them and evaluates evolution for all versions of the Linux kernel, released over the past 7.5 years. Also the paper presents a classification and a distribution of typical bugs that were fixed in the kernel, based on analysis of changes that have been made to stable branches of the kernel during the last 2 months of 2015. One can use the obtained results for evaluation of applicability of various methods and tools for software quality assurance.

Keywords: operating system; monolithic kernel; software quality; changes analysis.

DOI: 10.15514/ISPRAS-2017-29(2)-3

For citation: Novikov E.M. Evolution of the Linux kernel. Trudy ISP RAN/Proc. ISP RAS, volume 29, issue 2, 2017, pp. 77-96 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-3

References

[1]. V.E. Karpov, K.A. Kon'kov. Fundamentals of operating systems. Kurs lekcij. Uchebnoe posobie [Lectures. Study material]. M.: Internet-universitet informacionnyh tehnologij, 536 p., 2005 (in Russian).
[2]. A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. An empirical study of operating systems errors. In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP'01), pp. 73-88, 2001.
[3]. N. Palix, G. Thomas, S. Saha, C. Calves, J. Lawall, G. Muller. Faults in Linux: ten years later. In Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11), pp. 305-318, 2011.
[4]. V.S. Mutilin, E.M. Novikov, A.V. Horoshilov. Analysis of typical faults in Linux operating system drivers. Trudy ISP RAN/Proc. ISP RAS, volume 22, pp. 349-374, 2012 (in Russian). DOI: 10.15514/ISPRAS-2012-22-19
[5]. L. Lu, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, S. Lu. A study of Linux file system evolution. In Proceedings of the 11th USENIX conference on File and Storage Technologies (FAST'13), pp. 31-44, 2013.

[6]. N. Palix, G. Thomas, S. Saha, C. Calvès, G. Muller, J. Lawall. Faults in Linux 2.6. ACM Transactions on Computer Systems (TOCS), vol. 32, issue 2, 2014.

[7]. J. Corbet, G. Kroah-Hartman. Linux kernel development. How Fast It is Going, Who is Doing It, What They Are Doing and Who is Sponsoring the Work. http://go.linuxfoundation.org/linux-kernel-development-report-2016, 2016.

Возможности статической верификации монолитного ядра операционных систем[1]

Е.М. Новиков <novikov@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. У большинства современных, повсеместно используемых операционных систем архитектура ядра в той или иной степени является монолитной, поскольку именно данная архитектура позволяет обеспечить максимальную производительность работы. Как правило, размер монолитного ядра без различных расширений, таких как драйверы устройств, составляет несколько миллионов строк кода на языке программирования Си/Си++ и языке ассемблера. С течением времени исходный код достаточно интенсивно изменяется: добавляется поддержка новой функциональности, оптимизируется выполнение различных операций, исправляются ошибки. Высокая практическая значимость монолитного ядра операционных систем определяет строгие требования к его функциональности, безопасности, надежности и производительности. Те подходы к обеспечению качества программных систем, которые в настоящее время используются на практике, позволяют выявить и исправить достаточно большое количество ошибок, однако ни один из них не позволяет обнаружить все возможные ошибки искомых видов. В этой статье показывается, что различные подходы к статической верификации, которые нацелены на решение данной задачи, имеют существенные ограничения, если их применять к монолитному ядру операционных систем целиком, в первую очередь из-за большого размера и сложности исходного кода, который постоянно изменяется. В качестве первого шага в направлении статической верификации монолитного ядра операционных систем предлагается метод декомпозиции ядра на подсистемы.

Ключевые слова: операционная система; монолитное ядро; микроядро; качество программной системы; статическая верификация; формальная спецификация; декомпозиция программной системы.

DOI: 10.15514/ISPRAS-2017-29(2)-4

Для цитирования: Новиков Е.М. Возможности статической верификации монолитного ядра операционных систем. Труды ИСП РАН, том 29, вып. 2, 2017 г., стр. 97-116. DOI: 10.15514/ISPRAS-2016-29(2)-4

[1] Исследование выполнено при финансовой поддержке РФФИ, проект «Инкрементальная статическая верификация подсистем монолитного ядра операционных систем» № 16-31-60097.

1. Введение

У большинства современных, повсеместно используемых операционных систем (далее — ОС) архитектура ядра в той или иной степени является монолитной, поскольку именно данная архитектура позволяет обеспечить максимальную производительность работы [1]. Как правило, размер монолитного ядра без различных расширений, таких как драйверы устройств (далее в работе это будет называться монолитным ядром), составляет несколько миллионов строк кода на языке программирования Си/Си++ и языке ассемблера. Традиционно по мере разработки монолитного ядра ОС сохраняется совместимость его программного интерфейса для пользовательских приложений, однако при этом с течением времени исходный код достаточно интенсивно изменяется: добавляется поддержка новой функциональности, оптимизируется выполнение различных операций, исправляются ошибки. Например, за 7,5 лет размер монолитного ядра ОС Linux вырос более чем в 2 раза, и на сегодняшний день составляет около 1,4 миллионов строк кода [2].

Высокая практическая значимость монолитного ядра ОС определяет строгие требования к его функциональности, безопасности, надежности и производительности. В случае ошибок и сбоев в монолитном ядре возможны некорректная работа, повреждение данных и снижение производительности самого ядра, драйверов, модулей и пользовательских приложений, могут быть нарушены права и конфиденциальность данных пользователей ОС.

В данной работе делается обзор тех методов и инструментов, которые уже применяются на практике для обеспечения качества монолитного ядра ОС (раздел 2). Отмечается, что ни один из них не позволяет обнаружить все возможные ошибки искомых видов. На решение данной задачи нацелены различные подходы к статической верификации, которые рассматриваются в разделе 3. Также в этом разделе показывается, что существующие подходы к статической верификации ядра ОС имеют существенные ограничения, если их применять к монолитному ядру операционных систем целиком, в первую очередь из-за большого размера и сложности исходного кода, который постоянно изменяется. Раздел 4 представляет метод декомпозиции монолитного ядра ОС, который позволит применять инструменты автоматической статической верификации для монолитного ядра ОС. В заключении подводятся итоги данной работы.

2. Используемые на практике подходы к обеспечению качества монолитного ядра ОС

В настоящее время на практике качество монолитного ядра ОС обеспечивается посредством экспертизы кода, сборки и запуска ядра, тестирования, статического анализа и за счет исправления ошибок, обнаруженных пользователями. Данные подходы в совокупности позволяют

выявить и исправить достаточно большое количество ошибок. Однако ни один из подходов не позволяет обнаружить все возможные ошибки искомых видов [3].

Например, экспертиза кода широко используется в процессе разработки ядра ОС Linux [4]. Тем не менее, возможности данного подхода ограничены. С учетом того, что монолитное ядро ОС имеет достаточно большой объем сложного исходного кода, который быстро развивается, экспертиза кода не позволяет гарантировать отсутствие ошибок ни для существующего исходного кода, ни для постоянно идущих изменений.

Сборка и запуск ядра, тестирование и статический анализ могут быть проведены автоматизированным образом самими разработчиками монолитного ядра ОС. Помимо этого для проектов с открытым исходным кодом были разработаны специализированные инфраструктуры, например, 0-day[1], kernelci.org[2], OSS-Fuzz[3] и Coverity Scan[4], которые еще больше автоматизировали данные подходы. В случае использования данных инфраструктур разработчикам предлагается рассмотреть сообщения о выявленных ошибках и предпринять соответствующие действия, например, исправить ошибку или пометить, что выдано ложное сообщение об ошибке.

Сборка ядра позволяет выявить ошибки, которые сообщаются инструментами, осуществляющими сборку, в основном компилятором и компоновщиком. В ходе запуска ядра выявляются ошибки, которые происходят при инициализации ядра, например, при монтировании файловых систем. Можно обнаружить те же виды ошибок, что и при тестировании. Примечательно то, что в отличие от других автоматизированных подходов к обеспечению качества сборка и запуск ядра часто осуществляются для различных конфигураций и архитектур, поскольку они не требуют существенных усилий при первоначальной настройке и могут быть выполнены достаточно быстро.

Тестирование требует подготовки специального тестового окружения для того, чтобы осуществить реальное выполнение с различными входными данными. Тестирование позволяет выявить широкий спектр проблем: падения, например, вследствие разыменования нулевого указателя, некорректная функциональность, нарушение прав пользователей, деградация производительности, в том числе зависания, и т.д. Иногда для того чтобы расширить набор проверок, выполняемых в ходе тестирования, и упростить поиск причин ошибок, при сборке монолитного ядра ОС включаются соответствующие конфигурационные опции[5]. При обычной эксплуатации это, как правило, не делается, поскольку может привести к достаточно существенным накладным расходам. При тестировании достаточно сложно

[1] https://01.org/lkp.
[2] https://kernelci.org/.
[3] https://github.com/google/oss-fuzz.
[4] https://scan.coverity.com/.
[5] https://github.com/google/kasan/wiki.

достигнуть большого покрытия, проверяются не все возможные ситуации и часть ошибок пропускается. Для увеличения покрытия, а также поиска некоторых специфичных ошибок требуется прикладывать большие усилия [5, 6].

Статический анализ позволяет обнаружить ошибки на всех возможных путях выполнения, поскольку некоторое представление программы анализируется без ее реального выполнения с некоторыми определенными входными данными. Данное направление активно исследуется в последние годы. На сегодняшний день существует множество коммерческих и академических инструментов, с помощью которых уже удалось получить хорошие результаты [7-10]. Основным недостатком статического анализа является достаточно большое количество ложных сообщений об ошибках, которых, как правило, не бывает при использовании других подходов к обеспечению качества. Для того чтобы решить эту проблему, а также проводить анализ за время, сравнимое по порядку со временем сборки, в инструментах реализуются различные эвристики. Это приводит к тому, что, во-первых, анализируются не все возможные пути, например, не рассматриваются все итерации циклов или вызовы по функциональным указателям. Во-вторых, некоторые потенциальные ошибки могут преднамеренно игнорироваться, поскольку за отведенное время и с учетом сделанных упрощений инструменты не в состоянии определить точно, возможны они на самом деле или нет. Поскольку наборы эвристик в разных инструментах статического анализа отличаются, они находят разные ошибки одних и тех же видов в одних и тех же программных системах. Преимущественно эти инструменты применяются для поиска нарушений общих правил безопасного программирования, таких как разыменование нулевого указателя, выход за границу массива и т.д. Некоторые инструменты позволяют выявлять более специфичные ошибки, например, в ядре ОС Linux: checkpatch[1] и Coccinelle [8].

Достаточно много ошибок в монолитном ядре ОС, которые не обнаружили все рассмотренные ранее подходы, сообщается пользователями, которые сталкиваются с ними в процессе эксплуатации. Для того чтобы упростить этот процесс, разрабатываются специальные средства, которые могут в том числе сообщать разработчикам статистику использования и информацию о падениях автоматически[2]. Несмотря на то, что пользователей монолитного ядра ОС может быть чрезвычайно много, данный подход тем не менее также не гарантирует отсутствие ошибок.

Все рассмотренные подходы к обеспечению качества монолитного ядра ОС необходимо развивать и использовать в дальнейшем, поскольку каждый из них позволяет выявить определенные ошибки, в том числе критичные, и ни один из них не заменяет полностью другой. Наряду с этим требуется

[1] https://github.com/torvalds/linux/blob/master/scripts/checkpatch.pl.

[2] https://technet.microsoft.com/en-us/library/cc754364(v=ws.11).aspx, https://support.apple.com/en-us/HT202031.

предлагать и исследовать новые методы, которые должны быть нацелены на обнаружение всех возможных ошибок искомых видов.

3. Возможности статической верификации монолитного ядра ОС

Потенциально выявить все ошибки искомого вида в программных системах или доказать их корректность можно с помощью методов и инструментов статической верификации. В данном разделе рассматриваются три основных направления в области статической верификации ядра ОС и показывается, что в настоящее время соответствующие подходы не в состоянии решить проблему верификации монолитного ядра ОС целиком по различным причинам.

3.1 Статическая верификация микроядра ОС

Наибольшее продвижение в области статической верификации ядра ОС было сделано для микроядра. Размер микроядра ОС составляет несколько тысяч или десятков тысяч строк кода. Для его статической верификации строятся модели и формальные спецификации, которые покрывают функциональные требования, а также некоторые дополнительные свойства, такие как разделение памяти для пользовательских приложений. Верификация проводится при выполнении определенных условий, например:

- рассматриваются только определенные аппаратные платформы и модели процессоров;
- допускается использовать подмножества языков программирования;
- предполагается корректность работы аппаратного обеспечения и компилятора.

Несмотря на все упрощения и предположения задача полной статической верификации микроядра ОС является чрезвычайно трудоемкой и, как правило, осуществляется только для некоторого подмножества исходного кода.

Например, удалось формально доказать полную функциональную корректность для микроядра seL4, которое состоит из примерно 10 тысяч строк кода на языке программирования Си и языке ассемблера [11]. Размер спецификаций при этом составил примерно 400 тысяч строк. При доказательстве использовались предположения о корректности кода на языке ассемблера, компилятора и аппаратуры. В дополнение к функциональным требованиям позднее для микроядра seL4 были сформулированы такие высокоуровневые свойства, как соблюдение прав доступа и разграничение информационных потоков [12, 13]. Данные свойства важны для построения более надежных программных систем, которые используют микроядро в своей основе. Для них были разработаны соответствующие модели и спецификации, что позволило проверить их выполнимость [14-17]. В [18] подводятся итоги

многолетней работы по разработке и статической верификации микроядра seL4. На момент завершения работы размер спецификаций составил 480 тысяч строк. В целом на разработку и верификацию потребовалось около 2-х и 28-ми человеко-лет соответственно. Важный вывод, который продемонстрировали авторы данного исследования, состоит в том, что разработка программных систем с использованием методов и инструментов формальной верификации существенно более выгодна по стоимости и позволяет достигнуть более высокого уровня качества по сравнению с разработкой программных систем высокой надежности с использованием традиционных подходов.

В аналогичном проекте была поставлена цель верифицировать всю низкоуровневую программно-аппаратную систему, включая микроядро ОС реального времени PikeOS, которое уже использовалось в промышленности [19, 20]. В работе [21] были обсуждены разработка спецификаций и верификация для одного из наиболее важных нефункциональных свойств ядра ОС, а именно, для разделения памяти между пользовательскими приложениями. Примечательно то, что в данном проекте был также верифицирован исходный код на языке ассемблера [22].

В работе [23] было предложено изменить подход к разработке ядра ОС и спецификаций, благодаря чему удалось существенно сократить их размер и время разработки. Как и для PikeOS, удалось формально верифицировать код на языке ассемблера. Отмечается, что хотя в статьях часто декларируется полная формальная верификации, на деле часть кода остается непроверенной. Последняя работа не является исключением из данного правила.

В другой работе было предложено использовать сразу несколько методов анализа и верификации с целью проверки выполнения различных свойств для микроядра ОС [24]. Отмечается, что данный подход применим и для обеспечения качества монолитного ядра ОС. В статье представлены только первоначальные результаты проекта, так что не понятно, в какой мере удалось добиться поставленной цели хотя бы для рассматриваемого микроядра ОС.

В отличие от предыдущих исследований в [25] было предложено разрабатывать и верифицировать ядро ОС ExpressOS, которое позволяет запускать критичные пользовательские приложения ОС общего назначения Android. Статическая верификация была проведена только для проверки выполнения наиболее важных свойств безопасности, таких как разделение памяти и безопасное межпроцессное взаимодействие. Это позволило существенно сократить размер спецификаций. Было продемонстрировано, что разработанное ядро ОС оказалось неподверженным большинству известных серьезных уязвимостей. Несколько тестов показали, что по производительности оно не существенно уступало неверифицированному ядру ОС общего назначения.

Работа [26] интересна тем, что была статически верифицирована одна из наиболее важных частей существующего ядра ОС реального времени, а именно, планировщик. Была доказана функциональная корректность, а также

безопасность работы с памятью. Это исследование демонстрирует, что существующие методы и инструменты позволяют формально верифицировать небольшие важные части ядра. Однако процесс оказался достаточно трудоемким, поскольку потребовалось переписать исходный код таким образом, чтобы его нотация поддерживалась используемым инструментом верификации. Аналогичная работа была проделана для ядра ОС Linux [27]. Однако авторы указали достаточно большое количество ограничений, что ставит под сомнение возможность практического использования предложенного подхода.

Распространить опыт статической верификации микроядра ОС на монолитное ядро ОС в настоящее время не представляется возможным по следующим причинам:

- Размер монолитного ядра превышает размер микроядра на 2-3 порядка, при этом исходный код постоянно изменяется [2]. Соответственно для полной верификации потребуются чрезвычайно большие усилия высококвалифицированных инженеров. Можно попытаться тщательно верифицировать только наиболее критичные компоненты, но это не дало бы гарантии отсутствия ошибок определенных видов во всем монолитном ядре ОС в целом.
- При разработке монолитного ядра ОС используется вся мощь языков программирования, в том числе всевозможные расширения, например, GNU. Существующие инструменты для разработки спецификаций и верификации не поддерживают это.
- При проектировании микроядра разработчики стараются сделать так, чтобы впоследствии было проще разрабатывать спецификации и проводить статическую верификацию и сертификацию. Монолитное ядро ОС проектируется и разрабатывается по другим принципам, что существенно затрудняет его верификацию.

3.2 Использование специализированных языков программирования и архитектур

Существует достаточно много работ, в которых предлагается использовать специализированные языки программирования или архитектуры для того, чтобы разрабатывать более надежное ядро ОС, а также достаточно сильно упростить его последующую верификацию.

Например, в [28] базовая часть ядра ОС была разработана на типизированном языке ассемблера. Для нее были разработаны спецификации и доказана корректность относительно данных спецификаций. Основная часть ядра была разработана на языке программирования C#. Благодаря предложенной архитектуре для нее удалось автоматически проверить выполнимость свойств безопасности работы с памятью. Необходимо отметить, что данные свойства не исчерпывают все необходимые свойства безопасного и надежного ядра ОС.

В [29, 30] предлагается использовать высокоуровневый язык программирования для разработки частей ядра ОС с целью их постепенной интеграции в существующее ядро ОС, разработанное на низкоуровневых языках программирования. Это позволяет избежать некоторых ошибок, в том числе нарушения достаточно сложных функциональных требований. Однако подход не нацелен на возможность проверки всех необходимых свойств. Авторы уделили внимание тому, что новые компоненты должны работать как можно более эффективно, однако оценка накладных расходов не была сделана. Авторы рассмотрели несколько примеров с реализацией компонентов ядра ОС, которые не используется широко, но имеют достаточно высокоуровневую реализацию. Применимость данного подхода к компонентам монолитного ядра ОС, которые имеют достаточно сложную реализацию, не изучалась.

Также было предложено использовать альтернативные средства разработки только для сложных типов данных с указателями [31]. Авторы задали их специальным образом для микроядра ОС Fiasco.OC. Затем это представление было автоматически транслировано в исходный код на языке программирования Си++, на котором написан остальной исходный код данного микроядра. Авторы уверяют, что такой подход не вносит существенных накладных расходов, но зато позволяет автоматически проверять выполнимость некоторых свойств, а именно, безопасности работы с памятью. Проверка других свойств не рассматривалась.

Еще один подход к разработке и верификации программных систем состоит в том, что они полностью пишутся на специализированном языке программирования, причем одновременно с реализацией задаются спецификации [32]. Это позволяет проводить верификацию непосредственно по мере разработки. Автор рассмотрел несколько тестовых примеров, поэтому остается неясным, насколько данный подход масштабируем и эффективен на практике.

Помимо использования специализированных средств программирования предлагается использовать также и специализированные аппаратные средства [33, 34]. Авторы отмечают, что архитектура большинства используемых на сегодняшний день программно-аппаратных систем была построена по принципу эффективного использования достаточно ограниченного объема вычислительных ресурсов. Данный принцип необходимо пересматривать таким образом, чтобы гарантировать безопасность работы ценой дополнительных ресурсов.

Все данные подходы могут быть использованы при построении программно-аппаратных систем специального назначения, но не применимы к верификации монолитного ядра современных, повсеместно используемых ОС, поскольку разработчики предпочитают использовать всю мощь общецелевых языков программирования и инструментов, а работать оно должно эффективно на оборудовании общего назначения. Данное направление не

пользуется большой популярностью на практике, но вызывает достаточно большой интерес у исследователей.

3.3 Автоматическая статическая верификация

На сегодняшний день наиболее значимые результаты при статической верификации компонентов монолитного ядра повсеместно используемых ОС были получены при использовании инструментов автоматической статической верификации, таких как SLAM [35], BLAST, CPAchecker, CBMC и др. [36, 37]. Данные инструменты позволяют доказывать за приемлемое время выполнимость специфицированных свойств для программных систем размером несколько десятков тысяч строк кода на различных языках программирования. Они были успешно использованы для статической верификации драйверов ОС Microsoft Windows [38] и модулей ядра ОС Linux [39-42], большинство из которых укладываются в указанное ограничение по размеру. Удалось выявить сотни ошибок, которые были признаны разработчиками [38, 42].

Изначально инструменты автоматической статической верификации были нацелены на проверку такого свойства, как достижимость, к которому можно свести, например, правила корректного использования программного интерфейса [38-42]. Вообще говоря, таким образом можно выразить и традиционные функциональные требования. Позднее была добавлена поддержка проверки таких свойств, как безопасность работы с памятью и завершимость [43, 44].

Инструменты автоматической статической верификации не используют эвристики так, как это делают инструменты статического анализа, а выполняют точный анализ всех путей выполнения. Благодаря этому они позволяют выявить все ошибки искомого вида в программных системах или доказать их корректность.

Опыт использования данных инструментов показал, что они выдают большое количество ложных сообщений об ошибках [45]. Для получения более качественных результатов верификации необходимо разрабатывать достаточно точные спецификации моделей окружения и проверяемых правил [46-48]. Также инструменты могут потреблять чрезвычайно большое количество ресурсов, в первую очередь, процессорного времени и оперативной памяти, что особенно остро проявляется при увеличении размера анализируемых программных систем.

В следующем разделе данной статьи показывается, как предлагается учитывать эти особенности при использовании инструментов автоматической статической верификации для монолитного ядра ОС.

4. Метод декомпозиции монолитного ядра ОС на подсистемы

В предыдущем разделе было показано, что наиболее перспективными методами и инструментами для обнаружения всех ошибок искомого вида в монолитном ядре ОС являются методы и инструменты автоматической статической верификации. Данные инструменты позволяют анализировать исходный код монолитного ядра ОС в оригинальном виде, что особенно важно ввиду его большого размера и высокого темпа развития, и выявлять ошибки, которые обусловлены некорректной работой с памятью и некорректным использованием программного интерфейса[1]. Автором работы было показано, что эти ошибки являются одними из наиболее распространенных для монолитного ядра ОС Linux [2].

Как было отмечено, на текущий момент инструменты автоматической статической верификации способны верифицировать промышленные программы размером порядка нескольких десятков тысяч строк кода. Для того чтобы применить их для верификации монолитного ядра ОС, размер которого составляет несколько миллионов строк кода, необходимо декомпозировать монолитное ядро на подсистемы, которые возможно проверять по отдельности. Декомпозиция должна быть проведена таким образом, чтобы, во-первых, инструменты автоматической статической верификации могли анализировать получившийся исходный код с разумными ограничениями на используемые вычислительные ресурсы и общее время проверки, для чего все подсистемы должны быть ограничены по размеру и сложности. Во-вторых, необходимо иметь возможность постепенно повышать качество верификационных результатов (сокращать количество ложных сообщений об ошибках и находить ошибки новых видов) за счет инкрементальной разработки спецификаций моделей окружения и проверяемых свойств. Такой подход уже достаточно хорошо проявил себя при верификации драйверов ядра ОС Microsoft Windows и модулей ядра ОС Linux, поэтому с большой вероятностью аналогичных результатов удастся добиться и для подсистем монолитного ядра.

В настоящий момент автору представляется наиболее правильным проводить декомпозицию монолитного ядра ОС на группы файлов (подсистемы) по принципу разделения четко выраженной функциональности. Таким образом, следует выделить подсистему управления памяти, планировщик, компоненты сетевой подсистемы и т.д. Поскольку такому подходу, как правило, следуют при разработке, ожидается, что провести декомпозицию на практике будет достаточно легко.

[1] Автору неизвестен опыт удачного применения инструментов автоматической статической верификации для проверки выполнения других свойств для промышленных программных систем. Однако в ближайшем будущем такой опыт скорее всего появится, поскольку это направление пользуется большой популярностью у исследователей и активно развивается.

Каждая из выделенных подсистем, скорее всего, окажется достаточно компактной по размеру, а значит, будет поддаваться анализу. В том случае, если какая-нибудь подсистема получится слишком большой или сложной, потребуется провести дополнительную декомпозицию. В предположении, что размер каждой подсистемы составит порядка 10 тысяч строк кода, общее количество подсистем для типового монолитного ядра ОС составит порядка 100.

Задавать группы файлов предлагается по их принадлежности к директориям. В том случае, если файлы одной директории потребуется отнести к различным подсистемам, то их надо будет перечислить явным образом для каждой из этих подсистем. Такой подход позволит проще адаптироваться к другим версиям монолитного ядра ОС, поскольку директории создаются, удаляются и перемещаются намного реже, чем файлы.

При использовании такого подхода к декомпозиции монолитного ядра ОС на уровне каждой подсистемы реализация будет достаточно сильно взаимосвязана, при этом моделировать соответствующую функциональность не потребуется, поскольку будет верифицироваться сразу весь исходный код подсистемы. В том случае, если при верификации будут выдаваться ложные сообщения об ошибках вследствие отсутствия при анализе реализаций других подсистем, их нужно будет постепенно моделировать. Например, большинство подсистем обращаются к подсистеме управления памятью, поэтому с большой вероятностью потребуется разработать ее модель. Моделировать те подсистемы, которые не используются другими, не потребуется. Такими подсистемами с большой вероятностью окажутся, например, подсистемы поддержки различных типов устройств, которые часто реализуются в монолитном ядре ОС.

Также для подсистем нужно будет инкрементально разрабатывать:

- Спецификации моделей окружения, отвечающих за обращения к соответствующим подсистемам. Например, описывать всевозможные последовательности вызовов функций со всевозможными значениями их аргументов – соответствующий подход был предложен в работе [48].
- Спецификации правил использования программного интерфейса других подсистем [47]. Для проверки свойства безопасности работы с памятью разрабатывать дополнительные спецификации не потребуется, поскольку их проверка интегрирована в инструменты автоматической статической верификации [43, 44].

При разработке спецификаций нужно обязательно предусмотреть возможность достаточно быстро адаптировать их при условии изменений реализации ядра и требований к его окружению, что уже было учтено в проекте по статической верификации модулей ядра ОС Linux [47].

Можно предложить и другие подходы к декомпозиции монолитного ядра ОС на подсистемы, например, верифицировать отдельно каждый файл монолитного ядра ОС. Такой подход будет существенно более трудоемким, поскольку файлов в монолитном ядре ОС много, например, около 2,2 тысяч для монолитного ядра ОС Linux 4.8 [2], и придется разрабатывать достаточно большую спецификацию модели окружения. Рассматривать по отдельности каждую функцию представляется еще менее перспективным, поскольку обычно их намного больше, чем файлов, например, около 50 тысяч для монолитного ядра ОС Linux 4.8, и они сильно взаимосвязаны друг с другом.

Можно разделить исходный код монолитного ядра ОС таким образом, чтобы каждая подсистема содержала весь исходный код, который может использоваться при выполнении одного из системных вызовов или одной из функций из программного интерфейса ядра. Ожидается, что общее количество таких подсистем будет намного меньше, чем количество всех функций монолитного ядра ОС, но сравнимо по порядку с количеством файлов. При этом реализация всей связанной функциональности будет рассматриваться единовременно в отличие от того подхода, при котором рассматриваются отдельные файлы, так что моделирование окружения будет намного менее трудоемким. По сравнению с предложенным ранее подходом к декомпозиции на непересекающиеся группы файлов также потребуется разрабатывать существенно меньше моделей, поскольку не будет нужно моделировать другие подсистемы.

При использовании данного подхода с большой вероятностью некоторые подсистемы окажутся достаточно большими и сложными. Например, в большинство подсистем войдет та или иная часть подсистемы управления памятью. Провести статическую верификацию в таком случае окажется затруднительно или невозможно. Еще одной предполагаемой сложностью данного подхода является определение границы подсистем. Например, некоторые функции могут вызываться не напрямую, а посредством функциональных указателей. Поскольку для такого случая нельзя простыми методами определить конкретные функции в общем случае, потребуется выполнить определенное моделирование. Аналогичная ситуация возникает для глобальных переменных, которые могут быть косвенно модифицированы при вызове функций из других подсистем, – это также потребуется моделировать некоторым образом.

Без оценки на практике невозможно определить, какой из двух подходов к декомпозиции монолитного ядра ОС, разделение на непересекающиеся группы файлов или выбор всего исходного кода, относящегося к системным вызовам и функциям программного интерфейса ядра, окажется менее трудоемким и/или позволит получить более качественные результаты верификации. Возможно окажется целесообразным применять оба данных подхода или их композицию для некоторой части или даже для всего монолитного ядра ОС в целом.

5. Заключение

Используемые в настоящее время подходы к обеспечению качества монолитного ядра ОС позволяют выявить и исправить достаточно большое количество ошибок, однако ни один из подходов не позволяет обнаружить все возможные ошибки искомых видов. Решить эту задачу теоретически способна статическая верификация. Большинство существующих подходов к статической верификации ядра ОС имеют существенные ограничения, если их применять к монолитному ядру ОС целиком. Наиболее перспективными в этом направлении являются методы и инструменты автоматической статической верификации программных систем. В данной работе предлагается метод декомпозиции монолитного ядра ОС на подсистемы с целью их последующей верификации.

Список литературы

[1]. В.Е. Карпов, К.А. Коньков. Основы операционных систем. Курс лекций. Учебное пособие. М.: Интернет-университет информационных технологий, 536 с., 2005.

[2]. Е.М. Новиков. Развитие ядра операционной системы Linux. Труды ИСП РАН, т. 29, вып. 2, 2017, стр. xx-xx. DOI: 10.15514/ISPRAS-2017-29(2)-3

[3]. R.L. Glass. Facts and fallacies of software engineering. Addison-Wesley Professional, 2002.

[4]. J. Corbet, G. Kroah-Hartman. Linux kernel development. How Fast It is Going, Who is Doing It, What They Are Doing and Who is Sponsoring the Work. http://go.linuxfoundation.org/linux-kernel-development-report-2016, 2016.

[5]. А.В. Цыварев, В.А. Мартиросян. Тестирование драйверов файловых систем в ОС Linux. Труды ИСП РАН, т. 23, 2012, стр. 413-426. DOI: 10.15514/ISPRAS-2012-23-24

[6]. J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, A. Fedorova. The Linux scheduler: a decade of wasted cores. In Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16), article 1, 16 pages, 2016.

[7]. A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. An empirical study of operating systems errors. In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP'01), pp. 73-88, 2001.

[8]. J.L. Lawall, J. Brunel, N. Palix, H.R. Rydhof, H. Stuart, G. Muller. WYSIWIB: A declarative approach to finding API protocols and bugs in Linux code. Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 43-52, 2009.

[9]. А. Аветисян, А. Белеванцев, А. Бородин, В. Несов. Использование статического анализа для поиска уязвимостей и критических ошибок в исходном коде программ. Труды ИСП РАН, т. 21, 2011, стр. 23-38.

[10]. N. Palix, G. Thomas, S. Saha, C. Calves, J. Lawall, G. Muller. Faults in Linux: ten years later. In Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11), pp. 305-318, 2011.

[11]. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, S. Winwood. seL4: formal verification of an OS kernel. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP'09), pp. 207-220, 2009.

[12]. J. Andronick, D. Greenaway, K. Elphinstone. Towards proving security in the presence of large untrusted components. In Proceedings of the 5th international conference on Systems software verification (SSV'10), pp. 9-9, 2010.

[13]. G. Klein. From a verified kernel towards verified systems. In Proceedings of the 8th Asian conference on Programming languages and systems (APLAS'10), pp. 21-33, 2010.

[14]. I. Kuz, G. Klein, C. Lewis, A. Walker. capDL: a language for describing capability-based systems. In Proceedings of the first ACM asia-pacific workshop on Workshop on systems (APSys'10), pp. 31-36, 2010.

[15]. T. Sewell, S. Winwood, P. Gammie, T. Murray, J. Andronick, G. Klein. seL4 enforces integrity. In Proceedings of the Second international conference on Interactive theorem proving (ITP'11), pp. 325-340, 2011.

[16]. T. Murray, D. Matichuk, M. Brassil, P. Gammie, G. Klein. Noninterference for operating system kernels. In Proceedings of the Second international conference on Certified Programs and Proofs (CPP'12), pp. 126-142, 2012.

[17]. M. Daum, N. Billing, G. Klein. Concerned with the unprivileged: user programs in kernel refinement. Formal Aspects of Computing, vol. 26, issue 6, pp. 1205-1229, 2014.

[18]. G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, G. Heiser. Comprehensive formal verification of an OS microkernel. ACM Transactions on Computer Systems (TOCS), vol. 32, issue 1, 70 pages, 2014.

[19]. C. Baumann, B. Beckert, H. Blasum, T. Bormer. Formal Verification of a Microkernel Used in Dependable Software Systems. In Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security (SAFECOMP'09), pp. 187-200, 2009.

[20]. E. Alkassar, W. J. Paul, A. Starostin, A. Tsyban. Pervasive verification of an OS microkernel: inline assembly, memory consumption, concurrent devices. In Proceedings of the Third international conference on Verified software: theories, tools, experiments (VSTTE'10), pp. 71-85, 2010.

[21]. C. Baumann, T. Bormer, H. Blasum, S. Tverdyshev. Proving Memory Separation in a Microkernel by Code Level Verification. In Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW '11), pp. 25-32, 2011.

[22]. S. Schmaltz, A. Shadrin. Integrated semantics of intermediate-language c and macro-assembler for pervasive formal verification of operating systems and hypervisors from VerisoftXT. In Proceedings of the 4th international conference on Verified Software: theories, tools, experiments (VSTTE'12), pp. 18-33, 2012.

[23]. R. Gu, J. Koenig, T. Ramananandro, Z. Shao, X. Wu, S.-C. Weng, H. Zhang, Y. Guo. Deep Specifications and Certified Abstraction Layers. SIGPLAN Not. 50, 1, pp. 595-608, 2015.

[24]. M. Děcký. A road to a formally verified general-purpose operating system. In Proceedings of the First international conference on Architecting Critical Systems (ISARCS'10), pp. 72-88, 2010.

[25]. H. Mai, E. Pek, H. Xue, S. T. King, P. Madhusudan. Verifying security invariants in ExpressOS. In Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems (ASPLOS '13), pp. 293-304, 2013.

[26]. J.F. Ferreira, C. Gherghina, G. He, S. Qin, W.-N. Chin. Automated verification of the FreeRTOS scheduler in Hip/Sleek. Int. J. Softw. Tools Technol. Transf. 16, 4, pp. 381-397, 2014.

[27]. A. Gotsman, H. Yang. Modular verification of preemptive OS kernels. In Proceedings of the 16th ACM SIGPLAN international conference on Functional programming (ICFP '11), pp. 404-417, 2011.

[28]. J. Yang, Chris Hawblitzel. Safe to the last instruction: automated verification of a type-safe operating system. In Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '10), pp. 99-110, 2010.

[29]. M. Danish, H. Xi. Operating system development with ATS: work in progress. In Proceedings of the 4th ACM SIGPLAN workshop on Programming languages meets program verification (PLPV'10), pp. 9-14, 2010.

[30]. M. Danish, H. Xi. Using Lightweight Theorem Proving in an Asynchronous Systems Context. In Proceedings of the 6th International Symposium on NASA Formal Methods, vol. 8430, pp. 158-172. 2014.

[31]. B.W. Cronkite-Ratcliff. Development of automatically verifiable systems using data representation synthesis. In Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity (SPLASH '13), pp. 109-110, 2013.

[32]. K.R.M. Leino. Developing verified programs with dafny. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13), pp. 1488-1490, 2013.

[33]. A. DeHon, B. Karel, T. F. Knight, Jr., G. Malecha, B. Montagu, R. Morisset, G. Morrisett, B. C. Pierce, R. Pollack, S. Ray, O. Shivers, J. M. Smith, G. Sullivan. Preliminary design of the SAFE platform. In Proceedings of the 6th Workshop on Programming Languages and Operating Systems (PLOS '11), article 4, 5 pages, 2011.

[34]. A. Azevedo de Amorim, N. Collins, A. DeHon, D. Demange, C. Hriţcu, D. Pichardie, B. C. Pierce, R. Pollack, A. Tolmach. A verified information-flow architecture. In Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'14), pp. 165-178, 2014.

[35]. T. Ball, E. Bounimova, R. Kumar, V. Levin. SLAM2: Static driver verification with under 4% false alarms. In Proceedings of the 10th International Conference on Conference on Formal Methods in Computer-Aided Design (FMCAD'10), pp. 35-42, 2010.

[36]. D. Beyer. Status Report on Software Verification (Competition Summary SV-COMP 2014). In Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and of Analysis Systems (TACAS'14), LNCS 8413, pp. 373-388, 2014.

[37]. D. Beyer. Software Verification and Verifiable Witnesses (Report on SV-COMP 2015). In Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and of Analysis Systems (TACAS'15), LNCS 9035, pp. 401-416, 2015.

[38]. T. Ball, V. Levin, S.K. Rajamani. A decade of software model checking with SLAM. Communications of the ACM, vol. 54, issue 7, pp. 68-76, 2011.

[39]. T. Witkowski, N. Blanc, D. Kroening, G. Weissenbacher. Model checking concurrent Linux device drivers. In Proceedings of the 22nd IEEE/ACM international conference on Automated Software Engineering (ASE'07), pp. 501-504, 2007.

[40]. H. Post, W. Küchlin. Integrated static analysis for Linux device driver verification. In Proceedings of the 6th International Conference on Integrated Formal Methods (IFM'07), LNCS, vol. 4591, pp. 518-537, 2007.

[41]. Д. Бейер, А.К. Петренко. Верификация драйверов операционной системы Linux. Труды ИСП РАН, т. 23, стр. 405-412, 2012. DOI: 10.15514/ISPRAS-2012-23-23

[42]. И.С. Захаров, М.У. Мандрыкин, В.С. Мутилин, Е.М. Новиков, А.К. Петренко, А.В. Хорошилов. Конфигурируемая система статической верификации модулей ядра операционных систем. Программирование, т. 41, выпуск 1, стр. 44-67, 2015.

[43]. T. Ströder, C. Aschermann, F. Frohn, J. Hensel, J. Giesl. AProVE: Termination and Memory Safety of C Programs (Competition Contribution). In Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15), LNCS 9035, pp. 417-419, 2015.

[44]. M. Kotoun, P. Peringer, V. Šoková, T. Vojnar. Optimized PredatorHP and the SV-COMP Heap and Memory Safety Benchmark. In Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16), vol. 9636, pp. 942-945, 2016.

[45]. D. Engler, M. Musuvathi. Static analysis versus model checking for bug finding. In Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'04), LNCS, vol. 2937, pp. 191-210, 2004.

[46]. T. Ball, S.K. Rajamani. SLIC: A specification language for interface checking of C. Technical Report MSR-TR-2001-21, Microsoft Research, 2001.

[47]. Е.М. Новиков. Развитие метода контрактных спецификаций для верификации модулей ядра операционной системы Linux. Диссертация на соискание ученой степени кандидата физико-математических наук, Институт системного программирования РАН, 2013.

[48]. A. Khoroshilov, V. Mutilin, E. Novikov, I. Zakharov. Modeling Environment for Static Verification of Linux Kernel Modules. In Proceedings of the 9th International Ershov Informatics Conference (PSI'14), LNCS 8974, pages 400-414, 2014.

Static verification of operating system monolithic kernels

E.M. Novikov <novikov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The most of modern widely used operating systems have monolithic kernels since this architecture aims at reaching maximum performance. Usually monolithic kernels without various extensions like device drivers consist of several million lines of code in the programming language C/C++ and in the assembly language. With time, their source code evolves quite intensively: a new functionality is supported, various operations are optimized, bugs are fixed. The high practical value of operating system monolithic kernels defines strict requirements for their functionality, security, reliability and performance. Approaches for software quality assurance which are currently used in practice help to identify and to fix quite a number of bugs, but none of them allows to detect all possible bugs of kinds sought for. This article shows that different approaches to static verification, which are aimed at solving this task, have significant restrictions if applied to monolithic kernels as a whole, primarily due to a large size and complexity of source code that is constantly evolving. As a first step towards static verification of operating system monolithic kernels a method is proposed for decomposition of kernels into subsystems.

Keywords: operating system; monolithic kernel; microkernel; software quality; static verification; formal specification; program decomposition.

DOI: 10.15514/ISPRAS-2017-29(2)-4

For citation: Novikov E.M. Static verification of operating system monolithic kernels. Trudy ISP RAN/Proc. ISP RAS, volume 29, issue 2, 2017, pp. 97-116 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-4

References

[1]. V.E. Karpov, K.A. Kon'kov. Fundamentals of operating systems. Kurs lekcij. Uchebnoe posobie [Lectures. Study material]. M.: Internet-universitet informacionnyh tehnologij, 536 p., 2005 (in Russian).

[2]. E.M. Novikov. Evolution of the Linux kernel. Trudy ISP RAN/Proc. ISP RAS, volume 29, issue 2, 2017, pp. xx-xx (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-3

[3]. R.L. Glass. Facts and fallacies of software engineering. Addison-Wesley Professional, 2002.

[4]. J. Corbet, G. Kroah-Hartman. Linux kernel development. How Fast It is Going, Who is Doing It, What They Are Doing and Who is Sponsoring the Work. http://go.linuxfoundation.org/linux-kernel-development-report-2016, 2016.

[5]. A.V. Cyvarev, V.A. Martirosjan. Testing of Linux File System Drivers. Trudy ISP RAN/Proc. ISP RAS, volume 23, pp. 413-426, 2012 (in Russian). DOI: 10.15514/ISPRAS-2012-23-24

[6]. J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, A. Fedorova. The Linux scheduler: a decade of wasted cores. In Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16), article 1, 16 pages, 2016.

[7]. A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. An empirical study of operating systems errors. In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP'01), pp. 73-88, 2001.

[8]. J.L. Lawall, J. Brunel, N. Palix, H.R. Rydhof, H. Stuart, G. Muller. WYSIWIB: A declarative approach to finding API protocols and bugs in Linux code. Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 43-52, 2009.

[9]. A. Avetisjan, A. Belevancev, A. Borodin, V. Nesov. Using static analysis for finding security vulnerabilities and critical errors in source code. Trudy ISP RAN/Proc. ISP RAS, volume 21, pp. 23-38, 2011 (in Russian).

[10]. N. Palix, G. Thomas, S. Saha, C. Calves, J. Lawall, G. Muller. Faults in Linux: ten years later. In Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11), pp. 305-318, 2011.

[11]. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, S. Winwood. seL4: formal verification of an OS kernel. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP'09), pp. 207-220, 2009.

[12]. J. Andronick, D. Greenaway, K. Elphinstone. Towards proving security in the presence of large untrusted components. In Proceedings of the 5th international conference on Systems software verification (SSV'10), pp. 9-9, 2010.

[13]. G. Klein. From a verified kernel towards verified systems. In Proceedings of the 8th Asian conference on Programming languages and systems (APLAS'10), pp. 21-33, 2010.

[14]. I. Kuz, G. Klein, C. Lewis, A. Walker. capDL: a language for describing capability-based systems. In Proceedings of the first ACM asia-pacific workshop on Workshop on systems (APSys'10), pp. 31-36, 2010.

[15]. T. Sewell, S. Winwood, P. Gammie, T. Murray, J. Andronick, G. Klein. seL4 enforces integrity. In Proceedings of the Second international conference on Interactive theorem proving (ITP'11), pp. 325-340, 2011.

[16]. T. Murray, D. Matichuk, M. Brassil, P. Gammie, G. Klein. Noninterference for operating system kernels. In Proceedings of the Second international conference on Certified Programs and Proofs (CPP'12), pp. 126-142, 2012.

[17]. M. Daum, N. Billing, G. Klein. Concerned with the unprivileged: user programs in kernel refinement. Formal Aspects of Computing, vol. 26, issue 6, pp. 1205-1229, 2014.

[18]. G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, G. Heiser. Comprehensive formal verification of an OS microkernel. ACM Transactions on Computer Systems (TOCS), vol. 32, issue 1, 70 pages, 2014.

[19]. C. Baumann, B. Beckert, H. Blasum, T. Bormer. Formal Verification of a Microkernel Used in Dependable Software Systems. In Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security (SAFECOMP'09), pp. 187-200, 2009.

[20]. E. Alkassar, W. J. Paul, A. Starostin, A. Tsyban. Pervasive verification of an OS microkernel: inline assembly, memory consumption, concurrent devices. In Proceedings of the Third international conference on Verified software: theories, tools, experiments (VSTTE'10), pp. 71-85, 2010.

[21]. C. Baumann, T. Bormer, H. Blasum, S. Tverdyshev. Proving Memory Separation in a Microkernel by Code Level Verification. In Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW '11), pp. 25-32, 2011.

[22]. S. Schmaltz, A. Shadrin. Integrated semantics of intermediate-language c and macro-assembler for pervasive formal verification of operating systems and hypervisors from VerisoftXT. In Proceedings of the 4th international conference on Verified Software: theories, tools, experiments (VSTTE'12), pp. 18-33, 2012.

[23]. R. Gu, J. Koenig, T. Ramananandro, Z. Shao, X. Wu, S.-C. Weng, H. Zhang, Y. Guo. Deep Specifications and Certified Abstraction Layers. SIGPLAN Not. 50, 1, pp. 595-608, 2015.

[24]. M. Děcký. A road to a formally verified general-purpose operating system. In Proceedings of the First international conference on Architecting Critical Systems (ISARCS'10), pp. 72-88, 2010.

[25]. H. Mai, E. Pek, H. Xue, S. T. King, P. Madhusudan. Verifying security invariants in ExpressOS. In Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems (ASPLOS '13), pp. 293-304, 2013.

[26]. J.F. Ferreira, C. Gherghina, G. He, S. Qin, W.-N. Chin. Automated verification of the FreeRTOS scheduler in Hip/Sleek. Int. J. Softw. Tools Technol. Transf. 16, 4, pp. 381-397, 2014.

[27]. A. Gotsman, H. Yang. Modular verification of preemptive OS kernels. In Proceedings of the 16th ACM SIGPLAN international conference on Functional programming (ICFP '11), pp. 404-417, 2011.

[28]. J. Yang, Chris Hawblitzel. Safe to the last instruction: automated verification of a type-safe operating system. In Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '10), pp. 99-110, 2010.

[29]. M. Danish, H. Xi. Operating system development with ATS: work in progress. In Proceedings of the 4th ACM SIGPLAN workshop on Programming languages meets program verification (PLPV'10), pp. 9-14, 2010.

[30]. M. Danish, H. Xi. Using Lightweight Theorem Proving in an Asynchronous Systems Context. In Proceedings of the 6th International Symposium on NASA Formal Methods, vol. 8430, pp. 158-172. 2014.

[31]. B.W. Cronkite-Ratcliff. Development of automatically verifiable systems using data representation synthesis. In Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity (SPLASH '13), pp. 109-110, 2013.

[32]. K.R.M. Leino. Developing verified programs with dafny. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13), pp. 1488-1490, 2013.

[33]. A. DeHon, B. Karel, T. F. Knight, Jr., G. Malecha, B. Montagu, R. Morisset, G. Morrisett, B. C. Pierce, R. Pollack, S. Ray, O. Shivers, J. M. Smith, G. Sullivan. Preliminary design of the SAFE platform. In Proceedings of the 6th Workshop on Programming Languages and Operating Systems (PLOS '11), article 4, 5 pages, 2011.

[34]. A. Azevedo de Amorim, N. Collins, A. DeHon, D. Demange, C. Hriţcu, D. Pichardie, B. C. Pierce, R. Pollack, A. Tolmach. A verified information-flow architecture. In Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'14), pp. 165-178, 2014.

[35]. T. Ball, E. Bounimova, R. Kumar, V. Levin. SLAM2: Static driver verification with under 4% false alarms. In Proceedings of the 10th International Conference on Conference on Formal Methods in Computer-Aided Design (FMCAD'10), pp. 35-42, 2010.

[36]. D. Beyer. Status Report on Software Verification (Competition Summary SV-COMP 2014). In Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and of Analysis Systems (TACAS'14), LNCS 8413, pp. 373-388, 2014.

[37]. D. Beyer. Software Verification and Verifiable Witnesses (Report on SV-COMP 2015). In Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and of Analysis Systems (TACAS'15), LNCS 9035, pp. 401-416, 2015.

[38]. T. Ball, V. Levin, S.K. Rajamani. A decade of software model checking with SLAM. Communications of the ACM, vol. 54, issue 7, pp. 68-76, 2011.

[39]. T. Witkowski, N. Blanc, D. Kroening, G. Weissenbacher. Model checking concurrent Linux device drivers. In Proceedings of the 22nd IEEE/ACM international conference on Automated Software Engineering (ASE'07), pp. 501-504, 2007.

[40]. H. Post, W. Küchlin. Integrated static analysis for Linux device driver verification. In Proceedings of the 6th International Conference on Integrated Formal Methods (IFM'07), LNCS, vol. 4591, pp. 518-537, 2007.

[41]. D. Bejer, A.K. Petrenko. Linux Driver Verification. Trudy ISP RAN/Proc. ISP RAS, volume 23, pp. 405-412, 2012 (in Russian). DOI: 10.15514/ISPRAS-2012-23-23

[42]. I.S. Zakharov, M.U. Mandrykin, V.S. Mutilin, E.M. Novikov, A.K. Petrenko, A.V. Khoroshilov. Configurable toolset for static verification of operating systems kernel

modules. Programming and Computer Software, volume 41, issue 1, pp 49–64, 2015. DOI: 10.1134/S0361768815010065

[43]. T. Ströder, C. Aschermann, F. Frohn, J. Hensel, J. Giesl. AProVE: Termination and Memory Safety of C Programs (Competition Contribution). In Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15), LNCS 9035, pp. 417-419, 2015.

[44]. M. Kotoun, P. Peringer, V. Šoková, T. Vojnar. Optimized PredatorHP and the SV-COMP Heap and Memory Safety Benchmark. In Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16), vol. 9636, pp. 942-945, 2016.

[45]. D. Engler, M. Musuvathi. Static analysis versus model checking for bug finding. In Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'04), LNCS, vol. 2937, pp. 191-210, 2004.

[46]. T. Ball, S.K. Rajamani. SLIC: A specification language for interface checking of C. Technical Report MSR-TR-2001-21, Microsoft Research, 2001.

[47]. E.M. Novikov. Development of Contract Specifications Method for Verification of Linux Kernel Modules. PhD thesis, Institute for System Programming of the Russian Academy of Sciences, 2013.

[48]. A. Khoroshilov, V. Mutilin, E. Novikov, I. Zakharov. Modeling Environment for Static Verification of Linux Kernel Modules. In Proceedings of the 9th International Ershov Informatics Conference (PSI'14), LNCS 8974, pages 400-414, 2014. DOI: 10.1007/978-3-662-46823-4_32

Управление данными: 25 лет прогнозов

С.Д. Кузнецов <kuzloc@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, дом 25
Московский государственный университет имени М.В. Ломоносова,
119991 ГСП-1 Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус, факультет ВМК
Московский физико-технический институт,
141700, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. В октябре 2013 г. состоялась восьмая встреча исследователей в области баз данных. Первая подобная встреча прошла в феврале 1988 г., так что между ними прошло 25 лет. После каждой встречи публиковался отчет, содержащий обзор современного состояния области и программу исследований на ближайшее будущее – своего рода набор прогнозов развития исследовательской деятельности. В этой статье рассматриваются наиболее интересные прогнозы из отчетов о встречах исследования, обсуждается, насколько они оказались обоснованными, в какой мере сбылись или не сбылись. В числе рассматриваемых разнородных вопросов технологии баз данных содержатся следующие: роль специализированной аппаратуры при построении эффективных СУБД; SQL и приложения баз данных; перспективы объектно-реляционных расширений; распределенные неоднородные системы баз данных; базы данных и Web; базы и хранилища данных, OLAP и data mining; компонентная организация СУБД; критерии оптимизации запросов; самонастраиваемость и самоуправляемость СУБД; архитектура СУБД и новые аппаратные возможности: SSD, энергонезависимая память, массивно-многопоточные процессоры; специализированные СУБД; пространства данных; проблема Больших Данных и реакция на нее в сообществе баз данных; изменения в архитектуре компьютерных систем.

Ключевые слова: отчеты исследовательского сообщества баз данных; технологические прогнозы; анализ данных; машина баз данных; SQL; манифесты будущих систем баз данных; масштабируемость; неоднородность; распределенность; хранилище данных; расширяемость; оптимизация запросов; самонастраиваемость; SSD; энергонезависимая память; пространство данных; Большие Данные

DOI: 10.15514/ISPRAS-2017-29(2)-5

Для цитирования: Кузнецов С.Д. Управление данными: 25 лет прогнозов. Труды ИСП РАН, том 29, вып. 2, 2017 г., стр. 117-160. DOI: 10.15514/ISPRAS-2017-29(2)-5

1. Введение

Раз в несколько лет ведущие представители исследовательского сообщества баз данных проводят встречи, которые обычно длятся два дня. На этих встречах обсуждается и оценивается состояние дел в области баз данных и формулируются темы исследований, которые, по мнению участников, будут наиболее актуальны в ближайшие годы. По результатам встреч принято подготавливать и публиковать отчет. Такие отчеты пользуются высоким авторитетом в сообществе баз данных и оказывают серьезное влияние на развитие исследований и разработок.

К настоящему времени состоялись восемь таких встреч:

- Future Directions in DBMS Research, 1988 г., Лагуна Бич, Калифорния [1];
- NSF Invitational Workshop on the Future of Database Systems Research, Пало Альто, Калифорния, 1990 [3];
- NSF Workshop on the Future of Database Systems Research, Пало Альто, Калифорния, 1995 [4];
- Workshop on Strategic Directions in Computing Research, Кембридж, шт. Массачусетс, 1996 [5];
- Asilomar Meeting, Асиломар, неподалеку от г. Монтерей, Калифорния, 1998 [6];
- Lowell Meeting, Лоуэлл, шт. Массачусетс, 2003 [7];
- Claremont Meeting, Беркли, Калифорния, отель Claremont Resort, 2008 [9];
- Beckman Meeting, Ирвин, Калифорния, Бекманский университетский центр, 2013 [10].

В табл. 1 собрана информация о времени и месте проведения этих встреч, их участниках, официальных и неофициальных публикациях отчетов.

Табл. 1. Сводная информация о прошедших встречах

Table 1. Summary information on meetings of database research community

Время и место встречи	Список участников на английском языке	Список участников на русском языке	Официальная публикация	Публикация на русском языке
1. 4-5 февраля 1988 г. Лагуна Бич, шт. Калифорния «Laguna Beach meeting»	1. Philip A. Bemstein 2. Umeshwar Dayal 3. David J. DeWitt 4. Dieter Gawlick 5. Jim Gray 6. Matthias Jarke 7. Bruce G. Lindsay 8. Peter C. Lockemann 9. David Maier 10. Erich J. Neuhold 11. Andreas Reuter 12. Lawrence A. Rowe 13. Hans J. Schek 14. Joachim W. Schmidt	1. Филипп Бернштейн 2. Умешвар Дайал 3. Дэвид Девитт 4. Дитер Гавлик 5. Джим Грей 6. Маттиас Ярке 7. Брюс Линдсей 8. Питер Локман 9. Дэвид Майер 10. Эрик Ньюхолд 11. Андреас Рейтер 12. Лоуренс Роув 13. Ханс-Йорг Шек 14. Йохим Шмидт	Future Directions in DBMS Research - The Laguna Beach Participants. ACM SIGMOD Record, 18(1):17-26, 1989	Сергей Кузнецов. Будущие направления исследований в области баз данных: десять лет спустя, 1999, http://citforum.ru/database/articles/future_01.shtml

	15. Michael Schrefl 16. Michael Stonebraker	15. Михаэль Шрефл 16. Майкл Стоунбрейкер		
2. 22-24 февраля 1990 г. Пало Альто, шт. Калифорния. NSF Invitational Workshop on the Future of Database System Research «Lagunita meeting 1»	1. Michael Brodie 2. Peter Buneman 3. Mike Carey 4. Ashok Chandra 5. Hector Garcia-Molina 6. Jim Gray 7. Ron Fagin 8. Dave Lomet 9. Dave Maier 10. Marie Ann Niemat 11. Avi Silberschatz, 12. Michael Stonebraker 13. Irv Traiger 14. Jeff Ullman 15. Gio Wiederhold 16. Carlo Zaniolo 17. Maria Zemankova.	1. Майкл Броуди 2. Питер Бьюнман 3. Майк Кэри 4. Ашок Чандра 5. Гектор Гарсиа-Молина 6. Джим Грей 7. Рон Фейджин 8. Дейв Ломе 9. Дейв Майер 10. Мэри Энн Наймэт 11. Эви Зильбершац 12. Майкл Стоунбрейкер 13. Ирв Трейджер 14. Джеф Улльман 15. Джо Видерхолд 16. Карло Заниоло 17. Мария Земанкова	Avi Silberschatz, Michael Stonebraker, Jeff Ullman, editors. Database Systems: Achievements and Opportunities Comm. of the ACM, 34(10):110-120, 1991	
3. 26-27 мая 1995 г. Пало Альто, шт. Калифорния. NSF Workshop on the Future of Database. Systems Research «Lagunita meeting 2»	1. Phil Bernstein 2. Ron Brachman 3. Mike Carey 4. Rick Cattel 5. Hector Garcia-Molina 6. Laura Haas 7. Dave Maier 8. Jeff Naughton 9. Michael Schwartz 10. Pat Selinger 11. Avi Silberschatz 12. Mike Stonebraker 13. Jeff Ullman 14. Patrick Valduriez 15. Moshe Vardi 16. Jennifer Widom 17. Gio Wiederhold 18. Marianne Winslett 19. Maria Zemankova	1. Филипп Бернштейн 2. Рон Брахман 3. Майкл Кэри 4. Рик Каттел 5. Гектор Гарсиа-Молина 6. Лаура Хаас 7. Дейв Майер 8. Джефф Нотон 9. Майкл Шварц 10. Пат Селинджер 11. Эви Зильбершац 12. Майк Стоунбрейкер 13. Джефф Ульман 14. Патрик Вальдурец 15. Мойше Варди 16. Дженифер Вайдом 17. Джо Видерхолд 18. Марианна Винслетт 19. Мария Земанкова	Avi Silberschatz, Mike Stonebraker, Jeff Ullman. Database Research: Achievements and Opportunities into the 21st Century. ACM SIGMOD Record, 25(1):52-63, 1996	Эви Зильбершац, Майк Стоунбрейкер, Джефф Ульман. Базы данных: достижения и перспективы на пороге 21-го столетия. Новая редакция: Сергей Кузнецов, 2009 г., http://citforum.ru/database/classics/nfs_report/
4. 14-15 июня 1996 г. MIT, Кембридж, шт. Массачусетс ACM Workshop on Strategic Directions in Computing Research «Cambridge meeting»	1. Jose Blakeley 2. Peter Buneman 3. Umesh Dayal 4. Tomasz Imielinski 5. Sushil Jajodia 6. Hank Korth 7. Guy Lohman, 8. Dave Lomet 9. Dave Maier 10. Frank Manola 11. Tamer Ozsu 12. Raghu Ramakrishnan 13. Krithi Ramamritham 14. Hans Schek 15. Avi Silberschatz 16. Rick Snodgrass 17. Jeff Ullman 18. Jennifer Widom, 19. Stan Zdonik	1. Хосе.Блейкли 2. Питер .Бьюнман 3. Умеш.Дайал 4. Томаш.Имилинский 5. Сущил.Джаджодиа 6. Хэнк.Корт 7. Гай.Лохман 8. Дейв.Ломе 9. Дейв Майер 10. Френк.Манола 11. Тамер.Оззу 12. Раджу Рамакришнан 13. Крити.Рамамритан 14. Ганс.Шек 15. Эви.Зильбершац 16. Рик Снодгрясс 17. Джефф Ульман 18. Дженифер.Вайдом 19. Стенли.Здоник	Avi Silberschatz, Stan Zdonik, et al. «Strategic Directions in Database Systems: Breaking Out of the Box». ACM Computing Surveys, 28(4):764-778, 1996	А.Зильбершац, С.Здоник и др. Стратегические направления в системах баз данных Перевод: М.Р. Когаловский Новая редакция: Сергей Кузнецов, 2009 г., http://citforum.ru/database/classics/nsf_report2/
5. 19-21 августа 1998 г. Асиломар, г.	1. Phil Bernstein 2. Michael Brodie 3. Stefano Ceri	1. Филипп Бернштейн 2. Майкл Броуди 3. Стефано Чери	Phil Bernstein, Michael Brodie et al. The	Филипп Бернштейн, Майкл Броуди и

Пасифик Гров, шт. Калифорния «Asilomar meetong»	4. David DeWitt 5. Mike Franklin 6. Hector Garcia-Molina 7. Jim Gray 8. Jerry Held 9. Joe Hellerstein 10. H. V. Jagadish 11. Michael Lesk 12. Dave Maier 13. Jeff Naughton 14. Hamid Pirahesh 15. Mike Stonebraker 16. Jeff Ullman	4. Дэвид Девитт 5. Майк Франклин 6. Гектор Гарсиа-Молина 7. Джим Грей 8. Джерри Хелд 9. Джо Хеллерштейн 10. Х.В. Ягадиш 11. Майкл Леск 12. Дейв Майер 13. Джефф Нотон 14. Хамид Пирамеш 15. Майк Стонбрейкер 16. Джефф Ульман	Asilomar Report on Database Research. ACM SIGMOD Record, 27(4):74-80, 1998	др. Асиломарский отчет об исследованиях в области баз данных. Перевод: Сергей Кузнецов, 1999, http://citforum.ru/database/digest/asil_01.shtml
6. 4-6 мая 2003 г. Лоуэлл, шт. Массачусетс «Lowell meeting»	1. Serge Abiteboul 2. Rakesh Agrawal 3. Phil Bernstein 4. Mike Carey 5. Stefano Ceri 6. Bruce Croft 7. David DeWitt 8. Mike Franklin 9. Hector Garcia Molina 10. Dieter Gawlick 11. Jim Gray 12. Laura Haas 13. Alon Halevy 14. Joe Hellerstein 15. Yannis Ioannidis 16. Martin Kersten 17. Michael Pazzani 18. Mike Lesk 19. David Maier 20. Jeff Naughton 21. Hans Schek 22. Timos Sellis 23. Avi Silberschatz 24. Mike Stonebraker 25. Rick Snodgrass 26. Jeff Ullman 27. Gerhard Weikum 28. Jennifer Widom 29. Stan Zdonik	1. Серж Абитебуль 2. Ракеш Агравал 3. Филипп Бернштейн 4. Майк Кэри 5. Стефано Чери 6. Брюс Крофт 7. Дэвид Девитт 8. Майк Франклин 9. Гектор Гарсиа-Молина 10. Дитер Гавлик 11. Джим Грей 12. Лаура Хаас 13. Элон Хэлеви 14. Джо Хеллерштейн 15. Янис Ионнидис 16. Мартин Керстен 17. Майкл Паззани 18. Майк Леск 19. Дэвид Мейер 20. Джефф Нотон 21. Ганс Шек 22. Тимос Селлис 23. Эви Зильбершац 24. Майкл Стоунбрейкер 25. Рик Снодграсс 26. Джефф Ульман 27. Герхард Вейкум 28. Дженифер Вайдом 29. Стен Здоник	Serge Abiteboul, Rakesh Agrawal et al. The Lowell Database Research Self-Assessment. Communications of the ACM, 48(5):111-118, 2005	Сергей Кузнецов. Крупные проблемы и текущие задачи исследований в области баз данных, 2005, http://citforum.ru/database/articles/problems/
7. 29-30 мая 2008 г. Беркли, шт. Калифорния, Клермонт Ресорт «Claremont meeting»	1. Rakesh Agrawal 2. Anastasia Ailamaki 3. Philip A. Bernstein 4. Eric A. Brewer 5. Michael J. Carey 6. Surajit Chaudhuri 7. AnHai Doan 8. Daniela Florescu 9. Michael J. Franklin 10. Hector Garcia Molina 11. Johannes Gehrke 12. Le Gruenwald 13. Laura M. Haas 14. Alon Y. Halevy 15. Joseph Hellerstein 16. Yannis E. Ioannidis 17. Hank F. Korth 18. Donald Kossmann 19. Samuel Madden 20. Roger Magoulas	1. Ракеш Агравал 2. Анастасия Айламаки 3. Филипп Бернштейн 4. Эрик Брюер 5. Майкл Кэри 6. Сураджит Чаудхари 7. Анхай Доан 8. Даниела Флореску 9. Майкл Франклин 10. Гектор Гарсиа-Молина 11. Иоханнес Герке 12. Ле Грюнвальд 13. Лаура Хаас 14. Элон Хэлеви 15. Джозеф Хеллерштейн 16. Янис Ионнидис 17. Хэнк Корт 18. Дональд Коссманн 19. Сэмуэль Мэдден 20. Роджер Магулас	Rakesh Agrawal, Anastasia Ailamaki,et al. The Claremont Report on Database Research. Communications of the ACM, 52(6):56-65, 2009.	Ракеш Агравал, Анастасия Айламаки др. Клермонтский отчет об исследованиях в области баз данных. Пересказ и комментарии: Сергей Кузнецов, 2008, http://citforum.ru/database/articles/claremont_report/

	21. Beng Chin Ooi 22. Tim O'Reilly 23. Raghu Ramakrishnan 24. Sunita Sarawagi 25. Michael Stonebraker 26. Alexander S. Szalay 27. Gerhard Weikum	21. Бен Чин Ой 22. Тим О'Рейли 23. Раджу Рамакришнан 24. Сунита Сарагави 25. Майкл Стоунбрейкер 26. Александр Шалай 27. Герхард Вейкум		
8. 14-15 октября 2013 г. Ирвин, шт. Калифорния, Бекманский центр университета в Ирвине «Beckman meeting»	1. Daniel Abadi 2. Rakesh Agrawal 3. Anastasia Ailamaki 4. Magdalena Balazinska 5. Philip A. Bernstein 6. Michael J. Carey 7. Surajit Chaudhuri 8. Jeffrey Dean 9. AnHai Doan 10. Michael J. Franklin 11. Johannes Gehrke 12. Laura M. Haas 13. Alon Y. Halevy 14. Joseph Hellerstein 15. Yannis E. Ioannidis 16. H.V. Jagadish 17. Donald Kossmann 18. Samuel Madden 19. Sharad Mehrotra 20. Tova Milo 21. Jeffrey F. Naughton 22. Raghu Ramakrishnan 23. Volker Markl 24. Christopher Olston 25. Beng Chin Ooi 26. Christopher Re 27. Dan Suciu 28. Michael Stonebraker 29. Todd Walter 30. Jennifer Widom	1. Дэниел Абади 2. Ракеш Агравал 3. Анастасия Айламаки 4. Магдалена Балазинска 5. Филип А. Бершстейн 6. Майкл Дж. Кэри 7. Сураджит Чаудхари 8. Джеффри Дин 9. Анхай Доан 10. Майкл Дж. Франклин 11. Йоханнес Герке 12. Лаура М. Хаас 13. Элон И. Хэлеви 14. Джозеф Хеллерштейн 15. Янис Е. Ионнидис 16. Х.В. Ягадиш 17. Дональд Коссманн 18. Самуэль Мэдден 19. Шарад Мехротра 20. Това Мило 21. Джеффри Нотон 22. Раджу Рамакришнан 23. Волкер Маркл 24. Кристофер Олстон 25. Бен Чин Ой 26. Кристофер Ре 27. Дан Сучиу 28. Майкл Стоунбрейкер 29. Тодд Валтер 30. Джерифер Вайдом	Daniel Abadi, Rakesh Agrawa et al. The Beckman Report on Database Research. SIGMOD Record, September, 43(3):61-70, 2014	Дэниел Абади, Ракеш Агравал и др. Бекманский отчет об исследований в области баз данных. Перевод: Сергей Кузнецов, 2017

В разные годы во встречах принимали участие разные специалисты, но некоторые известные исследователи внесли особенно заметный вклад в эту работу. В табл. 2 перечислены исследователи, принявшие участие в двух и большем числе встреч.

Табл. 2. Наиболее активные участники встреч
Table 2. The most active participants

Число встреч	Имя участника	В каких встречах участвовал
7	Майкл Стоунбрейкер	1, 2, 3, 5, 6, 7, 8
6	Дэвид Майер	1, 2, 3, 4, 5, 6
	Филип Берштейн	1, 3, 5, 6, 7, 8
5	Гектор Гарсиа-Молина	2, 3, 5, 6, 7
	Джеффри Ульман	2, 3, 4, 5, 6,
	Майкл Кэри	2, 3, 6, 7, 8
4	Эви Зильбершац	2, 3, 4, 6
	Джеффри Нотон	3, 5, 6, 8

	Дженифер Вайдом	3, 4, 6, 8
	Джим Грей (пропал без вести в 2007 г.)	1, 2, 5, 6
	Джо Хеллерштейн	5, 6, 7, 8
	Лаура Хаас	3, 6, 7, 8
	Майкл Франклин	5, 6, 7, 8
3	Дэвид Девитт	1, 5, 6
	Ганс Шек	1, 4, 6
	Раджу Рамакришнан	4, 7, 8
	Ракеш Агравал	6, 7, 8
	Янис Ионнидис	6, 7, 8
2	Элон Хэлеви	7, 8
	Анастасия Айламаки	7, 8
	Анхай Доан	7, 8
	Бен Чин Ой	7, 8
	Дэвид Ломе	2, 4
	Дитер Гавлик	1, 6
	Дональд Коссманн	7, 8
	Герхард Вейкум	6, 7
	Джо Видерхолд	2, 3
	Х.В. Ягадиш	5, 6
	Хэнк Корт	4, 7
	Йоханнес Герке	7, 8
	Мария Земанкова	2, 3
	Майкл Броуди	2, 5
	Питер Бьюнман	2, 4
	Рик Снодграсс	4, 6
	Сэмуэль Мэдден	7, 8
	Стенли Здоник	4, 6
	Стефано Чери	5, 6
	Сураджит Чаудхари	7, 8
	Умешвар Дайал	1, 4

Легко видеть, что между первой и последней встречами прошло 25 лет, так что Бекманская встреча была юбилейной. В честь этого юбилея с осознанием важности для мирового исследовательского сообщества регулярных отчетов о перспективных исследованиях в области баз данных я решил написать эту статью. В некотором смысле статья должна предоставить читателям общую картину развития технологии баз данных за четверть века.

В статье обсуждаются самые интересные, с точки зрения автора, прогнозы, содержащиеся в отчетах о встречах прошлых лет [1–9], насколько они были реалистичными, точными, деловыми и прагматичными или, наоборот, утопичными, конъюнктурными или маркетинговыми, какие прогнозы сбылись, а какие канули в вечность и т.д. Почти не затрагивается отчет [10] о последней по счету встрече, состоявшейся осенью 2013 г. в Калифорнии, оценивать точность прогнозов которой еще не пришло время.

2. Встреча в Лагуна Бич: 1988

Первая встреча состоялась 4-5 февраля 1988 г. в г. Лагуна Бич, шт. Калифорния. Вот фрагмент табл. 1, описывающей детали этой встречи.

Время и место встречи	Список участников на английском языке	Список участников на русском языке	Официальная публикация	Публикация на русском языке
4-5 февраля 1988 г. Лагуна Бич, шт. Калифорния «Laguna Beach meeting»	17. Philip A. Bemstein 18. Umeshwar Dayal 19. David J. DeWitt 20. Dieter Gawlick 21. Jim Gray 22. Matthias Jarke 23. Bruce G. Lindsay 24. Peter C. Lockemann 25. David Maier 26. Erich J. Neuhold 27. Andreas Reuter 28. Lawrence A. Rowe 29. Hans J. Schek 30. Joachim W. Schmidt 31. Michael Schrefl 32. Michael Stonebraker	17. Филипп Бернштейн 18. Умешвар Дайал 19. Дэвид Девитт 20. Дитер Гавлик 21. Джим Грей 22. Маттиас Ярке 23. Брюс Линдсей 24. Питер Локман 25. Дэвид Майер 26. Эрик Ньюхолд 27. Андреас Рейтер 28. Лоуренс Роув 29. Ханс-Йорг Шек 30. Йохим Шмидт 31. Михаэль Шрефл 32. Майкл Стоунбрейкер	Future Directions in DBMS Research - The Laguna Beach Participants. ACM SIGMOD Record, 18(1):17-26, 1989	Сергей Кузнецов. Будущие направления исследований в области баз данных: десять лет спустя, 1999, http://citforum.ru/database/articles/future_01.shtml

Отчет о результатах встречи официально опубликован в 1989 г. [1] (официальные публикации отчетов о встречах, которым посвящена данная статья, обычно появляются позже неофициальных публикаций в виде технических отчетов различных организаций; в частности, исходный общедоступный вариант отчета Лагуна Бич находится на сайте университета Беркли: http://www.icsi.berkeley.edu/pubs/techreports/tr-88-001.pdf, дата доступа 24.03.2017). На русский язык отчет Лагуна Бич полностью не переводился. Подробный пересказ отчета с комментариями можно найти в [2].

Далее в этом разделе будут кратко рассмотрены наиболее интересные (с точки зрения автора) прогнозы, содержащиеся в [1], и то, насколько они сбылись к теперешнему времени или насколько актуальными они остаются сейчас.

2.1. Распространение аналитики

В отчете предсказывалось *«широкое внедрение систем поддержки принятия решений по мере снижения цен аппаратуры»*. Подход к построению «систем поддержки принятия решений» (decision support system, DSS) возник в 70-е гг. прошлого века на основе работ, выполнявшихся еще в 60-е гг. [11]. Термин DSS никогда не имел однозначного и четкого определения, как не имеет его и в настоящее время. Однако уже в конце 80-х гг. специалистам в области баз данных было понятно, что очень во многих случаях принятие решений в различных организациях должно основываться на анализе данных.

В 90-е гг. XX века были развиты теоретические основы и технология хранилищ данных (datawarehouse) и оперативной аналитической обработки данных (online analytical data processing, OLAP). В первом десятилетии XXI века появились горизонтально масштабируемые массивно-параллельные аналитические СУБД, возникла технология map/reduce (см., например, [12]). Тогда же активно развивались (и продолжают развиваться в настоящее время) методы интеллектуального анализа данных (data mining, text mining и т.д.). К анализу активно привлекаются данные, генерируемые различными пользователями Internet, в частности, социальных сетей.

Даже если не затрагивать варианты DSS, напрямую не основанные на анализе данных (например, разного рода экспертные системы), то можно сказать, что прогноз о широком внедрении DSS полностью оправдался. Использование аналитики с целью поддержки принятия решений сегодня повсеместно.

2.2. Малая перспективность специализированной аппаратуры

В отчете отмечалось, что *«в области управления базами данных компьютеры общего назначения более перспективны, чем специализированная аппаратура»*. Заметим, что к 1988 г. стало ясно, что японский проект компьютеров пятого поколения [13] не то чтобы с треском, но таки провалился (по крайней мере, в связи с разработкой специализированной аппаратуры для построения особенно эффективных и мощных СУБД).

У японцев предполагалась разработка двух видов специализированных аппаратных средств для поддержки СУБД: процессоры, в которых операции реляционной алгебры поддерживались на уровне системы команд, и магнитные диски с фиксированными головками, в головки которых должны были встраиваться микропроцессоры для фильтрации данных «на лету» по мере их чтения с дисков («умные» диски).

Первый подход не удался по разным причинам (скорее всего, можно существенно расширить приводимый список):

- поскольку базы данных сохраняются во внешней дисковой памяти, используемые структуры данных оптимизированы в расчете на это; для непосредственного применения команд процессора недостаточно переместить данные с диска в основную память;
- таблицы SQL-ориентированных баз данных (а к началу проекта пятого поколения уже было понятно, что будущее за SQL-ориентированными СУБД) часто настолько велики, что не могут целиком поместиться в основной памяти компьютера; итерационное применение команд «реляционной алгебры» далеко не всегда возможно и/или эффективно;
- наличие алгоритмов выполнения реляционных операций, «зашитых» в машинные команды, противоречит идее гибкой оптимизации

запросов, когда оптимизатор выбирает наиболее подходящий алгоритм в зависимости от текущего состояния базы данных;

- при сохранении баз данных на магнитных дисках основное время выполнения запроса уходит на выполнение обменов с внешней памятью; экономия на времени обработки данных в основной памяти часто бывает несущественной.

Что касается второго подхода, то он не нашел широкого применения, прежде всего, по экономическим причинам. Легко видеть, что объем необходимой аппаратуры в дисковом устройстве с фиксированными головками во много раз больше, чем в традиционных жестких дисках. Поэтому и стоимость таких устройств значительно выше (а надежность – ниже). Кроме того, по разным причинам (в частности, из-за проблем распределения основной памяти) в традиционной архитектуре SQL-ориентированных СУБД предполагается блочный обмен данными с внешним хранилищем баз данных, и поэтому не очень понятно, как СУБД может получить от «умного» диска отфильтрованные на лету данные.

Интересно, что в известной статье 1992 г. [14] Девитт и Грей авторитетно указывали на провал подхода «умных» дисков и предрекали, что будущие высокопроизводительные параллельные СУБД будут по-прежнему опираться на использование дисков с подвижными головками. Но уже 10 лет спустя в своем интервью [15] Джим Грей говорил, что из-за стремительного удешевления аппаратных средств (в частности, мощных микропроцессоров) теперь возможен перенос СУБД целиком на головку магнитного диска. Как видно, эта идея распространения не нашла (думаю, что Джим фантазировал, потому что такая архитектура даже на первый взгляд кажется очень проблематичной).

И вообще, список решений XXI века для управления базами данных, основанных на использовании специализированной аппаратуры, выглядит очень скромно.

Database appliance (программно-аппаратные комплексы для управления базами данных). По определению Gartner [16], database appliance – это заранее собранный и/или сконфигурированный набор оборудования (серверы, память, накопители и каналы ввода/вывода), программное обеспечение (операционная система, СУБД и т.д.), средства обслуживания и поддержки. Определение достаточно расплывчато; в соответствии с ним, database appliance является скорее маркетинговым, а не технологическим понятием, и это, на мой взгляд, отражает сложившуюся действительность. В некоторых продуктах этой категории (наиболее ярким примером является Oracle Exadata Database Machine [17]) используются специализированные программно-аппаратные компоненты, но сами продукты являются универсальными, равно пригодными для обработки и транзакционных, и аналитических рабочих нагрузок. Другой подход исповедует Майкл Стоубрейкер [18]. По его мнению, в состав database appliance не должна входить специализированная

аппаратура, но зато специализированным должно быть программное обеспечение самой СУБД. В любом случае, database appliance никак не похожи на машины баз данных 1980-х.

Использование графических процессоров. Как известно, в настоящее время очень модно (и, наверное, в ряде случаев перспективно) пытаться использовать GPU в областях, отличных от обработки графики. В частности, растет популярность применения GPU для обработки запросов в СУБД, ориентированных на хранение данных в основной памяти (например, [19]). По ряду причин, у меня нет уверенности в перспективности этого подхода (хотя бы потому, что разные графические процессоры слишком различаются, а разработчикам СУБД часто для получения эффективности требуется использовать уникальные возможности GPU). Кроме того, здесь мы имеем дело совсем не со специализированной аппаратурой, разработанной именно для поддержки СУБД.

Можно считать, что прогноз оказался достаточно точным.

2.3 Широкое распространение ОС с микроядерной архитектурой

Действительно, в 1980-е гг. многим (включая меня) казалось, что *будущее за микроядерными операционными системами с более развитыми и современными функциональными возможностями*, чем у тогдашнего (да и теперешнего) фаворита – семейства ОС UNIX. Выполнялись многочисленные проекты, среди которых мне ближе других Chorus [20], CLOS [21] и Mach [22]. По разным причинам большая часть этих проектов не удалась (по крайней мере, те результаты, на которые надеялись разработчики, получить не удалось).

Фактически, обе довлеющие сегодня операционные системы Linux и Windows микроядерными не являются. Как и 20 лет тому назад, единственной коммерческой микроядерной системой является QNX [23], но эта ОС (семейство ОС) является нишевой и не может служить подтверждением «широкого» распространения микроядерного подхода.

Интересную работу в XXI веке провел Эндрю Таненбаум – проект Minix 3 [24]. Это современный, хорошо проработанный проект, который вряд ли ждет широкое практическое использование (хотя бы потому, что в нем реализуется только API Posix). Занятно, что основная разработка Minix 3 завершилась в 2014 г. одновременно с уходом Таненбаума на пенсию.

Так что этот прогноз участников встречи в Лагуна Бич не оправдался.

2.4 Ужасный интерфейс SQL

Участники встречи отмечали, что было бы хорошо *улучшить существующий ужасный интерфейс SQL для встраиваемых и динамических запросов.* Заметим, что это происходило за четыре года до принятия стандарта SQL-92 (SQL:1992), в котором соответствующие возможности были полностью и

окончательно определены. С ранней историей стандартов языка SQL можно ознакомиться, например, в [25].

Идеи встраивания операторов языка SQL в языки программирования и динамической компиляции SQL появились еще на заре SQL в 1970-е гг. во время выполнения в IBM проекта System R (история этого замечательного проекта лучше всего описана в [26]). Причины же появления средств встраивания и динамической компиляции в SQL времени System R кратко и четко описаны в интервью одного из основных разработчиков начального SQL Дона Чемберлина [27].

Конечно, средства встраиваемого и динамического SQL трудно назвать красивыми и/или элегантными (впрочем, с моей точки зрения, то же относится и к языку SQL в целом). Но за время их существования (более 40 лет) ничего принципиально нового придумать не удалось. Широко распространено использование интерфейсов уровня вызова функций ODBC и JDBC. Но эти интерфейсы не избавляют от необходимости сохранять в программе или динамически формировать строки, содержащие тексты требуемых операторов SQL, а именно наличие в программе таких строк делает столь «ужасными» традиционные средства SQL для поддержки встраиваемых и динамически формируемых операторов SQL.

Так что улучшить интерфейс SQL так и не удалось.

3. Первая встреча в Пало Альто: 1990

Вторая встреча исследователей в области баз данных состоялась в Стэндфордском университете (Пало Альто, шт. Калифорния) 22-24 февраля 1990 г. в рамках организованного Национальным научным фондом США симпозиума «Future of Database System Research». Эта и следующая встречи неформально называются «встречами Лагунита», поскольку проходили вблизи одноименного искусственного озера. Вот соответствующий фрагмент табл. 1.

Время и место встречи	Список участников на английском языке	Список участников на русском языке	Официальная публикация
22-24 февраля 1990 г. Пало Альто, шт. Калифорния. NSF Invitational Workshop on the Future of Database System Research «Lagunita meeting 1»	1. Michael Brodie 2. Peter Buneman 3. Mike Carey 4. Ashok Chandra 5. Hector Garcia-Molina 6. Jim Gray 7. Ron Fagin 8. Dave Lomet 9. Dave Maier 10. Marie Ann Niemat 11. Avi Silberschatz, 12. Michael Stonebraker 13. Irv Traiger 14. Jeff Ullman 15. Gio Wiederhold 16. Carlo Zaniolo 17. Maria Zemankova.	1. Майкл Броуди 2. Питер Бьюнман 3. Майк Кэри 4. Ашок Чандра 5. Гектор Гарсиа-Молина 6. Джим Грей 7. Рон Фейджин 8. Дейв Ломе 9. Дейв Майер 10. Мэри Энн Наймэт 11. Эви Зильбершац 12. Майкл Стоунбрейкер 13. Ирв Трейджер 14. Джеф Улльман 15. Джо Видерхолд 16. Карло Заниоло 17. Мария Земанкова	Avi Silberschatz, Michael Stonebraker, Jeff Ullman, editors. Database Systems: Achievements and Opportunities. Communications of the ACM, 34(10):110-120, 1991

Официально отчет опубликован в 1991 г. [3], на русский язык не переводился и не пересказывался. Предварительная неформальная и общедоступная (если не бояться postscript) публикация размещена на infolab.stanford.edu/~hector/lagi.ps, дата доступа 24.03.2017.

Вот самые интересные прогнозы из Lagunita Report.

3.1 Будущие приложения баз данных

В отчете предсказывалось, что *приложения баз данных следующего поколения будут иметь мало общего с сегодняшними приложениями баз данных для обработки бизнес-данных. Они будут затрагивать гораздо больше данных; потребуют поддержки расширений системы типов данных, мультимедийных данных, сложных объектов, обработки правил и архивной системы хранения; в СУБД потребуется переосмысление алгоритмов выполнения большинства операций.*

Для правильной оценки этого прогноза необходимо восстановить общий контекст, в котором проходила первая встреча в Пало Альто. Только что был опубликован Манифест систем объектно-ориентированных баз данных (Первый манифест) [28], авторы которого, по сути, отрицали дальнейшую перспективность технологий реляционных и SQL-ориентированных баз данных, упрекая их в ограниченности и неспособности удовлетворять требования будущих приложений. Готовился ответный Манифест систем баз данных третьего поколения (Второй манифест) [29], в котором не отрицались конструктивные аспекты [28] (в частности, потребность в наличии у будущих СУБД развитой и расширяемой системы типов данных), но утверждалось, что требуемые возможности с СУБД можно получить путем эволюции имеющейся технологии баз данных. Кроме того, у Майкла Стоунбрейкера имелась коммерческая СУБД Illustra, основанная на исследовательском прототипе Postgres, в которой уже поддерживалась бо́льшая часть возможностей, упомянутых в прогнозе (история Postgres-Illustra хорошо описана Стоунбрейкером в [30]).

На встрече присутствовали представители обоих лагерей. Поэтому вполне естественно появление в отчете прогноза в обтекаемой формулировке, отражающей общие взгляды исследователей из реляционного и объектно-ориентированного миров и не выпячивающей их принципиальные противоречия. Несмотря на обтекаемость формулировки прогноза, в ней заметно влияние Стоунбрейкера, который с должным основанием всегда имел авторитет в сообществе баз данных.

К началу XXI века после драматических событий 1990-х, когда ведущие вендоры SQL-ориентированных СУБД приняли концепцию объектно-реляционных баз данных, и был выпущен стандарт SQL:1999, в котором эта концепция была затвержена, технология объектно-ориентированных СУБД стала стремительно терять популярность. Сегодня, несмотря на неоднократные попытки возродить притягательность ООСУБД для широкой

аудитории разработчиков баз данных и их приложений, эти системы являются нишевыми. С другой стороны, в современных ведущих SQL-ориентированных СУБД реализованы практически все возможности, отсутствие которых мотивировало авторов [28]. Можно сказать, что Второй манифест победил. Можно ли поэтому сказать, что прогноз, обсуждаемый в этом подразделе, полностью сбылся?

Безусловно, сегодняшние приложения баз данных в среднем работают с гораздо большими объемами данных, чем в начале 1990-х. Разработчики баз данных и их приложений имеют возможности определять собственные типы данных и произвольно сложной внутренней структурой и поведением. Но похоже, что сегодня, как и 10 лет тому назад [31], пользователи не слишком стремятся определять собственные типы данных внутри баз данных. Скорее, можно говорить о том, что определяемые пользователями типы данных применяют сами вендоры SQL-ориентированных СУБД для расширения их функциональных возможностей. Так что оценка потребности разработчиков баз данных и их приложений в расширяемой системе типов и сложных объектах оказалась преувеличенной.

Сложная система правил осталась прерогативой Postgres (теперь еще и PostgreSQL), а в подавляющем большинстве современных СУБД правила используются в объеме определенного в стандарте SQL механизма триггеров. Поддержка мультимедийных данных реализуется вендорами СУБД за счет наличия стандартного типа BLOB и дополнительных типов данных, реализуемых с помощью стандартного механизма определения типов данных. Наличие архивной системы хранения предполагалось в Postgres, но в современных SQL-ориентированных СУБД такая система явным образом отсутствует. Наконец, мне неизвестно, чтобы какой-либо производитель SQL-ориентированных СУБД коренным образом изменил алгоритмы выполнения операций.

Можно считать, что прогноз сбылся частично.

3.2 Неоднородность, распределенность, масштабируемость

Участники встречи полагали, что *для кооперации различных организаций при решении научных, технических и коммерческих проблем потребуются масштабируемые, неоднородные, распределенные базы данных. Сложными проблемами технологии распределенных СУБД являются несогласованность баз данных, безопасность и масштабируемость.*

Тематике интеграции неоднородных баз данных, начиная с 1980-х гг., посвящалось множество исследований, результаты которых описывались во множестве публикаций. Здесь речь идет о так называемой виртуальной интеграции разнородных баз данных. В отличие от физической интеграции нескольких баз данных с целью построения хранилища данных, основанной на тщательно разработанных процедурах ETL (Extract, Transform, Load), при виртуальной интеграции дополнительное хранилище данных не создается,

данные извлекаются из разных источников и интегрируются «на лету» при обработке интеграционным сервером запросов к «интегрированной» базе данных. Понятно, что интегрируемые источники данных в общем случае распределены в локальной или глобальной сети, могут поддерживаться различными СУБД и даже основываться на разных моделях данных.

Многолетние исследования привели к разработке многочисленных подходов и методов виртуальной интеграции баз данных (не очень старый и краткий обзор см. в [32]), но ни одна из работ не привела к созданию хотя бы относительно безупречной технологии. Несмотря на наличие на рынке многих интеграционных продуктов, широким спросом они не пользуются, и для сообщества исследователей в области данных тематика интеграции разнородных данных сегодня не является магистральным направлением, хотя потребность в виртуальной интеграции гетерогенных источников данных периодически возникает в разных областях человеческой деятельности. Так что нельзя сказать, что прогноз сбылся, но и нельзя считать его неоправданным.

Вместе с тем, в XXI веке технология распределенных систем смыкается с технологией баз данных в нескольких направлениях, отличных от интеграции данных. Первое направление – специализированные массивно-параллельные SQL-ориентированные СУБД. В первом десятилетии XXI века были разработаны исследовательские прототипы аналитической массивно-параллельной СУБД C-Store с хранением таблиц по столбцам [33] и транзакционной массивно-параллельной СУБД H-Store с хранением данных в основной памяти [34]. Обе эти системы основывались на подходе shared-nothing (без использования общих ресурсов) с целью обеспечения горизонтальной масштабируемости при возрастании числа узлов в системе, обе показали хорошие результаты на соответствующих стандартных тестовых наборах.

Оба проекта были коммерциализированы. На основе результатов проекта C-Store была основана компания Vertica, поглощенная в 2011 г. Hewlett-Packard [35]. Результаты проекта H-Store используются в коммерческом продукте VoltDB, разрабатываемом и поддерживаемом одноименной компанией [36]. Имеется ряд других SQL-ориентированных СУБД, изначально предназначенных для использования в массивно-параллельной среде. Конечно, под массивно-параллельной средой здесь понимаются кластеры, но что такое кластер, как не однородная специализированная локальная сеть? Поэтому в массивно-параллельных архитектурах СУБД активно используются методы распределенных систем: репликация данных, распределенные транзакции и т.д.

Другое направление – глобально распределенные СУБД категории NoSQL. Это направление бурно развивается и непрерывно изменяется, поэтому трудно охарактеризовать его текущее состояние. Тем не менее, думаю, что ряд характеристик, указанных в [37], остается справедливым. Наверное, наиболее общими чертами этого направления является стремление к обеспечению

горизонтальной масштабируемости систем и высокого уровня доступности данных за счет репликации. Доступность обеспечивается за счет отказа строгой транзакционности (в глобально распределенной среде строгая согласованность реплик стоит слишком дорого). Несмотря на общее название направления, в системах все чаще поддерживаются ограниченные диалекты SQL.

4. Вторая встреча в Пало Альто: 1995

Третья по счету встреча исследователей в области баз данных также состоялась в Пало Альто в рамках второго спонсированного Национальным научным фондом США симпозиума Future of Database Systems Research (Лагунита 2). Подробности в приводимом фрагменте табл. 1.

Время и место встречи	Список участников на английском языке	Список участников на русском языке	Официальная публикация	Публикация на русском языке
26-27 мая 1995 г. Пало Альто, шт. Калифорния. NSF Workshop on the Future of Database Systems Research «Lagunita meeting 2»	1. Phil Bernstein 2. Ron Brachman 3. Mike Carey 4. Rick Cattel 5. Hector Garcia-Molina 6. Laura Haas 7. Dave Maier 8. Jeff Naughton 9. Michael Schwartz 10. Pat Selinger 11. Avi Silberschatz 12. Mike Stonebraker 13. Jeff Ullman 14. Patrick Valduriez 15. Moshe Vardi 16. Jennifer Widom 17. Gio Wiederhold 18. Marianne Winslett 19. Maria Zemankova	1. Филипп Бернштейн 2. Рон Брахман 3. Майкл Кэри 4. Рик Каттел 5. Гектор Гарсиа-Молина 6. Лаура Хаас 7. Дейв Майер 8. Джефф Нотон 9. Майкл Шварц 10. Пат Селинджер 11. Эви Зильбершац 12. Майк Стоунбрейкер 13. Джефф Ульман 14. Патрик Вальдурец 15. Мойше Варди 16. Дженифер Вайдом 17. Джо Видерхолд 18. Марианна Винслетт 19. Мария Земанкова	Avi Silberschatz, Mike Stonebraker, Jeff Ullman. Database Research: Achievements and Opportunities into the 21st Century. ACM SIGMOD Record, 25(1):52-63, 1996	Эви Зильбершац, Майк Стоунбрейкер, Джефф Ульман. Базы данных: достижения и перспективы на пороге 21-го столетия. Новая редакция: Сергей Кузнецов, 2009 г., http://citforum.ru/database/classics/nfs_report/

Официально отчет о встрече опубликован в 1996 г. [4]. Неофициальная публикация доступна здесь: http://ilpubs.stanford.edu:8090/81/1/1995-15.pdf, дата обращения 26.03.2017. Имеется перевод отчета на русский язык. Вот что мне кажется наиболее интересным в этом отчете.

4.1 Рассредоточение информации

В 1996 г. уже всем было понятно, что человечество вступило в новую эру Всемирной Паутины. Участники встречи отмечали, что *WWW – это распределенная среда, состоящая из автономных систем, узлы которой все чаще формируются как реляционные базы данных. Наличие этой среды*

заставляет переосмыслить многие концепции существующей технологии распределенных баз данных. Имелась в виду гипотетическая система, поддерживающая обработку запросов к Сети в целом и автоматически определяющая, к каким Web-сайтам следует обратиться для выполнения запроса. Отмечались следующие направления требуемых исследований.

Учет и расчёты. Доступ к информации, хранимых в Web-сайтах, может быть платным. Гипотетическая система при обработке запроса пользователя должна учитывать, во сколько это может обойтись, и должна выбирать (может быть, в ущерб качеству ответа на запрос) тот набор Web-ресурсов, который окажет пользователю по карману. При этом, учитывая, что получение ответа на запрос не должно обходиться пользователю слишком дорого, сама эта служба учета и расчетов должна быть предельно экономичной.

Насколько мне известно, подобная проблема в масштабах Web не решена. Возможно (хотя и маловероятно), когда-нибудь за ее решение возьмется какая-нибудь крупная Internet-компания, например, Google. Однако имеются решения сходных проблем в контексте сервис-ориентированных архитектур (Data as a Service, [38]).

Безопасность и конфиденциальность. Как отмечалось в отчете, требуется *разработка исключительно гибких систем аутентификации и авторизации, поддерживающих доступ на основе разнообразных «ролей», исполняемых пользователями (например, один и тот же индивид может выступать в роли лечащего врача некоторого пациента, в роли «врача вообще» или в роли частного лица).* Тематика исследований и разработок в области безопасности и конфиденциальности в Internet чрезвычайно широка. Важность поддержки конфиденциальности данных многократно возросла в связи с появлением социальных сетей. Однако мне не приходилось слышать о защите конфиденциальности данных в масштабе Internet на основе ролей.

Репликация и согласование данных. Вот еще одна цитата из отчета: *из соображений эффективности данные часто реплицируются на нескольких узлах. Когда все эти узлы связаны сетью, можно поддерживать идентичность копий. Однако в ситуациях, когда связь нарушается, в копиях могут появиться различия. После восстановления связи должен включаться механизм согласования, который должен согласовать все копии и сформировать одну новую копию, отражающую все сделанные изменения. … В новой информационной среде подобные ситуации становятся уже не исключением, а нормой.* Особенности распределенных систем категории NoSQL показывают, что достаточно эффективные механизмы согласования копий так и не появились. Следуя теореме CAP Эрика Брюера [39], разработчики этих систем обеспечивают высокий уровень доступности в ущерб обеспечения идентичности реплик.

Не полностью структурированные и неструктурированные данные. Участники встречи отдавали себе отчет, что у большинства данных в Web имеется в лучшем случае нечеткая, динамически изменяемая схема, а в худшем случае данные вообще не структурированы. Поэтому сообществу баз

данных *очень важно расширить механизмы индексации и другие средства поддержки поиска для хорошо структурированных данных и адаптировать их к неструктурированному миру Web*. Естественно, такая работа была нужна, но выполнило (и продолжает выполнять) ее не сообщество баз данных. Поиск информации в Web поддерживают универсальные и специализированные поисковые системы, основываясь на результатах активно развивающихся областей анализа текстов и семантического поиска. Из средств управления базами данных в лучшем случае используются СУБД категории NoSQL (в частности, системы «ключ-значение»).

4.2 Новые применения баз данных

Как говорится в отчете, *образовались новые важные области применения баз данных, и каждая из них представляет принципиально новую среду, к которой необходимо адаптировать технологии СУБД: интеллектуальный анализ данных (data mining), хранилища данных (data warehousing), репозитарии данных (data repository)*.

Интеллектуальный анализ данных. Отмечалась потребность в исследованиях по следующим направлениям: *методы оптимизации сложных запросов, включающих агрегацию и группирование; поддержка «многомерных» запросов, относящихся к данным, организованным в виде «куба», в ячейках которого находятся интересующие данные; языки запросов очень высокого уровня, а также интерфейсы для поддержки пользователей, не являющихся экспертами, которым нужны ответы на нерегламентированные запросы*.

Здесь, прежде всего, нужно отметить имевшуюся на встрече путаницу между понятиями data mining и OLAP (On-Line Analytical Processing). Агрегация и группирование – это действия, требуемые для формирования многомерного куба, а анализ данных на основе их представления в виде куба – это OLAP. Так что в этом пункте речь идет об OLAP, а не о data mining.

Практически в то же время, когда проходила вторая встреча в Пало Альто, Джим Грей с коллегами придумали метод и соответствующие расширения языка SQL, позволяющие эффективно строить многомерный куб на основе табличного представления данных [40]. Эта работа оказала решающая воздействие на развитие технологии ROLAP (Relational OLAP), и ее принято считать одним из высших достижений Джима Грея. Эта же работа позволила создать новые методы оптимизации запросов с группировкой и агрегацией.

Годом позже в недрах компании Microsoft появился язык запросов в многомерном кубе MDX (MultiDimensional eXpressions) [41], основным разработчиком которого являлся Моша Пасуманский. Лично мне этот язык кажется еще более «ужасным», чем SQL, но распространено мнение, что он близок аналитикам. По крайней мере, являясь проприетарным языком, MDX реализован во всех системах, поддерживающих OLAP.

Близкие аналитикам интерфейсы, позволяющие на полуинтуитивном уровне исследовать многомерные кубы, поддерживаются во всех OLAP-системах. Однако какой-либо стандартизации в этом направлении не видно.

Хранилища данных. Участники встречи считали, что для эффективного построения хранилищ данных требуется создать дополнительные инструменты: *средства для создания насосов данных (data pump), т.е. модулей, функционирующих над средой источников данных и поставляющих в хранилище те изменения, которые существенны с точки зрения хранилища; методы «чистки данных» (data scrubbing), которые обеспечивают согласование данных, удаление элементов, соответствующих разным представлениям одного и того же а также удаление неправдоподобных значений; средства для создания и поддержки метасловаря, информирующего пользователей о способах получения данных.*

Поскольку физически интегрирующие данные из разных источников и сохраняющие «исторические» данные хранилища данных были и остаются незаменимым поставщиком данных для OLAP-аналитиков, их качественному построению в последние 20 лет уделяли серьезное внимание как вендоры SQL-ориентированных СУБД, так и независимые производители. В целом все средства, обеспечивающие извлечение данных из внешних источников, их преобразование к схеме хранилища данных и очистку, загрузку в хранилище данных, принято называть средствами ETL (Extract, Transform, Load). Относительно современное представление об ETL описано в Части IV [42] («научные» публикации на эту тему отсутствуют).

Конечно, в число средств ETL входит средство очистки данных. Однако, насколько я понимаю, очистка в основном предполагает приведение данных к единому формату и удаление избыточных данных. Менее формальная очистка требует выполнения семантического анализа загружаемых данных и оказывается слишком трудозатратной (речь идет об очень больших объемах данных). Про «насосы данных» в том смысле, в котором они упоминались в отчете, слышать не приходилось.

Отслеживание происхождения данных (data lineage) и сейчас считается важным средством. Исследования продолжаются, но практических результатов не видно.

Репозитарии данных. Вот, что имелось в виду в отчете под термином *репозиторий*. *Приложения, относящиеся к категории репозитариев, характеризуются тем, что они предназначаются для хранения и управления как данными, так и метаданными, т.е. информацией о структуре данных. Примеры репозитариев – базы данных для поддержки компьютерного проектирования, включая CASE (системы проектирования программного обеспечения), а также системы управления документами. Отличительная черта этих систем – частые изменения метаданных, характерные для любой среды проектирования.*

В 1990-е гг. технология репозиториев считалась перспективной (см., например, [43]). Однако впоследствии, как мне кажется, сообществу баз

данных не удалось договориться с сообществом автоматизации проектирования, для которого, собственно, технология и предназначалась. Сегодня о репозиториях в этом смысле и не слышно.

5. Встреча в Кембридже: 1996

Четвертая встреча состоялась через год после третьей в Массачусетском технологическом институте 14-15 июня 1996 г. в рамках организованного ACM симпозиума Strategic Directions in Computing Research. Детали содержатся в приводимом фрагменте табл. 1.

Время и место встречи	Список участников на английском языке	Список участников на русском языке	Официальная публикация	Публикация на русском языке
14-15 июня 1996 г. MIT, Кембридж, шт. Массачусетс ACM Workshop on Strategic Directions in Computing Research «Cambridge meeting»	1. Jose Blakeley 2. Peter Buneman 3. Umesh Dayal 4. Tomasz Imielinski 5. Sushil Jajodia 6. Hank Korth 7. Guy Lohman, 8. Dave Lomet 9. Dave Maier 10. Frank Manola 11. Tamer Ozsu 12. Raghu Ramakrishnan 13. Krithi Ramamritham 14. Hans Schek 15. Avi Silberschatz 16. Rick Snodgrass 17. Jeff Ullman 18. Jennifer Widom, 19. Stan Zdonik	1. Хосе.Блейкли 2. Питер .Бьюнман 3. Умеш.Дайал 4. Томаш.Имилинский 5. Сущил.Джаджодиа 6. Хэнк.Корт 7. Гай.Лохман 8. Дейв.Ломе 9. Дейв Майер 10. Френк.Манола 11. Тамер.Оззу 12. Раджу Рамакришнан 13. Крити.Рамамритан 14. Ганс.Шек 15. Эви.Зильбершац 16. Рик Снодграсс 17. Джефф Ульман 18. Дженифер.Вайдом 19. Стенли.Здоник	Avi Silberschatz, Stan Zdonik, et al. Strategic Directions in Database Systems: Breaking Out of the Box. ACM Computing Surveys, 28(4):764-778, 1996	А.Зильбершац, С.Здоник и др. Стратегические направления в системах баз данных Перевод: М.Р. Когаловский Новая редакция: Сергей Кузнецов, 2009 г., http://citforum.ru/database/classics/nsf_report2/

Официально отчет о встрече опубликован в 1996 г. [5], ненамного позже отчета о предыдущей встрече. Предварительной неформальной публикации отчета не было, но в свободном доступе имеется текст официальной статьи: https://www2.cs.arizona.edu/~rts/pubs/CompSurvDec96.pdf, дата обращения 27.03.2017. Статья переведена на русский язык. Вот наиболее интересные прогнозы.

5.1 Расширяемость и компонентизация

Среди стратегически важных исследовательских проблем в отчете упоминаются следующие. *Нужно создать системы, которые дают возможность разработчику легко вводить новые типы данных, разработанные вне данной СУБД, которыми можно манипулировать внутри базы данных наравне с ее собственными полноправными типами. Нужно найти способы сделать архитектуру СУБД открытой таким образом, чтобы могли подключаться новые функциональные компоненты, и чтобы функциональные возможности системы базы данных могли конфигурироваться более гибкими способами в соответствии с потребностями приложений.*

Напомним, что встреча состоялась в середине 1996 г. В начале года компания Informix приобрела компанию Illustra вместе с ее технологий построения объектно-реляционных СУБД [30]. Насколько я понимаю, ко времени встречи в Кембридже в Informix кипела работа по интеграции тогдашнего основного продукта компании Informix Dynamic Server и СУБД Illustra, чтобы к концу года можно было построить обещанный руководством компании Informix Universal Server. Стандарта SQL, включающего объектно-реляционные возможности, еще не было, и разработчики опирались на еще очень недоработанные драфты SQL3 [25]. В том же 1996 г. компания Sun Microsystems реализовала язык и виртуальную машину Java [44], компонентную модель разработки программного обеспечения Java Beans [45] и интерфейс Java с SQL-ориентированными базами данных JDBC [46].

Думаю, что в этом контексте нужно рассматривать первую часть прогноза. Конечно, нет и не было возможности экспортировать в базу данных разработанные вне ее пользовательские типы данных, чтобы можно было использовать их наравне с типами данных, определенными средствами SQL (например, определять столбцы таблиц). Но серверные программы (хранимые процедуры, функции и методы новых типов данных) можно программировать на разных языках, используя разные библиотеки, в том числе, и технологию Java Beans.

Что касается компонентной организации самих СУБД, то эта идея привлекала многих исследователей и до, и после встречи в Кембридже. На мой вкус, наиболее убедительной публикацией на эту тему является [47]. Идея внешне выглядит очень понятной и привлекательной. Универсальные СУБД рассчитаны на общий случай использования. В любой конкретной среде и при любой конкретной рабочей нагрузке многие составляющие СУБД просто не работают, а их наличие только замедляет обработку пользовательских запросов. Было бы логически правильно учитывать (желательно автоматически) особенности текущей среды использования и конфигурировать СУБД таким образом, чтобы в ней не оставались ненужные в данное время компоненты.

Наверное, технически можно построить такую компонентную, автоматически конфигурируемую СУБД. Но сложность состоит в том, что чем более структурно строится любая сложная система (ОС, СУБД и т.д.), тем больше внутри нее требуется взаимодействия компонентов, и эти взаимодействия сами снижают производительность системы. Многие программные системы изначально проектировались высоко структурным образом как набор относительно автономных взаимодействующий компонентов, а потом в целях повышения эффективности приводились к практически монолитной архитектуре.

Интересно, что примерно те же соображения о недостатках универсальных СУБД привели в начале XXI века к идее о потребности перехода к специализированным СУБД [48].

5.2 Оптимизация запросов

По этому поводу в отчете говорится, что *могут измениться критерии оптимизации. Например, пользователи могут предпочесть получать ответы на свои запросы с меньшей скоростью, точностью и полнотой, если это будет стоить заметно дешевле*.

Оптимизация запросов – эта одна из наиболее сложных функций СУБД (и сейчас, и 20 лет тому назад). Здесь не место вдаваться в технические подробности (их очень много, и они нетривиальны). На мой взгляд, сегодня общее представление о сути оптимизации запросов лучше получать не из академических публикаций, а из документации вендоров (например, [49]).

Грубо говоря, работа оптимизатора запросов состоит из трех фаз: преобразование запроса, генерация планов выполнения запроса, оценка планов и выбор наиболее дешевого плана. На вход оптимизатора поступает внутреннее представление запроса, полученное синтаксическим анализатором. На первой фазе это представление преобразуется в семантически эквивалентное (непроцедурное) представление, обладающее предположительно улучшенным качеством (например, запросы с вложенными подзапросами приводятся к запросам с соединениями). На второй фазе, руководствуясь набором заложенных в СУБД стратегий и методов выполнения отдельных операций низкого уровня, оптимизатор строит набор процедурных планов выполнения запроса. Для сложных запросов этот набор потенциально может быть очень большим, и задачей оптимизатора является сохранение его осмысленного размера, используя эвристики, отбрасывающие предположительно худшие планы. Наконец, на третьей фазе производится оценка каждого оставленного в наборе плана, и выбирается план с наименьшей стоимостью.

В работе оптимизатора используются два фундаментальных предположения: запрос нужно выполнить полностью и точно; база данных сохраняется в традиционной дисковой памяти на устройствах с подвижными головками. Отказ от этих предположений может потребовать внесение принципиальных изменений на каждой фазе работы оптимизатора. Поэтому изменять критерии оптимизации запросов – дело накладное и сложное. Производители СУБД идут на это с большой неохотой. Прогноз можно считать несбывшимся.

5.3 Интеллектуальный анализ данных внутри СУБД

Участники встречи считали, что *классификация и кластеризация могут рассматриваться как случайные запросы, для которых необходимы новые семейства языков запросов. К числу исследовательских задач в этой области относится разработка адекватного набора простых примитивов запросов и нового поколения методов оптимизации запросов*.

В 1990-е гг. тема нагрузки SQL-ориентированных СУБД функциями интеллектуального анализа данных была сравнительно популярна в исследовательском сообществе баз данных [50]. Тогда это мотивировалось тем, что технология баз данных уже переросла традиционные потребности

бизнес-приложений, она способна поддерживать новые классы приложений, в том числе и аналитических. Один из вопросов, который занимал тогдашних исследований, состоял в том, не повлияет ли выполнение интеллектуального анализа данных внутри СУБД на эффективность систем при обработке запросов?

Сегодня ведущие поставщики SQL-ориентированных СУБД (например, компании Oracle и Microsoft) поставляют готовые решения data mining внутри баз данных, а также предоставляют средства разработки дополнительных серверных аналитических механизмов. Сегодняшняя мотивировка серверного интеллектуального анализа данных – это потребность в переносе обработки данных как можно ближе к месту их хранения. Серверная аналитика позволяет значительно сократить объем данных, передаваемых от серверов баз данных на рабочие станции, снизить уровень требований к аппаратному оснащению рабочих станций.

Вместе с тем, современные средства и механизмы интеллектуального анализа данных все более ориентируются на обработку неструктурированных данных (текст, изображения, видео-, аудиоданные и т.д.). Преимущества СУБД же в основном связаны с управлением структурированными данными. Трудно сказать, насколько востребованы сегодня средства интеллектуального анализа данных, встроенные в SQL-ориентированные СУБД. К сожалению, часто наблюдается обратная картина – в приложениях data mining вообще не используются СУБД.

А прогноз о развитии языков запросов и их оптимизаторов для поддержки интеллектуального анализа данных в целом не оправдался.

6. Асиломарская встреча: 1998

Пятая встреча состоялась 19-21 августа 1998 г. в парке Асиломар неподалеку от г. Монтерей, шт. Калифорния. Во фрагменте табл. 1 описаны детали встречи.

Время и место встречи	Список участников на английском языке	Список участников на русском языке	Официальная публикация	Публикация на русском языке
19-21 августа 1998 г. Асиломар, г. Пасифик Гров, шт. Калифорния «Asilomar meetong»	1. Phil Bernstein 2. Michael Brodie 3. Stefano Ceri 4. David DeWitt 5. Mike Franklin 6. Hector Garcia-Molina 7. Jim Gray 8. Jerry Held 9. Joe Hellerstein 10. H. V. Jagadish 11. Michael Lesk 12. Dave Maier 13. Jeff Naughton 14. Hamid Pirahesh 15. Mike Stonebraker 16. Jeff Ullman	1. Филипп Бернштейн 2. Майкл Броуди 3. Стефано Чери 4. Дэвид Девитт 5. Майк Франклин 6. Гектор Гарсиа-Молина 7. Джим Грей 8. Джерри Хелд 9. Джо Хеллерштейн 10. Х.В. Ягадиш 11. Майкл Леск 12. Дейв Майер 13. Джефф Нотон 14. Хамид Пирамеш 15. Майк Стонбрейкер 16. Джефф Ульман	Phil Bernstein, Michael Brodie et al. The Asilomar Report on Database Research. ACM SIGMOD Record, 27(4):74-80, 1998	Филипп Бернштейн, Майкл Броуди и др. Асиломарский отчет об исследованиях в области баз данных. Перевод: Сергей Кузнецов, 1999, http://citforum.ru/database/digest/asil_01.shtml

Официальный отчет о встрече был опубликован в том же 1998 г. [6]. Предварительной неофициальной публикацией можно считать https://arxiv.org/html/cs/9811013, дата обращения 30.03.2017. Среди прогнозов Асиломарского отчета наиболее интересны следующие.

6.1 Системы управления базами данных в стиле «plug and play»

Участники Асиломарской встречи отмечали, что *в связи с ростом относительной стоимости человеческого фактора в компьютерных системах требуется, чтобы будущие компьютерные системы стали полностью автоматическими: автоинсталлируемыми, автоуправляемыми, авторемонтируемыми и автопрограммируемыми. В том числе требуется обеспечение самонастраиваемости систем баз данных, т.е. удаление массы параметров настройки производительности, которые должны определять пользователи в текущих продуктах. В частности, должны появиться методы автоматического выбора индексов*.

Важность проблемы самонастраиваемости систем баз данных в XXI веке осознается всеми крупными вендорами SQL-ориентированных СУБД. По всей видимости, первой компанией, в которой начались масштабные исследования в этом направлении, была Microsoft (проект AutoAdmin под руководством Сураджита Чаудхари начался в 1996 г [51, 52]).

В начале этого проекта было разработано средство Index Tuning Wizard, которое анализировало текущую рабочую нагрузку СУБД (операции выборки данных, вставки, модификации и удаления строк таблиц) и устанавливала, при каком наборе индексов на соответствующих таблицах соответствующий набор операций обрабатывался бы СУБД в целом наиболее эффективно. При анализе рабочей нагрузки уже тогда использовались методы data mining. В дальнейшем в этом средстве стали учитываться не только индексы, но и материализованные представления и другие объекты базы данных, наличие или отсутствие которых влияет на эффективность обработки операций.

Имеющиеся сегодня в передовых СУБД средства самонастройки в основном направлены на то, чтобы помочь администраторам баз данных настроить физическую схему базы данных, дабы она наилучшим образом соответствовала особенностям текущей среды использования. Кроме того, в системы широко внедряются средства мониторинга, результаты которых позволяют администраторам распознать как собственные недочеты при настройке базы данных, так и слабые места СУБД (например, оптимизатора запросов). Кроме работ, ведущихся в Microsoft, в этой области заметны усилия исследователей и разработчиков компании Oracle [53].

Конечно, имеющиеся и ожидаемые средства этого рода помогают администраторам баз данных и настройщикам приложений. Однако совсем не факт, что их наличие позволяет существенно сократить число «ручек управления», предоставляемых администраторам со стороны СУБД.

Автоматическую настройку баз данных без участия человека-эксперта по-прежнему не поддерживает ни одна система, а от администраторов требуется еще более высокая квалификация. Так что прогноз нельзя считать сбывшимся, хотя он остается путеводной звездой для производственных исследователей.

6.2 Переосмысление традиционной архитектуры систем баз данных

В отчете утверждалась *потребность в переосмыслении традиционной архитектуры систем баз данных в свете появления среды, которая будет доступна в 2010-ом году.*

Эти утверждения созвучны названию статьи [34]: «Конец архитектурной эпохи, или Наступило время полностью переписывать системы управления данными». Да и вышла эта статья в 2007 г., аккурат к концу первого десятилетия XXI века. Однако при всем уважении к авторам [34] нельзя сказать, что они предлагают полностью переосмысленные архитектуры СУБД. Они переосмысливают их только в связи с переходом от архитектур универсальных СУБД к архитектурам специализированных систем. Компоненты архитектур остаются известными и традиционными.

Да в общем-то и оснований для полного пересмотра архитектуры в 2000-е гг. не было. Имелись в основном количественные, а не качественные изменения (существенно увеличенные размеры основной памяти, кластеры с очень большим числом узлов и т.д.). Неизменной оставалась опора SQL-ориентированных СУБД на жесткие диски с подвижными головками. (Мне близка следующая цитата из Асиломарского отчета: «в будущем могут оказаться неуместными архитектуры, “скрывающие наличие подвижных магнитных головок”, такие как RAID 5» – с позиций оптимизации запросов мне всегда были подозрительны устройства внешней памяти, скрывающие свою организацию.)

Зато, по моему мнению, это время наступило сейчас (и, видимо, продлится в следующем десятилетии, потому что новые архитектуры быстро не рождаются). Вот что может действительно повлиять на архитектуру будущих СУБД (новые аппаратные возможности перечисляются в порядке возрастания их влияния на архитектуру СУБД).

Флэш-память и твердотельные диски. Твердотельная внешняя память (Solid-State Disk, SSD) обладает принципиально иными характеристиками, чем традиционная дисковая память на дисковых устройствах с подвижными головками (Hard Disk, HD). Основным отличием является то, что у устройств SSD отсутствуют механически перемещаемые магнитные головки, и они действительно являются устройствами прямого доступа – время доступа к любому блоку данных в SSD одно и то же. И даже без учета времени перемещения головок у HD SSD показывают время обмена, в несколько раз меньшее, чем у HD.

Раньше SSD проигрывали HD по максимально доступному объему, стоимости в расчете на гигабайт данных и износоустойчивости. В настоящее время уже доступны SSD емкостью в несколько терабайт. Показатели стоимости и износоустойчивости пока уступают соответствующим показателям HD, но со временем и они будут улучшены.

Естественно, возникает желание перевести имеющиеся СУБД на платформу SSD. Но просто это не делается. Особенности SSD заставляют пересмотреть алгоритмы, используемые во многих компонентах SQL-ориентированных СУБД, а может быть и архитектуру СУБД в целом. Например, при переходе от SSD к HD перестают эффективно работать методы кэширования блоков базы данных в основной памяти [54], требуется их принципиальная переделка.

Безусловно, переход к SSD требует значительной переделки оптимизатора запросов, начиная, по-видимому, с эвристик «отсеивания» заведомо негодных планов запросов. Более того, в наборе планов для данного запроса могут появиться планы, которые вообще не генерировались при использовании HD. Полностью меняется компонент оценки стоимости плана, поскольку теперь обмены с внешней памятью будут стоить гораздо дешевле, и эта стоимость не будет зависеть от расположения блока данных во внешней памяти.

У меня имеется ощущение, что основные вендоры SQL-ориентированных СУБД опасаются этих изменений, затрагивающих самые внутренние части СУБД, и поэтому используют SSD, в основном пренебрегая их отличиями от HD.

Энергонезависимая основная память. Этот вид памяти, называемой non-volatile memory (долговременной, или энергонезависимой памятью), а также storage class memory (память класса хранения данных), наконец-то становится доступным. Еще 30 лет тому назад использование энергонезависимой памяти предполагалось в проекте системы хранения Postgres [55]. Стоунбрейкер хотел использовать такую память (в небольшом объеме) для хранения части кэша блоков базы данных и журнала. Конечно, тогда энергонезависимую память взять было негде, как, собственно, и в следующие десятилетия вплоть до нашего времени. Сейчас память класса хранения данных стала реальностью. Имеется несколько вариантов физического исполнения такой памяти, но все они обеспечивают прямую адресацию и сохранность данных после отключения электропитания.

Конечно, очень заманчиво использовать энергонезависимую память в СУБД. Сегодня достаточно популярны СУБД, хранящие данные в традиционной основной памяти (in-memory DBMS). Но эти системы не могут обойтись без дисковой памяти, потому что для всех транзакций, изменяющих данные, должна обеспечиваться гарантированная сохранность результатов (durability), т.е. эти результаты так или иначе должны быть отображены во внешнюю память. По этой причине системы in-memory обычно обеспечивают существенно большую производительность, чем традиционные СУБД, для только читающих транзакций. (Исключением является массивно-параллельная

архитектура H-Store/VoltDB [34, 36], в которой durability транзакций поддерживается за счет репликации данных.)

Если вся система хранения данных в СУБД поддерживается в энергонезависимой памяти, то внешняя память перестает быть необходимой. Но для построения таких СУБД нужны кардинально другие подходы к представлению данных, организации индексов, управлению транзакциями, журнализации, оптимизации запросов и т.д. Что-то можно позаимствовать из арсенала СУБД, хранящих данные в традиционной памяти, но очень многое, включая общую архитектуру системы нужно изобретать заново. В последние годы соответствующие исследования начались [56-57], но как показывают публикации, они находятся на начальной стадии, и предстоит еще много работы, пока удастся получить OLTP-ориентированную СУБД, работающую со скоростью основной памяти.

Массивно-мультипоточные архитектуры компьютеров. Сегодняшние компьютеры рассчитаны на то, что выполняемые в них программы соблюдают принцип локальности. Это означает, что в любой момент выполнения любого процесса ему достаточно обеспечить некоторый ограниченный набор команд и данных (рабочий набор), причем если процесс обладает рабочим набором *WS* в момент времени *T*, то с большой вероятностью этот рабочий набор сохранится и в момент времени *T+t* для некоторого значения *t*. Принцип локальности позволяет поддерживать в процессорах аппаратный кэш, и именно наличие этого кэша дает возможность сгладить разрыв в скорости между процессором и основной памятью.

Однако существуют классы приложений, в которых программам не свойственен принцип локальности (приложения биоинформатики, анализа социальных сетей и т.д.). При работе таких приложений наличие кэша никак не способствует эффективности, и они работают, по сути, со скоростью основной памяти. Идея аппаратной архитектуры для поддержки таких приложений появилась в конце прошлого десятилетия в компании Cray Inc. [58].

Суть идеи состоит в том, что нужно сделать массивно-многопоточный процессор, поддерживающий на аппаратном уровне десятки тысяч потоков с общим доступом к основной памяти. Кэш в процессоре не поддерживается, но для каждого потока обеспечивается собственный набор регистров. Если в данный момент времени выполняется поток *Tr1*, и в нем потребовался обмен с основной памятью, то он аппаратно блокируется, и запускается один из потоков *Tri*, у которого все требуемые данные находятся в регистрах. При достаточно большом числе потоков готовый к выполнению поток всегда найдется, и многопоточная программа будет в целом выполняться со скоростью процессора.

Для достижения таких результатов нужно научиться распараллеливать программу на очень большое число небольших (несколько десятков команд) потоков. Это очень сложная задача, отсутствие решение которой тормозит развитие массивно-мультипоточных архитектур. Но, по моему мнению,

потенциальная возможность построения такой архитектуры должна стимулировать исследования в сообществе баз данных. Предположим, что массивно-мультипоточный компьютер оснащен энергонезависимой основной памятью, и эта память используется для хранения баз данных. СУБД работает на некотором внешнем компьютере с традиционной архитектурой и при компиляции операторов SQL распараллеливает их на большое число потоков. Это гораздо более реально, чем для произвольной программы, потому что SQL – декларативный язык.

Конечно, это очень непростая задача, но если ее решить, мы сможем получить системы баз данных, обрабатывающие запросы запросы со скоростью кэша.

7. Лоуэллская встреча: 2003

Следующая, шестая встреча исследователей в области баз данных состоялась 4-6 мая 2003 г. в Лоуэлле, шт. Массачусетс. Фрагмент табл. 1 описывает детали встречи.

Время и место встречи	Список участников на английском языке	Список участников на русском языке	Официальная публикация	Публикация на русском языке
4-6 мая 2003 г. Лоуэлл, шт. Массачусетс «Lowell meeting»	1. Serge Abiteboul 2. Rakesh Agrawal 3. Phil Bernstein 4. Mike Carey 5. Stefano Ceri 6. Bruce Croft 7. David DeWitt 8. Mike Franklin 9. Hector Garcia Molina 10. Dieter Gawlick 11. Jim Gray 12. Laura Haas 13. Alon Halevy 14. Joe Hellerstein 15. Yannis Ioannidis 16. Martin Kersten 17. Michael Pazzani 18. Mike Lesk 19. David Maier 20. Jeff Naughton 21. Hans Schek 22. Timos Sellis 23. Avi Silberschatz 24. Mike Stonebraker 25. Rick Snodgrass 26. Jeff Ullman 27. Gerhard Weikum 28. Jennifer Widom 29. Stan Zdonik	1. Серж Абитебуль 2. Ракеш Агравал 3. Филипп Бернштейн 4. Майк Кэри 5. Стефано Чери 6. Брюс Крофт 7. Дэвид Девитт 8. Майк Франклин 9. Гектор Гарсиа-Молина 10. Дитер Гавлик 11. Джим Грей 12. Лаура Хаас 13. Элон Хэлеви 14. Джо Хеллерштейн 15. Янис Ионнидис 16. Мартин Керстен 17. Майкл Паззани 18. Майк Леск 19. Дэвид Мейер 20. Джефф Нотон 21. Ганс Шек 22. Тимос Селлис 23. Эви Зильбершац 24. Майкл Стоунбрейкер 25. Рик Снодграсс 26. Джефф Ульман 27. Герхард Вейкум 28. Дженифер Вайдом 29. Стен Здоник	Serge Abiteboul, Rakesh Agrawal et al. The Lowell Database Research Self-Assessment. Communications of the ACM, 48(5):111-118, 2005	Сергей Кузнецов. Крупные проблемы и текущие задачи исследований в области баз данных, 2005, http://citforum.ru/database/articles/problems/

Официальная публикация Лоуэллского отчета [7] появилась в 2005 г., предварительная неофициальная публикация доступна на http://jimgray.azurewebsites.net/lowell/lowelldatabaseresearchselfassessment.pdf, дата обращения 1 апреля 2017 г. В [8] основные положения отчета пересказываются и комментируются на русском языке. Вот наиболее интересные прогнозы.

7.1 Интеграция текста, данных, кода и потоков

В отчете утверждается, что *пора прекратить встраивать новые конструкции в старую реляционную архитектуру. Нужно переосмыслить базовую архитектуру СУБД с целью поддержки структурированных данных; текстовых, пространственных, темпоральных и мультимедийных данных; процедурных данных, т.е. типов данных и инкапсулирующих их методов; триггеров; потоков и очередей данных как равноправных компонентов первого сорта внутри архитектуры СУБД (как на уровне интерфейсов, так и на уровне реализации). Для исследовательского сообщества требуется выработка новой системы понятий.*

На мой взгляд, никакой «новой системы понятий» не появилось. Да и попыток переосмысления базовой архитектуры универсальных СУБД за прошедшие годы видно не было. С одной стороны, основные вендоры коммерческих СУБД продолжали наращивать функциональные возможности своих продуктов, стараясь как можно меньше изменять ядро систем (т.е., по сути, делая новые компоненты объектами «второго сорта»). С другой стороны, с 2005 г. на мир баз данных воздействует слоган Майкла Стоунбрейкера «Один размер непригоден для всех» [48], и с разным успехом было выполнено несколько проектов специализированных СУБД, в которых на первое место выходили именно те понятия, на поддержку которых ориентировалась система.

В качестве поучительного примера попытки расширить число объектов «первого сорта» в универсальных СУБД стоит вспомнить сравнительно недавнюю историю поддержки XML в базах данных. Спецификация XML (eXtendable Markup Language) появилась в начале 1998 г. Язык в основном предназначался для представления сообщений, передаваемых в Internet. Представлялось, что таких сообщений будет очень много, полезно уметь их сохранять и производить поиск в получаемых коллекциях XML-документов. Это было толчком к разработке специализированных XML-ориентированных СУБД с собственными системами хранения и поддержкой языка запросов XQuery. Первыми эту работу начали компания Software AG (Tamino [59]) и ИСП РАН (Sedna [60]).

Tamino и Sedna были уже сравнительно работоспособными системами, когда в игру вступили IBM и Oracle. В SQL-ориентированной СУБД XML с собственным языком запросов поддерживать не так уж просто, и вендорам, фактически, пришлось реализовать отдельную среду хранения XML,

процессор и оптимизатор языка XQuery, а затем интегрировать все это новое хозяйство со средой SQL, обеспечив возможность встраивания XQuery в SQL, SQL – в XQuery и т.д. Потребовалась большая и дорогостоящая работа, чтобы сделать XML и XQuery «полноправными жителями» SQL-ориентированной СУБД. По моим понятиям, обе компании выполнили эту работу прекрасно. Но ко времени достижения готовности к использованию таких интегрированных систем интерес к XML затух. Новым фаворитом стал язык JSON, а результаты проделанной работы (как в области создания специализированных XML-ориентированных систем, так и в направлении разработки интегрированных СУБД) оказались во много невостребованными.

Но на создание специализированных систем было затрачено меньше сил и средств. Так что стоит подумать о целесообразности новой системы понятий для универсальных СУБД.

7.2 Объединение информации

Участники встречи отмечали, что *в Internet парадигма ETL не приемлема. Теперь требуется производить интеграцию информации между несколькими предприятиями. В результате потребуется интеграция, возможно, миллионов информационных источников «на лету».*

В такой общей постановке проблема не решена (скорее всего, она и вовсе неразрешима). Реалистический подход был предложен в 2005 г. в [61]. В этой статье утверждалось, что унифицированный подход к интеграции данных невозможен и не нужен. Должен обеспечиваться тот уровень интеграции, который требуется конкретному типу приложений. Например, для выполнения интеллектуального анализа данных достаточно физически собрать данные из разных источников без какого-либо их преобразования, а для классического OLAP требуется построение хранилища данных с полной поддержкой ETL. К сожалению, на практике красивая идея пространств данных пока востребована не была.

Если продолжать говорить про ETL и OLAP, то в последние годы сравнительно жизнеспособной является идея виртуальных хранилищ данных [62], в которых поддерживаются обширные метаданные и инкрементнально обновляемые агрегаты, а доступ к фактам производится «на лету» в соответствующих внешних источниках. В таких «хранилищах данных» не поддерживаются исторические данные, и возможности OLAP ограничены. За это и другие объективные недостатки виртуальные хранилища данных достаточно резко ругает классик технологии хранилищ данных Билл Инмонн [63].

8. Клермонтская встреча: 2008

Седьмая встреча состоялась 29-30 мая 2008 г. в гостинице Клермонт Ресорт в Беркли, шт. Калифорния. Подробности приведены во фрагменте табл. 1.

Время и место встречи	Список участников на английском языке	Список участников на русском языке	Официальная публикация	Публикация на русском языке
29-30 мая 2008 г. Беркли, шт. Калифорния, Клермонт Ресорт «Claremont meeting»	1. Rakesh Agrawal 2. Anastasia Ailamaki 3. Philip A. Bernstein 4. Eric A. Brewer 5. Michael J. Carey 6. Surajit Chaudhuri 7. AnHai Doan 8. Daniela Florescu 9. Michael J. Franklin 10. Hector Garcia Molina 11. Johannes Gehrke 12. Le Gruenwald 13. Laura M. Haas 14. Alon Y. Halevy 15. Joseph Hellerstein 16. Yannis E. Ioannidis 17. Hank F. Korth 18. Donald Kossmann 19. Samuel Madden 20. Roger Magoulas 21. Beng Chin Ooi 22. Tim O'Reilly 23. Raghu Ramakrishnan 24. Sunita Sarawagi 25. Michael Stonebraker 26. Alexander S. Szalay 27. Gerhard Weikum	1. Ракеш Агравал 2. Анастасия Айламаки 3. Филипп Бернштейн 4. Эрик Брюер 5. Майкл Кэри 6. Сураджит Чаудхари 7. Анхай Доан 8. Даниела Флореску 9. Майкл Франклин 10. Гектор Гарсиа-Молина 11. Иоханнес Герке 12. Ле Грюнвальд 13. Лаура Хаас 14. Элон Хэлеви 15. Джозеф Хеллерштейн 16. Янис Ионнидис 17. Хэнк Корт 18. Дональд Коссманн 19. Сэмуэль Мэдден 20. Роджер Магулас 21. Бен Чин Ой 22. Тим О'Рейли 23. Раджу Рамакришнан 24. Сунита Сарагави 25. Майкл Стоунбрейкер 26. Александр Шалай 27. Герхард Вейкум	Rakesh Agrawal, Anastasia Ailamaki,et al. The Claremont Report on Database Research. Comm. of the ACM, 52(6):56-65, 2009.	Ракеш Агравал, Анастасия Айламаки др. Клермонтский отчет об исследованиях в области баз данных. Пересказ и комментарии: Сергей Кузнецов, 2008, http://citforum.ru/database/articles/claremont_report/

Официальный отчет о встрече опубликован в [9] в 2009 г., предварительная версия доступна на специальном сайте http://db.cs.berkeley.edu/claremont/, дата обращения 2 апреля 2017 г. (на этом сайте также размещены слайды участников встречи – нечто вроде их представления друг другу, очень интересно!). На русском языке имеется мой пересказ отчета с комментариями. На встрече отмечался ряд факторов, влияющих на направления исследований в области баз данных.

8.1 Повышение ажиотажа вокруг Больших Данных

По мнению участников встречи, наличие проблемы Больших Данных *приводит к быстрому росту числа пользователей традиционных систем управления базами данных (СУБД), а также стимулирует разработку новых специализированных решений управления данными на основе упрощенных компонентов. Повсеместное использование больших данных приводит и к возрастанию числа разработчиков технологий управления данными, что, несомненно, вызовет коренную реорганизацию этой области.*

С этим мнением участников Клермонтской встречи нельзя не согласиться, хотя уже в 2008 г. оно выглядело не прогнозом, а скорее наблюдением за происходящим. К этому времени уже существовали специализированные массивно-параллельные аналитические СУБД Vertica [35], Greemplum и Aster Data [12], специализированная потоковая система StreamBase [48], был готов прототип специализированной массивно-параллельной транзакционной СУБД H-Store [34] и велась подготовка ее коммерциализации (VoltDB [36]) и т.д. Разработка новых специализированных решений ведется и в настоящее время (см. например, [64]).

Интересно, что почти все перечисленные компании были поглощены более крупными и заслуженными компаниями, в основном не специализирующимися на производстве продуктов управления базами данных: Vertica была поглощена в 2011 г. компанией Hewlett-Packard: Greemplum – в 2010 г. компанией EMC; Aster Data – в 2011 г. компанией Teradata; StreamBase – в 2013 г. компанией TIBCO. Некоторые из этих продуктов видны за пределами их новых владельцев, некоторые – нет, т.е. несмотря на использование передового подхода Майкла Стоунбрейкера мы имеем всего лишь ряд удачных стартапов, а реальная технология управления базами данных по-прежнему в руках традиционных гигантов.

Что касается второй части наблюдения авторов отчета, то уже в 2008 г. направление NoSQL развивалось небывалыми в истории баз данных темпами. Быстро возрастало число реализованных (почти всегда с открытыми исходными кодами) систем, в которых отрицались SQL и ACID-транзакции. В 2006 г. начался проект Hadoop для открытой реализации MapReduce. Сегодняшний список нереляционных, распределенных, горизонтально масштабируемых СУБД с открытыми текстами насчитывает более 230 систем [66], Hadoop широко используется, развивается и совершенствуется.

Непонятно, означает ли это коренную реорганизацию области баз данных? Трудно сказать. Сегодня сообщество NoSQL сторонится как исследовательской работы, так и традиционного исследовательского сообщества баз данных. Уже несколько лет проводятся специализированные конференции, посвященные исключительно проблематике NoSQL, но если посмотреть на сайты этих конференций, видно, что они совсем не такие, как VLDB, Data Engineering, SIGMOD и т.д. – конференции, на которых исследователи рассказывают о полученных результатах. Они больше похожи на конференции, которые вендоры программных систем проводят для своих пользователей.

Не знаю, может ли технология развиваться без исследований. Не зайдет так в тупик направление NoSQL?

8.2 Изменения в архитектуре компьютерных систем

В отчете отмечаются следующие архитектурные изменения. *На макроуровне фундаментальным изменения в архитектуре программного обеспечения сулит развитие «облачных» компьютерных служб. На микроуровне в компьютерных архитектурах закон Мура теперь трактуется в пользу не*

повышения тактовой частоты микропроцессоров, а увеличения числа процессорных ядер и потоков управления в одном кристалле. Основные изменения в технологии хранения данных относятся к иерархии памяти в связи с доступностью большего числа кэшей увеличенного объема на одном кристалле, все более дешевой основной памяти большого объема и флэш-памяти.

С утверждением насчет *макроуровня* в целом нельзя не согласиться. Модель «Программное обеспечение как услуга» (Software As A Service, SAAS) в большом числе случаев позволяет пользователям избавиться от потребности в установке, администрировании и поддержке программного обеспечения. Конечно, это относится и к облачным СУБД. Но с ними связана проблема, на которую, как мне кажется, в сообществе облачных вычислений (да и в сообществе баз данных) закрывают глаза.

Основой облачных вычислений является виртуализация. Пользователь запрашивает у поставщика облачных услуг услугу с требуемыми характеристиками (на основе Соглашение об уровне предоставления услуги – Service Level Agreement, SLA), и его не касается, каким образом, за счет каких физических ресурсов эта слуга будет оказываться. Виртуализация на всех уровнях, от уровня IAAS (инфраструктура как услуга) до уровня SAAS предполагает, в частности, что облачная СУБД работает на виртуальном сервере, оснащенном виртуальной системой хранения.

В то же время, оптимизатор запросов этой облачной СУБД полагает, что СУБД по-прежнему работает с реальными магнитными дисками с подвижными головками и выбирает планы запросов, выполнение которых потребует меньшего числа обменов с дисковыми устройствами. Результаты могут оказаться непредсказуемыми и очень неприятными для пользователей. Я вижу только один реальный путь к устранению этой проблемы – при запросе услуги облачной СУБД обеспечивать ей реальную, а не виртуальную аппаратуру с характеристиками, определяемыми пользователями. Возможно, дело к этому и идет, но подтверждающие это публикации отсутствуют.

Что касается *микроуровня*, то распараллеливание запросов в симметричных мультипроцессорных системах (системах с общей основной памятью), к которым относятся и современные многоядерные и частично многопоточные компьютеры, поддерживается в развитых SQL-ориентированных СУБД не первый десяток лет. Реальной проблемой является то, что СУБД не может обеспечить горизонтальную масштабируемость при возрастании числа ядер (или нитей) в процессоре из-за ограничений параллельного доступа к основной памяти.

Некоторую надежду на возможность решения этой проблемы дает публикация [65]. В ней предлагается эскиз архитектуры СУБД с хранением данных в основной памяти, вроде бы обеспечивающей горизонтальную масштабируемость. Общая идея состоит в том, что в системе с *n* ядрами выполняется один процесс СУБД с *n* потоками, жестко привязанными к ядрам. В каждой нити выполняется вся СУБД целиком, как в массивно-параллельных

СУБД без общих ресурсов, а разделение данных выполняется за счет механизма виртуальной памяти. Потоки взаимодействуют путем обмена сообщениями на основе наличия общей памяти. При потребности заново разделить базу данных перепись данных не требуется, изменяется лишь таблица страниц виртуальной памяти. Компиляция и оптимизация запросов производятся во внешнем компьютере.

Наконец, свои соображения относительно средств долговременного хранения данных я уже изложил в 6.2. К этому можно добавить лишь то, что доступность больших объемов основной памяти активизировала разработки СУБД класса in-memory.

9. Бекманская встреча, 2013

Последняя к настоящему времени встреча прошла 14-15 октября 2013 г. в Бекманском центре университета в г. Ирвин, шт. Калифорния. Подробности в приводимом фрагменте табл. 1.

Время и место встречи	Список участников на английском языке	Список участников на русском языке	Официальная публикация	Публикация на русском языке
14-15 октября 2013 г. Ирвин, шт. Калифорния, Бекманский центр университета в Ирвине «Beckman meeting»	1. Daniel Abadi 2. Rakesh Agrawal 3. Anastasia Ailamaki 4. Magdalena Balazinska 5. Philip A. Bernstein 6. Michael J. Carey 7. Surajit Chaudhuri 8. Jeffrey Dean 9. AnHai Doan 10. Michael J. Franklin 11. Johannes Gehrke 12. Laura M. Haas 13. Alon Y. Halevy 14. Joseph Hellerstein 15. Yannis E. Ioannidis 16. H.V. Jagadish 17. Donald Kossmann 18. Samuel Madden 19. Sharad Mehrotra 20. Tova Milo 21. Jeffrey F. Naughton 22. Raghu Ramakrishnan 23. Volker Markl 24. Christopher Olston 25. Beng Chin Ooi 26. Christopher Re 27. Dan Suciu 28. Michael Stonebraker 29. Todd Walter 30. Jennifer Widom	1. Дэниел Абади 2. Ракеш Агравал 3. Анастасия Айламаки 4. Магдалена Балазинска 5. Филип А. Бершстейн 6. Майкл Дж. Кэри 7. Сурадждит Чаудхари 8. Джеффри Дин 9. Анхай Доан 10. Майкл Дж. Франклин 11. Йоханнес Герке 12. Лаура М. Хаас 13. Элон И. Хэлеви 14. Джозеф Хеллерштейн 15. Янис Е. Ионнидис 16. Х.В. Ягадиш 17. Дональд Коссманн 18. Самуэль Мэдден 19. Шарад Мехротра 20. Това Мило 21. Джеффри Нотон 22. Раджу Рамакришнан 23. Волкер Маркл 24. Кристофер Олстон 25. Бен Чин Ой 26. Кристофер Ре 27. Дан Сучиу 28. Майкл Стоунбрейкер 29. Тодд Валтер 30. Джерифер Вайдом	Daniel Abadi, Rakesh Agrawal et al. The Beckman Report on Database Research. SIGMOD Record, September, 43(3):61-70, 2014	Дэниел Абади, Ракеш Агравал и др. Бекманский отчет об исследований в области баз данных. Перевод: Сергей Кузнецов, 2017

Официальная публикация появилась в 2014 г., неофициальная доступна на сайте встречи https://beckman.cs.wisc.edu/, дата обращения 2 апреля 2017 г. На этом сайте имеются слайды с представлениями участников и их групповой фотографией (рис. 1). Отчет переведен на русский язык.

Рис. 1. Групповая фотография участников Бекманской встречи

Pic. 1. The group photo of the Beckman meeting participants

(from https://beckman.cs.wisc.edu/)

В центре внимания участников Бекманской встречи находилась проблема Больших Данных. В отчете утверждается следующее. *Являясь сообществом, которое в течение 45 лет раздвигало границы обработки больших наборов данных, сообщество баз данных может помочь двигаться вперед миру, управляемому данными, основываясь на собственных результатах и опыте. Тем самым, у нашего сообщества имеются уникальные возможности для решения проблемы Больших Данных, огромный потенциал для революционного воздействия.*

Для реализации этого потенциала требуется обратить особое внимание на пять областей исследований: *масштабируемые инфраструктуры больших/быстро поступающих данных; преодоление разнородностей ландшафта управления данными; сквозные обработка и интерпретация данных; облачные службы; управление различными ролями людей в жизненном цикле данных.*

Мне кажется, что сейчас еще рано комментировать прогнозы, содержащиеся в этом отчете, оценивать, действительно ли исследования, касающиеся проблемы Больших Данных, сосредотачиваются в указанных пяти областях. Нужно дождаться хотя бы следующей встречи. Но следует заметить, что создание методов и средств сквозной обработки данных (от необработанных данных до пригодных для практического использования знаний) кажется очень объемной задачей, выходящей за замки традиционных исследовательских работ.

В связи с этим очень важной является обсуждаемая в конце отчета проблема культуры исследований. *В последние годы тревожным явлением стало*

возрастающее внимание к числу публикаций и счетчикам цитирования, а к не результатам исследований. Это мешает выполнять крупные исследовательские проекты, проводить качественные конференции и т.д. Публикации и доклады на конференциях становятся не приносящим удовольствие авторам средством представить коллегам свои достижения и результаты, а всего лишь рутинной обузой.

Увы, с этим постоянно приходится сталкиваться и российским исследователям в разных научных областях. Уже выросло целое поколение исследователей, которые никогда не писали статьи и не выступали на конференциях просто потому, что у них назрела потребность сделать это. Участники Бекманской встречи не пришли к согласию о способах решения этой проблемы, но очевидно, что нужно стремиться к тому, чтобы написание и публикация статьи приносили авторам радость, а не были навязанной извне дополнительной неприятной нагрузкой.

9. Заключение

Отчеты о встречах специалистов исследовательского сообщества исследований баз данных очень полезны для всех людей, интересующихся технологий баз данных. Они дают понять, когда и чем руководствовались исследователи в своей работе, какие на них воздействовали внешние силы, всегда ли выбираемые направления исследований соответствовали потребностям развития общей технологии.

В этой статье я попытался обеспечить ретроспективный взгляд в многолетнюю историю баз данных после первой встречи исследователей в 1988 г. в Лагуна-Бич. Мои размышления, безусловно, субъективны, и с ними совсем не обязательно нужно соглашаться, но это размышления заинтересованного человека, который все эти годы ждал новых встреч, жадно читал (и с удовольствием переводил и/или комментировал) отчеты и соизмерял свою собственную работу с мнением и прогнозами участников этих встреч.

Статья написана вдогонку к моему выступлению на Семинаре Московской секции ACM SIGMOD [66] в конце 2015 – начале 2016 гг., хотя многие высказанные тогда мной соображения в статье были пересмотрены.

Список литературы

[1]. Philip A. Bernstein, Umeshwar Dayal, David J. DeWitt et al. Future Directions in DBMS Research. ACM SIGMOD Record, vol. 18, № 1, 1989, pp. 17-26

[2]. Сергей Кузнецов. Будущие направления исследований в области баз данных: десять лет спустя. http://citforum.ru/database/articles/future_01.shtml, дата обращения 2 апреля 2017 г.

[3]. Abraham Silberschatz, Michael Stonebraker, and Jeffrey D. Ullman. Database Systems: Achievements and Opportunities. Communications of the ACM, vol. 34, № 10, 1991, pp. 110-120,

[4]. Abraham Silberschatz, Michael Stonebraker, Jeffrey D. Ullman: Database Research: Achievements and Opportunities Into the 21st Century. SIGMOD Record, vol. 25, № 1, 1996, pp. 52-63. Перевод на русский язык: http://citforum.ru/database/classics/nfs_report, дата обращения 2 апреля 2017 г./

[5]. Avi Silberschatz, Stan Zdonik et al. Strategic Directions in Database Systems – Breaking Out of the Box. ACM Computing Surveys, vol. 28, № 4, Dec 1996, pp. 764-778. Перевод на русский язык: http://citforum.ru/database/classics/nsf_report2/, дата обращения 2 апреля 2017 г.

[6]. Philip A. Bernstein, Michael L. Brodie, Stefano Ceri et. al. The Asilomar Report on Database Research. SIGMOD Record, vol. 27, № 4, 1998, pp. 74-80. Перевод на русский язык: http://citforum.ru/database/digest/asil_01.shtml, дата обращения 2 апреля 2017 г.

[7]. Serge Abiteboul, Rakesh Agrawal, Philip A. Bernstein et al. The Lowell Database Research Self-Assessment. Communications of the ACM, vol 48, № 5, 2005, pp. 111-118

[8]. Сергей Кузнецов. Крупные проблемы и текущие задачи исследований в области баз данных. http://citforum.ru/database/articles/problems/, 2005, дата обращения 2 апреля 2017 г.

[9]. Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, et. al. The Claremont Report on Database Research. Communications of the ACM, vol. 52, № 6, 2009, pp. 56-65. Пересказ с комментариями: http://citforum.ru/database/articles/claremont_report/, дата обращения 2 апреля 2017 г.

[10]. Daniel Abadi, Rakesh Agrawal, Anastasia Ailamaki, et al. The Beckman Report on Database Research. ACM SIGMOD Record, vol. 43, № 3, September 2014, pp. 61-70. Перевод на русский язык: http://citforum.ru/database/articles/beckman_report/, дата обращения 4 мая 2017 г.

[11]. Keen, P. G. W. and M. S. Scott Morton. Decision support systems: an organizational perspective. Reading, Mass., Addison-Wesley Pub. Co., 1978

[12]. Сергей Кузнецов. MapReduce: внутри, снаружи или сбоку от параллельных СУБД?. Труды ИСП, т. 19, 2010, стр. 35-40

[13]. Edward A.Feigenbaum and Pamela McCorduck. The fifth generation: Japan's computer challenge to the world. Creative Computing Magazine, volume 10, Number 08, August 1984, pp. 103-111.

[14]. David DeWitt, Jim Gray. Parallel database systems: the future of high performance database systems. Communications of the ACM, vol. 35, Issue 6, June 1992, pp. 85-98

[15]. Marianne Winslett. Jim Gray speaks out. ACM SIGMOD Record, vol. 32, Issue 1, March 2003, pp. 53-61

[16]. Gartnet IT Glassary. Database Appliances. http://www.gartner.com/it-glossary/database-appliances, дата обращения 22.03.2017

[17]. Exadata. https://ru.wikipedia.org/wiki/Exadata, дата обращения 22.03.2017

[18]. Michael Stonebraker. My Top 10 Assertions About Data Warehouses. BLOG@CACM, August 26, 2010. Перевод на русский язык: http://citforum.ru/gazeta/166/, 2010 г., дата обращения 22.03.2017

[19]. С.О. Приказчиков, П.С. Костенецкий. Применение графических ускорителей для обработки запросов над сжатыми данными в параллельных системах баз данных. Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика, т. 4, вып. 1, 2015, стр. 64-70

[20]. M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhause. CHORUS Distributed Operating Systems. Computing Systems, vol. I, No. 4, Fall 1988, pp. pp. 305-370

[21]. Igor Burdonov, Victor Ivannikov, German Kopytov, Alexander Kosachev, Sergei Kuznetsov. The CLOS project: Towards an object-oriented environment for application development. Next Generation Information System Technology. Lecture Notes in Computer Science (LNCS), vol. 504, 1991, pp. 422-427, DOI: 10.1007/3-540-54141-1_23

[22]. David B. Golub , Daniel P. Julin , Richard F. Rashid , Richard P. Draves , Randall W. Dean , Alessandro Forin , Joseph Barrera , Hideyuki Tokuda , Gerald Malan , David Bohman. Microkernel operating system architecture and mach. In Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, 1992, pp. 11-30

[23]. QNX Operating Systems. http://www.qnx.com/content/qnx/en/products/neutrino-rtos/index.html, дата обращения 22.03.2017

[24]. Andrew Tanenbaum, Raja Appuswamy, Herbert Bos, Lorenzo Cavallaro, Cristiano Giuffrida, Tomáš Hrubý, Jorrit Herder, Erik van der Kouwe, David van Moolenbroek. MINIX 3: Status Report and Current Research. ;login, vol. 35, № 3, June 2010, pp. 7-13

[25]. С.Д. Кузнецов. Стандарты языка реляционных баз данных SQL: краткий обзор. СУБД, № 2, 1996, стр. 6-36. http://citforum.ru/database/articles/art_2.shtml, дата обращения 22.03.2017

[26]. The 1995 SQL Reunion: People, Projects, and Politics. Edited by Paul McJones, August 20, 1997 (2nd edition), http://www.mcjones.org/System_R/SQL_Reunion_95/SRC-1997-018.pdf, дата обращения 22.03.2017. Имеется перевод на русский язык: http://citforum.ru/database/digest/sql1.shtml, дата обращения 22.03.2017.

[27]. Philip L. Frana. Oral history interview with Donald D. Chamberlin. Charles Babbage Institute, 2001. http://hdl.handle.net/11299/107215, дата обращения 22.03.2017

[28]. Malcolm Atkinson, Francois Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik. The Object-Oriented Database System Manifesto. Proc. 1st International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan. New York, N.Y.: Elsevier Science, 1989, pp. 223-240. Имеется перевод на русский язык: http://citforum.ru/database/classics/oo_manifesto/, дата обращения 25.03.2017

[29]. M. Stonebraker, L. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, Ph. Bernstein, D. Beech. Third-Generation Data Base System Manifesto. ACM SIGMOD Record 19, № 3, 1990, pp. 31-44. Имеется перевод на русский язык: http://citforum.ru/database/classics/manifest/, дата обращения 25.03.2017

[30]. Michael Stonebraker. The Land Sharks Are on the Squawk Box. Communications of the ACM, vol. 59, issue 2, 2016, pp. 74-83

[31]. С.Д. Кузнецов. Объектно-реляционные базы данных: прошедший этап или недооцененные возможности? Труды ИСП РАН, т. 13, часть 2, 2007, стр. 115-140

[32]. M. N. Grinev, S. D. Kuznetsov. UQL: A UML-based Query Language for Integrated Data. Programming and Computer Software, vol. 28, issue 4, 2002, pp 189–196. DOI: 10.1023/A:1016366916304

[33]. M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. R. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-Store: A Column-Oriented DBMS. In VLDB, 2005, pp. 553–564

[34]. Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, Pat Helland. The End of an Architectural Era (It's Time for a Complete Rewrite). Proceedings of VLDB, 2007, pp. 1150-1160. Имеется перевод на русский

язык: http://citforum.ru/database/articles/end_of_arch_era/, дата обращения 25.03.2017
[35]. Vertica, https://www.vertica.com/, дата обращения 26.03.2017
[36]. VoltDB, https://www.voltdb.com/, дата обращения 26.03.2017
[37]. Кузнецов С.Д., Посконин А.В. Системы управления данными категории NoSQL. Программирование, том 40, № 6, стр. 34-47, 2014, http://www.ispras.ru/publications/2014/nosql_data_management_systems/, дата обращения 26.03.2017
[38]. Daniel Newman. Data As A Service: The Big Opportunity For Business. https://www.forbes.com/sites/danielnewman/2017/02/07/data-as-a-service-the-big-opportunity-for-business/#47708f3c24d9, дата обращения 26.03.2017
[39]. Сергей Кузнецов. Когда, как и зачем стоит применять теорему CAP? Открытые системы. СУБД, № 04, 2012, стр. 56-59, https://www.osp.ru/os/2012/04/13015765/, дата обращения 26.03.2017
[40]. Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao. Frank Pellow, Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Data Mining and Knowledge Discovery, vol. 1, issue 1, 1997, pp 29–53
[41]. Carl Nolan. Manipulate and Query OLAP Data Using ADOMD and Multidimensional Expressions. Microsoft Systems Journal, vol 14, № 8, 1999, https://www.microsoft.com/msj/0899/mdx/mdx.aspx, дата обращения 28.03.2017
[42]. Oracle Database, Data Warehousing Guide, 10g Release 2 (10.2), 2005, https://docs.oracle.com/cd/B19306_01/server.102/b14223.pdf, дата обращения 28.03.2017
[43]. Philip A. Bernstein, Umeshwar Dayal. An Overview of Repository Technology. In VLDB '94, Proceedings of the 20th International Conference on Very Large Data Bases, September 12 - 15, 1994, pp. 705-713
[44]. James Gosling, Bill Joy, Guy Steele. The Java Language Specification. Addison Wesley, 1996. http://titanium.cs.berkeley.edu/doc/java-langspec-1.0.pdf, дата обращения 28.03.2017
[45]. Sun Microsystems. JavaBeans API specification, version 1.01, August 1997, http://download.oracle.com/otn-pub/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/beans.101.pdf?AuthParam=1490694903_89224574569a2575b5c6d8a54a8c2ebc, дата обращения 28.03.2017
[46]. Graham Hamilton, Rick Cattell. JDBC: A Java SQL API, Version 1.20. Sun Microsystems Inc., January 1997, http://www.dcs.ed.ac.uk/teaching/cs2/prac6/jdbc-spec-0120.pdf, дата обращения 29.03.2017
[47]. Surajit Chaudhuri, Gerhard Weikum. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In Proceeding VLDB '00 Proceedings of the 26th International Conference on Very Large Data Bases, September 10 - 14, 2000, pp. 1-10, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.9260, дата обращения 29.03.2017
[48]. Michael Stonebraker, Uğur Çetintemel. «One Size Fits All»: An Idea Whose Time Has Come and Gone. In Proceeding ICDE '05 Proceedings of the 21st International Conference on Data Engineering, April 05 - 08, 2005, pp. 2-11. Перевод на русский язык: http://citforum.ru/database/articles/one_size_fits_all/, дата обращения 29.03.2017

[49]. Database SQL Tuning Guide. Chapter 4, Query Optimizer Concepts. https://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm#TGSQL192, дата обращения 29.03.2017

[50]. Tomasz Imielinski, Heikki Mannila. A database perspective on knowledge discovery. Communications of the ACM, vol. 39, issue 11, 1996, pp. 58-64

[51]. Surajit Chaudhuri, Vivek Narasayya. AutoAdmin «what-if» index analysis utility. In Proceeding SIGMOD '98 Proceedings of the 1998 ACM SIGMOD international conference on Management of data, Seattle, Washington, USA — June 01 - 04, 1998, pp. 367-378

[52]. Nicolas Bruno, Surajit Chaudhuri, Arnd Christian Kцnig, Vivek Narasayya, Ravi Ramamurthy, and Manoj Syamala. AutoAdmin Project at Microsoft Research: Lessons Learned. Bulletin of the Technical Committee on Data Engineering, Vol. 34, №. 4, December 2011, pp. 12-19

[53]. Pete Belknap, John Beresniewicz, Benoit Dageville, Karl Dias, Uri Shaft, Khaled Yagoub. A Decade of Oracle Database Manageability. Bulletin of the Technical Committee on Data Engineering, Vol. 34, №. 4, December 2011, pp. 20-27

[54]. С.Д. Кузнецов, А.А. Прохоров. Алгоритмы управления буферным пулом СУБД при работе с флэш-накопителями. Труды ИСП РАН, том 23, 2012, стр. 173-194. DOI: 10.15514/ISPRAS-2012-23-11

[55]. Michael Stonebraker. The Design of the POSTGRES Storage System. In Proceeding VLDB '87 Proceedings of the 13th International Conference on Very Large Data Bases, September 01 - 04, 1987, pp. 289-300

[56]. Justin DeBrabant, Joy Arulraj, Andrew Pavlo, Michael Stonebraker, Stanley B. Zdonik, Subramanya Dulloor. A Prolegomenon on OLTP Database Systems for Non-Volatile Memory. ADMS 2014, Fifth International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures, Monday, September 1, 2014, pp. *57-63*

[57]. Ismail Oukid, Wolfgang Lehner, Towards a Single-Level Database Architecture on Non-Volatile Memory (Presentation Abstract). 8th Annual Non-Volatile Memories Workshop 2017, University of California, San Diego - Price Center Ballroom East, March 12-14, 2017, http://nvmw.ucsd.edu/2017/assets/abstracts/50, дата обращения 31.03.2017

[58]. Antonino Tumeo, Simone Secchi, and Oreste Villa. Designing Next-Generation Massively Multithreaded Architectures for Irregular Applications. Computer, vol. 45, № 8, 2012, pp. 53-61

[59]. Harald Schöning. Tamino - A DBMS Designed for XML. In Proceeding ICDE '01 Proceedings of the 17th International Conference on Data Engineering, April 02 - 06, 2001, p. 149

[60]. Andrey Fomichev, Maxim Grinev, Sergey Kuznetsov. Sedna: A Native XML DBMS. Lecture Notes in Computer Science, vol 3831, 2006, pp. 272-281. DOI: 10.1007/11611257_25

[61]. Michael Franklin, Alon Halevy, David Maier. From Databases to Dataspaces: A New Abstraction for Information Management, SIGMOD Record, Vol. 34, No. 4, Dec. 2005, pp. 27-33. Перевод на русский язык: http://citforum.ru/database/articles/from_db_to_ds/, дата обращения 2 апреля 2017 г.

[62]. Paresh V. Virparia, Sanjay H. Buch, Roohana F. Parabia. Trade and Tricks: Traditional vs. Virtual Data Warehouse, An International Journal of Advanced Engineering & Applications, January 2010, pp.:225-239

[63]. Bill Inmon. The Elusive Virtual Data Warehouse. http://www.b-eye-network.com/view/9956, March 19, 2009, дата обращения 2 апреля 2017 г.

[64]. Vijay Gadepally, Peinan Chen, Jennie Duggan, Aaron Elmore, Brandon Haynes, Jeremy Kepner, Samuel Madden, Tim Mattson, Michael Stonebraker. The BigDAWG Polystore System and Architecture. In Proceedings 2016 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1-6, https://arxiv.org/pdf/1609.07548.pdf, дата обращения 2 апреля 2017 г.

[65]. Ippokratis Pandis, Ryan Johnson, Nikos Hardavellas, Anastasia Ailamaki. Proceedings of the VLDB Endowment, Vol. 3, No. 1, 2010, pp. 928-939. Перевод на русский язык: http://citforum.ru/database/articles/ailamaki_vldb2010/, дата обращения 2 апреля 2017 г.

[66]. Ежемесячный семинар Московской Секции ACM SIGMOD. С.Д. Кузнецов. 25 лет прогнозов: что день грядущий нам готовит? Часть 1, 24 декабря 2015 г., http://synthesis.ipi.ac.ru/sigmod/seminar/s20151224, Часть 2, 21 января 2016 г., http://synthesis.ipi.ac.ru/sigmod/seminar/s20160121, дата обращения 03.04.2017

Data Management: 25 Years of Forecasts

Kuznetsov S.D. <kuzloc@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
Moscow Institute of Physics and Technology (State University)
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia

Abstract. In October 2013, the eighth meeting of researchers in the field of databases was held. The first such meeting took place in February 1988, so that 25 years passed between them. After each meeting, a report was published containing an overview of the current state of the field and a research program for the nearest future, a kind of set of forecasts for the development of research activities. This paper looks at the most interesting forecasts from the reports of the research meetings, discusses how they proved to be valid, to what extent they were true or not. Among the various problems of database technology under consideration are the following: the role of specialized hardware in building effective DBMS; SQL and database applications; perspectives of object-relational extensions; distributed heterogeneous database systems; databases and Web; databases and data warehouses, OLAP and data mining; component organization of DBMS; query optimization criteria; self-tuning and self-management of DBMS; DBMS architecture and new hardware capabilities: SSD, non-volatile memory, massively multithreaded processors; specialized DBMS; data fusion and data spaces; the Big Data problem and the response to it in the database community; architectural shifts in computing.

Keywords: database research community reports; technology forecasts; data analysis; database machine; SQL; manifest of future database systems; scalability; heterogeneity; distribution; data warehouse; extendibility; query optimization; self-tuning; SSD; non-volatile memory; data space; Big Data

DOI: 10.15514/ISPRAS-2017-29(2)-5

For citation: Kuznetsov S.D. Data Management: 25 years of Forecasts. *Trudy ISP RAN/Proc.ISP RAS*, vol. 29, issue 2, 2017, pp. 117-160 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-5

References

[1]. Philip A. Bernstein, Umeshwar Dayal, David J. DeWitt et al. Future Directions in DBMS Research. ACM SIGMOD Record, vol. 18, № 1, 1989, pp. 17-26
[2]. Sergey Kuznetsov. Future directions of database research: Ten years later. http://citforum.ru/database/articles/future_01.shtml (in Russian), accessed 02.04.2017
[3]. Abraham Silberschatz, Michael Stonebraker, and Jeffrey D. Ullman. Database Systems: Achievements and Opportunities. Communications of the ACM, vol. 34, № 10, 1991, pp. 110-120
[4]. Abraham Silberschatz, Michael Stonebraker, Jeffrey D. Ullman: Database Research: Achievements and Opportunities Into the 21st Century. SIGMOD Record, vol. 25, № 1, 1996, pp. 52-63
[5]. Avi Silberschatz, Stan Zdonik et al. Strategic Directions in Database Systems – Breaking Out of the Box. ACM Computing Surveys, vol. 28, No. 4, Dec 1996, 764-778
[6]. Philip A. Bernstein, Michael L. Brodie, Stefano Ceri et. al. The Asilomar Report on Database Research. SIGMOD Record, vol. 27, № 4, 1998, pp. 74-80
[7]. Serge Abiteboul, Rakesh Agrawal, Philip A. Bernstein et. al. The Lowell Database Research Self-Assessment. Communications of the ACM, vol. 48, № 5, 2005, pp. 111-118
[8]. Sergey Kuznetsov. Major problems and current tasks of database. http://citforum.ru/database/articles/problems/, 2005 (in Russian), accessed 02.04.2017
[9]. Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, et. al. The Claremont Report on Database Research. Communications of the ACM, vol. 52, No. 6, 2009, pp. 56-65
[10]. Daniel Abadi, Rakesh Agrawal, Anastasia Ailamaki, et. al. The Beckman Report on Database Research. ACM SIGMOD Record, vol. 43, issue 3, September 2014, pp. 61-70.
[11]. Keen, P. G. W. and M. S. Scott Morton. Decision support systems: an organizational perspective. Reading, Mass., Addison-Wesley Pub. Co., 1978
[12]. Sergey Kuznetsov. MapReduce: inside, outside or side of parallel DBMS? Trudy ISP RAN/Proc. ISP RAS, vol. 19, 2010, pp. 35-40 (in Russian)
[13]. Edward A.Feigenbaum and Pamela McCorduck. The fifth generation: Japan's computer challenge to the world. Creative Computing Magazine, vol. 10, No. 08, August 1984, pp. 103-111.
[14]. David DeWitt, Jim Gray. Parallel database systems: the future of high performance database systems. Communications of the ACM, vol. 35, issue 6, June 1992, pp. 85-98
[15]. Marianne Winslett. Jim Gray speaks out. ACM SIGMOD Record, Volume 32, Issue 1, March 2003, pp. 53-61
[16]. Gartnet IT Glassary. Database Appliances. http://www.gartner.com/it-glossary/database-appliances, accessed 22.03.2017
[17]. Exadata. https://ru.wikipedia.org/wiki/Exadata, accessed 22.03.2017
[18]. Michael Stonebraker. My Top 10 Assertions About Data Warehouses. BLOG@CACM, August 26, 2010

[19]. Stepan O. Prikazchikov, Pavel S. Kostenetskiy. Using graphics accelerators for query processing over compressed data in parallel database systems. Bulletin of the South Ural State University. Series «Computational Mathematics and Software Engineering», vol. 4, issue 1, 2015, pp. 64-70 (in Russian)

[20]. M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhause. CHORUS Distributed Operating Systems. Computing Systems, vol. I, No. 4, Fall 1988, pp. pp. 305-370

[21]. Igor Burdonov, Victor Ivannikov, German Kopytov, Alexander Kosachev, Sergei Kuznetsov. The CLOS project: Towards an object-oriented environment for application development. Next Generation Information System Technology. Lecture Notes in Computer Science (LNCS), vol. 504, 1991, pp. 422-427, DOI: 10.1007/3-540-54141-1_23

[22]. David B. Golub , Daniel P. Julin , Richard F. Rashid , Richard P. Draves , Randall W. Dean , Alessandro Forin , Joseph Barrera , Hideyuki Tokuda , Gerald Malan , David Bohman. Microkernel operating system architecture and mach. In Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, 1992, pp. 11-30

[23]. QNX Operating Systems. http://www.qnx.com/content/qnx/en/products/neutrino-rtos/index.html, accessed 22.03.2017

[24]. Andrew Tanenbaum, Raja Appuswamy, Herbert Bos, Lorenzo Cavallaro, Cristiano Giuffrida, Tomáš Hrubý, Jorrit Herder, Erik van der Kouwe, David van Moolenbroek. MINIX 3: Status Report and Current Research. ;login, vol. 35, № 3, June 2010, pp. 7-13

[25]. S.D. Kuznetsov. SQL Relational Database Language Standards: An Brief Overview. DBMS, № 2, 1996, pp. 6-36 (in Russian)

[26]. The 1995 SQL Reunion: People, Projects, and Politics. Edited by Paul McJones, August 20, 1997 (2nd edition), http://www.mcjones.org/System_R/SQL_Reunion_95/SRC-1997-018.pdf, accessed 22.03.2017

[27]. Philip L. Frana. Oral history interview with Donald D. Chamberlin. Charles Babbage Institute, 2001. http://hdl.handle.net/11299/107215, accessed 22.03.2017

[28]. Malcolm Atkinson, Francois Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik. The Object-Oriented Database System Manifesto. Proc. 1st International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan. New York, N.Y.: Elsevier Science, 1989, pp. 223-240.

[29]. M. Stonebraker, L. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, Ph. Bernstein, D. Beech. Third-Generation Data Base System Manifesto. ACM SIGMOD Record 19, № 3, 1990, pp. 31-44

[30]. Michael Stonebraker. The Land Sharks Are on the Squawk Box. Communications of the ACM, vol. 59, issue 2, 2016, pp. 74-83

[31]. S.D. Kuznetsov. Object-relational databases: past stage or undervalued capabilities? Trudy ISP RAN/Proc. ISP RAS, vol. 13, part 2, 2007, pp. 115-140 (in Russian)

[32]. M. N. Grinev, S. D. Kuznetsov. UQL: A UML-based Query Language for Integrated Data. Programming and Computer Software, vol. 28, issue 4, 2002, pp 189–196. DOI: 10.1023/A:1016366916304

[33]. M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. R. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-Store: A Column-Oriented DBMS. In VLDB, 2005, pp. 553–564

[34]. Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, Pat Helland. The End of an Architectural Era (It's Time for a Complete Rewrite). Proceedings of VLDB, 2007, pp. 1150-1160

[35]. Vertica, https://www.vertica.com/, accessed 26.03.2017

[36]. VoltDB, https://www.voltdb.com/, accessed 26.03.2017
[37]. S. D. Kuznetsov, A. V. Poskonin. Programming and Computer Software. November 2014, Volume 40, Issue 6, pp 323-332. DOI: 10.1134/S0361768814060152
[38]. Daniel Newman. Data As A Service: The Big Opportunity For Business. https://www.forbes.com/sites/danielnewman/2017/02/07/data-as-a-service-the-big-opportunity-for-business/#47708f3c24d9, accessed 26.03.2017
[39]. Sergey Kuznetsov. When, How and Why the CAP Theorem Should Be Applied? Open Systems. DBMS, № 04, 2012, pp. 56-59 (in Russian)
[40]. Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao. Frank Pellow, Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Data Mining and Knowledge Discovery, vol. 1, issue 1, 1997, pp 29–53
[41]. Carl Nolan. Manipulate and Query OLAP Data Using ADOMD and Multidimensional Expressions. Microsoft Systems Journal, vol 14, № 8, 1999, https://www.microsoft.com/msj/0899/mdx/mdx.aspx, дата обращения 28.03.2017
[42]. Oracle Database, Data Warehousing Guide, 10g Release 2 (10.2), 2005, https://docs.oracle.com/cd/B19306_01/server.102/b14223.pdf, accessed 28.03.2017
[43]. Philip A. Bernstein, Umeshwar Dayal. An Overview of Repository Technology. In VLDB '94, Proceedings of the 20th International Conference on Very Large Data Bases, September 12 - 15, 1994, pp. 705-713
[44]. James Gosling, Bill Joy, Guy Steele. The Java Language Specification. Addison Wesley, 1996. http://titanium.cs.berkeley.edu/doc/java-langspec-1.0.pdf, accessed 28.03.2017
[45]. Sun Microsystems. JavaBeans API specification, version 1.01, August 1997, http://download.oracle.com/otn-pub/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/beans.101.pdf?AuthParam=1490694903_89224574569a2575b5c6d8a54a8c2ebc, accessed 28.03.2017
[46]. Graham Hamilton, Rick Cattell. JDBC: A Java SQL API, Version 1.20. Sun Microsystems Inc., January 1997, http://www.dcs.ed.ac.uk/teaching/cs2/prac6/jdbc-spec-0120.pdf, accessed 29.03.2017
[47]. Surajit Chaudhuri, Gerhard Weikum. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In Proceeding VLDB '00 Proceedings of the 26th International Conference on Very Large Data Bases, September 10 - 14, 2000, pp. 1-10
[48]. Michael Stonebraker, Uğur Çetintemel. «One Size Fits All»: An Idea Whose Time Has Come and Gone. In Proceeding ICDE '05 Proceedings of the 21st International Conference on Data Engineering, April 05 - 08, 2005, pp. 2-11
[49]. Database SQL Tuning Guide. Chapter 4, Query Optimizer Concepts. https://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm#TGSQL192, accessed 29.03.2017
[50]. Tomasz Imielinski, Heikki Mannila. A database perspective on knowledge discovery. Communications of the ACM, vol. 39, issue 11, 1996, pp. 58-64
[51]. Surajit Chaudhuri, Vivek Narasayya. AutoAdmin «what-if» index analysis utility. In Proceeding SIGMOD '98 Proceedings of the 1998 ACM SIGMOD international conference on Management of data, Seattle, Washington, USA — June 01 - 04, 1998, pp. 367-378
[52]. Nicolas Bruno, Surajit Chaudhuri, Arnd Christian Kцnig, Vivek Narasayya, Ravi Ramamurthy, and Manoj Syamala. AutoAdmin Project at Microsoft Research: Lessons Learned. Bulletin of the Technical Committee on Data Engineering, Vol. 34, №. 4, December 2011, pp. 12-19

[53]. Pete Belknap, John Beresniewicz, Benoit Dageville, Karl Dias, Uri Shaft, Khaled Yagoub. A Decade of Oracle Database Manageability. Bulletin of the Technical Committee on Data Engineering, Vol. 34, №. 4, December 2011, pp. 20-27

[54]. Kuznetsov S.D., Prokhorov A.A. Flash-based algorithms of database buffer management. Trudy ISP RAN/Proc. ISP RAS, vol. 23, 2012, pp. 173-194 (in Russian). DOI: 10.15514/ISPRAS-2012-23-11

[55]. Michael Stonebraker. The Design of the POSTGRES Storage System. In Proceeding VLDB '87 Proceedings of the 13th International Conference on Very Large Data Bases, September 01 - 04, 1987, pp. 289-300

[56]. Justin DeBrabant, Joy Arulraj, Andrew Pavlo, Michael Stonebraker, Stanley B. Zdonik, Subramanya Dulloor. A Prolegomenon on OLTP Database Systems for Non-Volatile Memory. ADMS 2014, Fifth International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures, Monday, September 1, 2014, pp. *57-63*

[57]. Ismail Oukid, Wolfgang Lehner, Towards a Single-Level Database Architecture on Non-Volatile Memory (Presentation Abstract). 8th Annual Non-Volatile Memories Workshop 2017, University of California, San Diego - Price Center Ballroom East, March 12-14, 2017, http://nvmw.ucsd.edu/2017/assets/abstracts/50, accessed 31.03.2017

[58]. Antonino Tumeo, Simone Secchi, and Oreste Villa. Designing Next-Generation Massively Multithreaded Architectures for Irregular Applications. Computer, vol. 45, № 8, 2012, pp. 53-61

[59]. Harald Schöning. Tamino - A DBMS Designed for XML. In Proceeding ICDE '01 Proceedings of the 17th International Conference on Data Engineering, April 02 - 06, 2001, p. 149

[60]. Andrey Fomichev, Maxim Grinev, Sergey Kuznetsov. Sedna: A Native XML DBMS. Lecture Notes in Computer Science, vol 3831, 2006, pp. 272-281. DOI: 10.1007/11611257_25

[61]. Michael Franklin, Alon Halevy, David Maier. From Databases to Dataspaces: A New Abstraction for Information Management, SIGMOD Record, Vol. 34, No. 4, Dec. 2005, pp. 27-33

[62]. Paresh V. Virparia, Sanjay H. Buch, Roohana F. Parabia. Trade and Tricks: Traditional vs. Virtual Data Warehouse, An International Journal of Advanced Engineering & Applications, January 2010, pp.:225-239

[63]. Bill Inmon. The Elusive Virtual Data Warehouse. http://www.b-eye-network.com/view/9956, March 19, 2009, accessed 2 апреля 2017 г.

[64]. Vijay Gadepally, Peinan Chen, Jennie Duggan, Aaron Elmore, Brandon Haynes, Jeremy Kepner, Samuel Madden, Tim Mattson, Michael Stonebraker. The BigDAWG Polystore System and Architecture. In Proceedings 2016 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1-6, https://arxiv.org/pdf/1609.07548.pdf, дата обращения 2 апреля 2017 г.

[65]. Ippokratis Pandis, Ryan Johnson, Nikos Hardavellas, Anastasia Ailamaki. Proceedings of the VLDB Endowment, Vol. 3, No. 1, 2010, pp. 928-939

[66]. Monthly Seminar of the Moscow Section of ACM SIGMOD. 25 years of Forecasts: What is the future preparing for us? Part 1, December 24th, 2015, http://synthesis.ipi.ac.ru/sigmod/seminar/s20151224, Part 2, January 21st, 2016, http://synthesis.ipi.ac.ru/sigmod/seminar/s20160121

Обзор и экспериментальное сравнение методов кластеризации текстов[1]

[1,2] *П. А. Пархоменко <parhomenko @ispras.ru>*
[1,3] *А. А. Григорьев <agrigorev@ispras.ru>*
[1] *Н. А. Астраханцев <astrakhantsev@ispras.ru>*
[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1
[3] *Национальный исследовательский университет «Высшая школа экономики»*
101000 Россия, Москва, ул. Мясницкая, д.20

Аннотация. Кластеризация текстовых документов применяется во многих приложениях, таких как информационный поиск, исследовательский поиск, определение спама. Этой задаче посвящено множество научных работ, однако в настоящее время остается недостаточно изученным влияние специфики научных статей, в частности принадлежности документов одной предметной области или недоступности полных текстов, на эффективность кластеризации. В данной работе предлагаются обзор и экспериментальное сравнение методов кластеризации текстовых документов в приложении к научным статьям. Исследуются методы, основанные на мешке слов, извлечении терминологии, тематическом моделировании, а также векторном представлении слов (word embedding) и документов, полученном с помощью искусственных нейронных сетей (word2vec, paragraph2vec).

Ключевые слова: кластеризация текстовых документов; мешок слов; извлечение терминологии; тематическое моделирование; векторное представление; искусственные нейронные сети

DOI: 10.15514/ISPRAS-2017-29(2)-6

Для цитирования: Пархоменко П.А., Григорьев А.А, Астраханцев Н.А. Обзор и экспериментальное сравнение методов кластеризации текстов. Труды ИСП РАН, том 29, вып. 2, 2017 г., стр. 161-200. DOI: 10.15514/ISPRAS-2017-29(2)-6

[1] Эта работа поддержана грантом РФФИ №14-07-00692

1. Введение

Кластеризация текстовых документов, то есть разбиение множества документов на близкие по смыслу подмножества, является фундаментальной задачей обработки текстов. Ее результаты используются как для непосредственного анализа исходного множества документов, так и для информационного поиска [1], определения спама [2], помощи в проведении судебномедицинских экспертиз [3] и социологических исследований [4].

Особого внимания заслуживает кластеризация научных статей. В настоящее время их количество настолько велико, что прочитать все их, даже в одной области знаний, не представляется возможным; более того, возникают серьезные сложности и с самим поиском нужных статей, особенно при отсутствии четкого понимания предметной области или самой информационной потребности.

Возможное решение заключается в навигации на основе кластеров (clustering based navigation) [5] и других методов исследовательского поиска, в которых часто используется кластеризация как один из этапов [6].

К настоящему времени произведено множество обзоров и экспериментальных сравнений методов кластеризации, но в большей их части не рассматриваются современные методы, например векторные представления слов, полученные с помощью нейронных сетей (word embedding), а также не учитывается специфика научных статей, в частности, тот факт, что во многих практических приложениях необходимо кластеризовать статьи, принадлежащие одной предметной области, по более узким направлениям, причем полные тексты статей не всегда доступны.

Данная работа призвана восполнить эти недостатки путем анализа и экспериментального сравнения как классических, так и современных методов кластеризации текстов в приложении к научным статьям. Статья устроена следующим образом. Во втором разделе приводится обзор существующих работ, в том числе других обзоров и экспериментальных сравнений. В следующем разделе описывается методика экспериментальных исследований. Четвертый раздел посвящен результатам экспериментального сравнения и их обсуждению. Далее приводится заключение, подводящее итог статьи и предлагающее направления дальнейшей работы.

2. Существующие обзоры и экспериментальные сравнения

В большинстве обзоров, посвященных методам кластеризации текстов, документы представляются как векторы, в виде мешка слов (bag of words, bow), так что кластеризация документов рассматривается именно как кластеризация bowвекторов.

Так, Andrews и Fox в работе 2007 года [7] описывают способы представления набора документов в виде векторной модели, в том числе различные способы предобработки текстов, а также алгоритмы их кластеризации, такие как

модификации k-means (или метод k-средних), EM-алгоритм и спектральная кластеризация. Так как одним из главных недостатков представления документов в виде мешка слов является высокая размерность и разреженность получаемых векторов, авторы также представляют методы понижения размерности векторного пространства.

Рассматривая разделительные (partitional) алгоритмы кластеризации документов, в частности k-means, более детально, Huang представляет описание и сравнение мер близости между bow-векторами [8]. В статье описаны шесть различных мер близости, между которыми проведено экспериментальное сравнение на алгоритме k-means; лучшие результаты по метрикам чистоты (purity) и энтропии показала кластеризация, использующая в качестве меры близости коэффициент Жаккара (Jaccard coeficient) и коэффициент корреляции Пирсона (Pearson correlation coefficient).

Sathiyakumari и др. [9] также рассматривают кластеризацию документов только применительно к их представлению в виде мешка слов. Они выделяют четыре группы методов кластеризации таких представлений: разделительная кластеризация, иерархическая кластеризация, k-средних и EM-алгоритм, хотя во многих других работах k-means включается в группу разделительных алгоритмов [10, 11].

Как видно в вышеупомянутых работах, кластеризация документов обычно сводится к кластеризации их векторных представлений в виде мешка слов. Кластеризации векторов в общем случае, безотносительно к текстовым документам, также посвящено множество работ [10, 12].

Более широкий спектр возможных векторных представлений документа разбирается в одной из глав книги Mining Text Data [13]. В частности, в ней описываются методы, использующие в качестве признаков документов часто встречающиеся наборы слов, а также методы тематического моделирования. Кроме того, обозреваются подходы к онлайн-кластеризации текстов, использованию графовых методов кластеризации (в случае если между текстами существуют связи) и имеющейся заранее информации для кластеризации на основе алгоритмов частичного обучения (semi-supervised).

В некоторых обзорах авторы отдельно выделяют методы так называемой семантической кластеризации. Saiyad и др. [14] считают определяющим отличием семантической кластеризации от традиционной, основанной на мешке слов, использование семантических отношений между словами документов. Авторы относят к методам семантической кластеризации несколько групп алгоритмов: алгоритмы, основанные на онтологиях, таких как WordNet; алгоритмы, использующие в качестве признаков документа наборы связанных по смыслу слов; а также алгоритмы, основанные на графах концептов или именованных сущностей с семантическими отношениями между ними. Кроме того, к этой группе авторами отнесены алгоритмы, использующие латентное семантическое индексирование, хотя обычно они считаются методами тематического моделирования.

3. Методы кластеризации документов

Как правило, процесс кластеризации текстовых документов можно логически разделить на два основных этапа. На первом этапе текстовые представления документов по определенным правилам переводят в векторные представления, для того чтобы на втором этапе применить к полученным векторам методы кластеризации, основанные на расстоянии между ними.

Ниже будут сначала представлены различные способы отображения документов в векторное пространство, а затем — методы кластеризации векторов. Кроме того, будут описаны возможные варианты предобработки текстов и меры, с помощью которых обычно оценивают эффективность кластеризации.

3.1 Методы на основе bag-of-words

Наиболее простым представлением документов в векторном виде является так называемый мешок слов. В данном случае на основе набора документов строится словарь из всех встречающихся в нем n-грамм, где n меньше или равно какому-то заранее заданному значению. Документ представляется набором признаков, каждому из которых соответствует одна n-грамма из словаря.

BinaryBOW. В простейшем, бинарном, случае данный признак принимает значение 1 в случае, если в документе встречается соответствующая n-грамма, и 0 — иначе.

CountBOW. Предполагая, что значимость появления n-граммы в документе тем больше, чем чаще она в нем появляется, этот метод учитывает количество вхождений n-граммы в документе, помимо самого факта вхождения. Таким образом, каждый признак показывает, сколько раз соответствующая n-грамма появляется в документе.

TF-IDF. Для того чтобы снизить влияние длины текста на его признаки, используется нормализация количества вхождений n-грамм на размер текста. Тогда каждый признак принимает вид частоты n-граммы (term frequency или tf) [15], которая считается как отношение количества вхождений соответствующей n-граммы к общему количеству слов в документе. Поскольку, как правило, значимость появления в документе различных n-грамм различается, применяются различные схемы взвешивания признаков.

Наиболее широко используемая из таких схем — TF-IDF. Она использует предположение о том, что значимость n-граммы прямо пропорциональна частоте ее появления в документе и обратно пропорциональна доле документов в наборе, в которых эта n-грамма встречается. Таким образом, наибольший вес получает n-грамма, часто встречающаяся в одном документе, но не встречающаяся в остальных, а значит — отличающая этот документ от остальных. Признаки документов в этом подходе представляют собой произведение двух величин, частоты n-граммы и обратной частоты документа (inverse document frequency):

$$TF \cdot IDF(t_i, d_j, D) = tf(t_i, d_j) \cdot \log \frac{|D|}{|(d_j \supset t_i)|},$$

где $tf(t_i, d_j)$ — частота n-граммы t_i в документе d_j, D — набор документов, $|(d_j \supset t_i)|$ — все такие документы в наборе, в которых встречается n-грамма t_i.

BM25. Whissel и др. [16] экспериментально показывают, что лучшие результаты в кластеризации текстов демонстрирует другой вариант взвешивания значимости слов: метод BM25. В нем ограничивается значимость частоты n-граммы, а также она не только нормализуется по его размеру, но и ограничивается сверху, что позволяет избежать присваивания слову слишком большого веса [13]. Значение признаков для n-граммы t_i в документе d_j в этом методе рассчитывается по следующей формуле.

$$idf(t_i) \cdot \frac{tf(t_i, d_j) \cdot (k_1 + 1)}{k_1 \cdot (1 - b + b \cdot \frac{|d_j|}{|d_{avg}|}) + tf(t_i, d_j)},$$

где $|d_j|$ — длина данного документа; $|d_{avg}|$ — средняя длина документов в наборе; k_1 и b — свободные параметры.

Стоит отметить, что в методах использующих в качестве признаков n-граммы, как правило учитываются не все из них. Есть несколько способов отбрасывать незначимые n-граммы. Один из них: не учитывать те n-граммы, количество вхождений в наборе документов которых ниже, чем определенный заранее порог. Другой способ: отсортировать все n-граммы в словаре по частоте употребления и учитывать только m первых, где m также задается заранее. Третий способ: не учитывать n-граммы, входящие в слишком большую долю документов из набора, поскольку такие слова, как правило, не несут смысловой нагрузки, позволяющей характеризовать документ. Также можно не учитывать слова, входящие в заранее подготовленный список стоп-слов.

3.2 Bag-of-terms

Голомазов [17] использует термины в качестве признаков документа для кластеризации и классификации документов. В таком подходе значимыми считаются не все n-граммы, а только определенный набор заранее выделенных терминов. При этом, построив матрицу вхождений терминов в документы, с ними можно оперировать точно так же, как с обычными n-граммами в выше описанных методах.

Методы кластеризации с использованием извлеченных терминов полезны в случае документов небольшой длины из узкой предметной области. Например, D. Pinto [18] и др. кластеризуют аннотации научных статей используя оригинальные методы извлечения терминов.

3.3 Тематическое моделирование

Другие способы извлечения признаков из документов часто называют методами тематического моделирования, так как в них каждый результирующий признак можно отнести к определенной теме, представленной в наборе документов. К основным методам тематического моделирования относятся Latent Semantic Analysis (LSA), Nonnegative Matrix Factorization (NMF), Probablistic LSA (pLSA) и Latent Dirichlet Allocation (LDA).

LSA. Метод LSA (также называемый latent semantic indexing, LSI), предложенный Deerwester и др. [19], использует для понижения размерности метод сингулярного разложения (singular value decomposition или SVD). SVD позволяет отобразить данные в новое пространство меньшей размерности, в котором все базисные вектора будут ортогональны, а разброс данных в ортогональной проекции на эти оси — максимальным. В нем изначальная матрица данных X (как правило, TF-IDF в случае кластеризации текстов) раскладывается на 3 следующим образом: $X = U\Sigma V^T$, где U и V — матрицы, состоящие из левых и правых сингулярных векторов матрицы X, а Σ — диагональная матрица, состоящая из сингулярных значений X. Результирующая же матрица, строки которой соответствуют векторам документов, имеет имеет вид $T = XV_l$, где V_l — первые l строк матрицы V, соответствующие l наибольшим сингулярным значениям из Σ.

NMF. Xu и др. [20] предлагают использовать для кластеризации вектора, полученные методом NMF, в котором, как и в LSA, данные отображаются в новое пространство с целью максимизировать разброс по каждой из его осей. Отличия NMF от LSA состоят в том, что в NMF новое пространство может быть не ортогонально, а также принимает только матрицы, в которых все элементы неотрицательны. Как утверждается в оригинальной статье, это позволяет достичь более сильного соответствия между результирующими осями и кластерами документов.

pLSA. В работе Hofmann и др. [21] вводится метод probabilistic latent semantic allocation (pLSA). На вход методу подаются набор слов W, набор документов D, а также количество тем $|T|$ в этом наборе. В результате он генерирует две матрицы. Элементы матрицы $\Phi_{|W|\times|T|}$ соответствуют вероятности того, что определенное слово относится к определенной теме: $\phi_{wt} = p(w|t)$. Элементы $\Theta_{|T|\times|D|}$ соответствуют вероятности того, что определенная тема встречается в документе: $\theta_{td} = p(t|d)$. Эти матрицы строятся с помощью максимизации логарифма правдоподобия следующей функции.

$$L(D, \Phi, \Theta) = \sum_{d\in D}\sum_{w\in d} c(w,d) \ln \sum_{t\in T} \phi_{wt}\theta_{td} \to \max_{\Phi,\Theta}$$

где $c(w,d)$ — количество вхождений слова w в документе d. Затем в качестве представления документов используются столбцы матрицы Θ.

LDA. Другой метод моделирования документов с помощью тематического моделирования, LDA, предлагается Blei и др. [22]. Они выделяют несколько недостатков pLSA, с которыми справляется их метод. Во-первых, количество параметров pLSA линейно зависит от размера обучающего корпуса. Во-вторых, неясно, как оценивать вероятности документов не из обучающего корпуса. В отличие от pLSA, LDA делает предположения о случайном распределении векторов тем и векторов документов. И для тем, и для документов предполагается, что их вектора порождаются распределением из параметрического семейства распределений Дирихле.

3.4 Word embeddings

В 2013 году T. Миколов и др. [23] представили модель skipgram (также часто упоминается как word2vec модель). Эта модель, обученная на корпусе текстов, отображает слова в векторное пространство небольшой размерности таким образом, что расстояние между ними тем меньше, чем ближе значения этих слов. Такой эффект достигается с помощью искусственной нейронной сети, натренированной предсказывать по вектору слова его контекст; таким образом слова, появляющиеся в схожих контекстах, отображаются в близкие вектора.

WVAvgPool. С помощью таких векторов можно получить и векторное представление документов: например Xing и др. [24] предлагают строить вектора документов простым усреднением векторов всех слов в этом документе. В экспериментах на задаче классификации текстов данный подход значительно превзошел LDA.

В 2015, через два года после публикации статьи о векторном представлении слов, Le и Mikolov [25] описали два метода векторного представления документов под общим названием Paragraph Vectors. Они используют схожую с word2vec нейросетевую модель, пытаясь по вектору, относящемуся к документу, предсказать встречающиеся в нем слова.

В первом методе, названном Distributed Memory (**PV-DM**), нейросеть по вектору документа и некоторой последовательности векторов слов тренируется предсказывать вектор следующего слова в документе.

Во втором методе, Distributed Bag of Words (**PV-DBOW**), нейросеть обучается предсказывать все слова в документе по его вектору.

Таким образом, основное отличие PV-DM от PV-DBOW состоит в том, что PV-DM учитывает информацию о порядке слов в документе.

3.5 Кластеризация признаков

WordClustering. Slonim и Tishby предлагают использовать в качестве признаков не сами nграммы, а их кластеры [26]. В таком подходе проводится два шага кластеризации: сначала кластеризуются все n-граммы в словаре, затем количество вхождений n-грамм каждого кластера используется в качестве признаков для кластеризации документов. При этом n-граммы представляются в виде столбцов TF-IDF матрицы и могут быть

кластеризованы любым методом кластеризации векторов. В оригинальной статье для кластеризации был использован Information Bottleneck Algorithm.

WVClustering. Вместо представления слов в виде столбцов TF-IDF матрицы можно также использовать вектора, полученные с помощью word2vec. Такой подход используют в своей статье Qimin и др. [27].

3.6 Семантическая кластеризация

Некоторые работы выделяют в отдельную группу методы семантической кластеризации, которые используют семантические отношения между словами для представления документов.

К таким методам относятся, в том числе, алгоритмы, основанные на онтологиях. Например, Hotho и др. [28] используют онтологии, чтобы находить в тексте синонимы и воспринимать их в качестве одного элемента, тем самым сокращая размерность пространства.

Choudhary и Bhattacharyya [29] представляют каждый текст в виде графа, чьи вершины соответствуют словам текста, а ребра — семантическим отношениям между этими словами.

Также к семантической кластеризации относят методы, использующие в качестве признаков документа лексические цепочки — наборы связанных по смыслу слов в тексте [30].

3.7 Методы кластеризации векторов

Можно выделить несколько основных групп методов кластеризации векторов

Методы разделительной кластеризации итеративно переприсваивают объектам метки кластеров пока не будет найдено оптимальное разделение на кластеры в соответствии с определенной функцией близости между объектами. Как правило, количество кластеров в таких методах определяется как параметр заранее и обозначается как k.

В кластеризации документов широко используется метод **k-means**, который изначально случайным образом выбирает центр масс для каждого из k кластеров и присваивает каждому документу метку того кластера, расстояние до центра масс которого от него меньше. А затем, на каждой итерации, алгоритм вычисляет центры масс кластеров и переприсваивает их метки документам до сходимости, то есть неизменности меток всех документов.

В отличие от предыдущего метода, **k-medoids** выбирает в качестве центра масс медианный объект из кластера, таким образом, решая проблему устойчивости к выбросам.

Иерархическая кластеризация подразумевает построение дендрограммы — дерева кластеров, в котором корнем является кластер состоящий из всего набора данных, а дети каждой вершины этого дерева соответствуют разделению этого кластера на подкластеры. Дендрограмма может строиться двумя способами: снизу вверх или сверху вниз.

В первом случае, изначально каждый объект выделен в отдельный кластер, наиболее близкие из которых затем объединяются в один. Такой подход называется **агломеративной кластеризацией**. В обратном подходе — **дивизивной кластеризации** — сначала все объекты объединены в один кластер, который затем рекурсивно разделяется на подкластеры.

Методы кластеризации основанные на плотности (density-based) определяют как кластеры плотно расположенные группы объектов. Один из широко используемых методов этой группы — **DBSCAN** — работает следующим образом. Начиная выполнение на случайном объекте выборки, он определяет, есть ли в окрестности радиуса ϵ этого объекта количество объектов, не меньшее заранее заданного параметра $minSamples$, и, если есть, определяет эту окрестность как кластер; далее все объекты, лежащие в ϵ-окрестности кластера, присваиваются этому кластеру. Это повторяется до тех пор, пока есть непосещенные объекты. Если в итоге объект оказывается не принадлежащим никакому кластеру, он помечается как шум. ϵ также задаётся как внешний параметр метода.

3.8 Меры оценки эффективности

Для оценки эффективности кластеризации традиционно выделяют два типа мер: внешние меры, использующие дополнительную (внешнюю) информацию о настоящем распределении объектов по классам, и внутренние меры, использующие только информацию о самой кластеризации.

Следуя обзору Amigo и др. [31], можно выделить следующие основные группы **внешних мер эффективности**.

Меры, основанные на сопоставлении множеств: Purity, Inverse Purity [32], F-measure. Эти меры основаны на метриках точности и полноты, стандартных для оценки эффективности информационного поиска.

Меры, основанные на подсчете пар: Jaccard Coefficient, Folkes-Mallows Index, Rand Index (RI) [33], Adjusted Rand Index (ARI) [34]. Меры из данной группы основаны на подсчете пар объектов, в зависимости от их попадания в один и тот же класс/кластер или в разные.

Меры, основанные на энтропии: собственно Entropy, а также Class Entropy [35], Variation of Information [33], Mutual Information (MI) [20], Adjusted Mutual Information(AMI) [36], Normalized Mutual Information(NMI) [37], Vmeasure [38]. Меры из данной группы основаны на подсчете пар объектов, в зависимости от их попадания в один и тот же класс/кластер или в разные.

Меры, сочетающие свойства предыдущих групп мер: BCubed Precision [39], BCubed Recall, BCubed F-measure. Эти меры усредняют стандартные метрики точности/полноты/F-меры для каждого объекта; как показали Амиго и др. [31], BCubed F-measure удовлетворяет всем предложенным в этой работе аксиомам, в отличие от остальных мер.

К настоящему времени предложено более 30 **внутренних мер эффективности** [40] и проведено множество их сравнений [40, 41, 42].

В экспериментальном сравнении 30 мер Arbelaitz и др. [40] показывают, что меры Silhouette [43], Davies–Bouldin [44], Calinski–Harabasz [45], обобщенные индексы Dunn [46], индекс COP [47] и SDbw [48] показывают лучшие результаты, чем остальные меры, при этом превосходство меры Silhouette статистически значимо (тест Шаффера с уровнем значимости 10%).

4. Методика экспериментальных исследований

Общая схема работы исследуемых методов состоит из 3 этапов, которые подробно описаны в подразделах 4.1–4.3. Подраздел 4.4 посвящен используемым наборам данных; в последнем подразделе аргументируется выбор мер эффективности.

4.1 Предобработка

Для предварительной обработки входного текста применялась следующая последовательность действий:

1. токенизация: использовалась библиотека NLTK (Natural Language Toolkit)[2] [49];
2. удаление знаков препинания;
3. перевод слов в нижний регистр;
4. удаление стоп-слов: использовались списки стоп-слов из NLTK и Scikit-learn[3] [50];
5. стемминг: использовался стемминг Snowball (Porter2)[4] из библиотеки NLTK.

4.2 Векторизация

Основное отличие исследуемых методов заключалось в способе векторизации текста. Были исследованы следующие методы: BinaryBOW, CountBOW, TermBOW, TF-IDF, BM25, NMF, LDA, WVAvgPool, PV-DM, PV-DBOW, WordClustering, WVClustering.

Метод TermBOW представляет собой модификацию методов CountBOW и TF-IDF, в которых вместо слов рассматриваются термины, найденные с помощью методов CValue, Weirdness, LinkProbability, NovelTopicModel, DomainModel, KeyConceptRelatedness, Voting, PU (см. обзор методов извлечения терминологии [51]). Использовалась реализация библиотеки ATR4S[5] [52].

[2] http://www.nltk.org/
[3] http://scikit-learn.org/
[4] http://snowball.tartarus.org/algorithms/english/stemmer.html
[5] https://github.com/ispras/atr4s

Для реализации методов BinaryBOW, CountBOW, TF-IDF, LDA, NMF, WordClustering, WVClustering, TermBOW использовалась библиотека Scikit-learn.

Для реализации методов WVAvgPool, WVClustering, PV-DM, PV-DBOW использовалась библиотека Gensim[6] [53], предоставляющая методы для тематического моделирования и получения векторных представлений слов (word2vec, doc2vec).

Для WVAvgPool использовалась модель Word2Vec, обученная на текстах английской Википедии (на февраль 2015 года)[7].

Поскольку алгоритм k-means, выбранный в качестве основного метода кластеризации (см. 4.3), основан на Евклидовом расстоянии, которое учитывает длину векторов, и поскольку для кластеризации документов по тематикам их длина не важна [8], все векторы были нормализованы в L2-норме.

4.3 Кластеризация

Были исследованы следующие методы кластеризации: k-means, агломеративная кластеризация, спектральная кластеризация (использовалась реализация библиотеки Scikit-learn). После предварительных экспериментов было решено не проводить исследование алгоритма DBSCAN [54], так как он продемонстрировал слишком высокую чувствительность к выбору параметров (min_samples и eps); кроме того, DBSCAN достаточно много объектов не относит ни к одному из кластеров, помечая их как шум, что затрудняет его сравнение с другими алгоритмами кластеризации.

Количество кластеров k задавалось в качестве параметра для алгоритмов k-means и агломеративной кластеризации. Для каждого исследуемого набора данных фиксировалось множество возможных значений параметра k, которые перебирались в процессе поиска лучшего набора параметров. Остальные параметры алгоритмов кластеризации использовались по умолчанию[8].

В частности, для инициализации центров кластеров в k-means применялся алгоритм k-means++ [55]. Для получения устойчивых результатов совершалось 10 запусков k-means; из 10 полученных кластеризаций выбиралась лучшая (минимизирующая суммарное расстояние всех кластеризуемых объектов до ближайших центров кластеров).

4.4 Наборы данных

Экспериментальное исследование проводилось на наборах данных 20 Newsgroups (20 NG)[9] [56], Krapivin (KR) [57], аннотации из Krapivin (Krapivin-abstracts, KRabs), TREC GEN 2007 (TG2007) [58].

[6] https://radimrehurek.com/gensim/
[7] https://github.com/idio/wiki2vec
[8] Версия Scikit-learn: 0.18.1
[9] http://qwone.com/~jason/20Newsgroups/

Набор данных 20 Newsgroups представляет собой 18846 новостных статей, каждая из которых посвящена одной из 20 тем. Включение этого набора данных в исследование было продиктовано его частым использованием в работах, посвященных кластеризации текстов (например, в [8, 26, 59]).

Набор данных Krapivin состоит из научных статей, посвященных компьютерным наукам (Computer Science). В качестве ожидаемой кластеризации статей были использованы темы верхнего уровня классификации ACM CCS (Computing Classification System)[10], которые проставляются статьям вручную экспертами. Из набора данных Krapivin были взяты статьи, имеющие ровно одну тему верхнего уровня. Таких статей оказалось 1478, количество различных тем: 12.

Помимо основного набора данных Krapivin, также использовался набор данных, состоящий только из аннотаций научных статей. Это было сделано для того, чтобы оценить эффективность работы методов в случае, когда доступны только небольшие части текстов.

Набор данных TREC GEN 2007 состоит из статей, посвященных геномике. Набор данных был составлен для проведения конкурса по извлечению сущностей TREC 2007 Genomics Track[11]. Организаторы использовали корпус научных статей Highwire Press[12], состоящий из 160 000 статей, взятых из 49 журналов, посвященных геномике.

Из этого корпуса было выделено несколько тысяч статей, которые были предоставлены экспертам в данной предметной области для разметки. Разметка заключалась в нахождении в статьях ответов на 36 вопросов, фиксированных организаторами. Положительный ответ на каждый вопрос обозначает наличие в тексте статьи описания некоторой темы. Из этого множества статей были удалены те, для которых были даны ответы на два и более вопроса. Оставшиеся статьи относились в отдельный класс в том случае, если экспертами были даны ответы только на один вопрос. Итоговый набор составляет 2325 статей.

4.5 Меры эффективности

В ходе экспериментального исследования были использованы следующие внешние меры эффективности: Adjusted Mutual Information(AMI), Normalized Mutual Information(NMI), Adjusted Rand Index (ARI), V-measure. Мы использовали несколько мер эффективности, поскольку (1) для данной задачи не существует единой общепринятой меры (см. подраздел 3.8) и (2) это позволяет произвести сравнение с исследованиями, описанными в других работах.

При этом в качестве основной меры эффективности была выбрана AMI по следующим причинам. Во-первых, меры, основанные на Mutual Information и

[10] http://dl.acm.org/ccs/ccs.cfm
[11] http://trec.nist.gov/data/t2007_genomics.html
[12] http://home.highwire.org/

Rand Index являются наиболее популярными. Во-вторых, в мерах AMI и ARI вводится поправка на случайность (adjusted for chance [60]: при сравнении случайных кластеризаций эти меры имеют близкое к нулю значение, в то время как значения NMI могут быть сильно больше 0 при большом количестве кластеров). В-третьих, классы в научных статьях обычно несбалансированны, то есть данные представляют собой набор как больших, так и малых классов, а Романо и др. [60] показали, что в таких случаях AMI является предпочтительной мерой по сравнению с ARI.

Также использовались следующие внутренние меры эффективности: Silhouette Coefficient (Silhouette, SC); Calinski-Harabaz Index (CHI). Помимо их популярности, выбор этих мер обусловлен тем, что они хорошо подходят для оптимизации параметров алгоритма k-means, так как основаны на похожих предположениях [61].

Стоит отметить, что меньшему значению меры Calinski-Harabaz Index соответствует большее значение эффективности. Для всех остальных рассматриваемых мер верно обратное: большее значение меры соответствует большей эффективности.

5. Результаты экспериментальных исследований

В данном разделе описаны результаты экспериментальных исследований методов кластеризации текстов.

5.1. Сравнение методов

В таблице 1 представлены максимальные значения меры эффективности AMI для разных наборов данных. Эти данные были получены путем перебора различных параметров методов векторизации и параметра k (количество кластеров) алгоритма k-means, и выбора максимально значения для каждого метода и для каждого набора данных. Представленные значения можно считать потенциальными максимумами для исследуемых методов.

Табл, 1. Максимальное значение AMI (k-means)
Table 1. Maximum value of AMI (k-means)

	20NG	KR	KRabs	TG2007
BinaryBOW	0.2586	0.2402	0.2041	0.3581
CountBOW	0.2957	0.2453	0.1598	0.4018
TermBOW	0.3123	0.2659	0.1266	0.5038
TF-IDF	0.4911	0.2826	0.2705	0.5051
BM25	0.4261	**0.3069**	**0.2824**	0.5291
NMF	0.4438	0.2642	0.262	0.4882
LDA	0.3391	0.2831	0.2237	0.4155

WVAvgPool	0.141	0.174	0.1608	0.2847
PV-DM	0.5901	0.3014	0.2483	**0.56**
PV-DBOW	**0.6735**	0.2773	0.251	0.5026
WordClustering	0.2188	0.2296	0.2059	0.4193
WVClustering	0.1159	0.0875	0.0613	0.1636

В соответствии с таблицей 1, наибольшие значения AMI имеют PV-DBOW, PV-DM, BM25, TF-IDF и NMF.

Можно видеть, что значение функции эффективности AMI на различных наборах при фиксированных методах сильно отличается. К примеру, для всех исследуемых методов значение AMI на наборе данных 20 Newsgroups выше, чем на Krapivin и Krapivin-abstracts. Возможная причина — большинство тем в 20 Newsgroups семантически далеки друг от друга (политика, спорт, автомобили), в то время как все статьи набора данных krapivin и krapivin-abstracts посвящены компьютерным наукам. Также все методы на наборе данных Krapivin-abstracts работают хуже, чем на Krapivin, однако разница не столь велика, что может быть объяснено тем, в аннотациях пытаются кратко изложить суть статьи и используют для этого специфичные термины.

5.2 Визуализация матрицы ошибок

Для визуализации результатов кластеризации принято использовать матрицу ошибок (confusion matrix) — матрицу, каждая строка которой соответствует распределению объектов классов, присутствующих в разметке набора данных, по кластерам, полученным с помощью используемых методов, а каждый столбец — распределению объектов кластера по классам.

Чем интенсивнее цвет прямоугольника на пересечении, тем большее количество объектов класса, соответствующего строке, были отнесены методом в кластер, соответствующий столбцу.

Визуализация матрицы ошибок в случае, когда метод работает идеально (все объекты одного класса и только они относятся в один кластер), представляет собой квадратную матрицу, у которой в каждой строке и в каждом столбце закрашен только один квадрат.

На рисунке 1 изображена матрица ошибок для метода PV-DBOW на наборе данных 20 Newsgroups. Можно видеть, что метод довольно точно определяет классы: практически для каждой строки и для каждого столбца существует только один темный квадрат. В соответствии с матрицей ошибок, большие части 3 и 4 классов (нумерация с 0) были отнесены в один кластер. Это можно объяснить их темами: 3 класс посвящен устройствам IBM (’comp.sys.ibm.pc.hardware’), а 4 класс — устройствам Mac (’comp.sys.mac.hardware’). Похожая ситуация наблюдается с классами 16 и 20 (’soc.religion.christian’ и ’talk.religion.misc’), а также 17 и 19 (’talk.politics.guns’ и ’talk.politics.misc’).

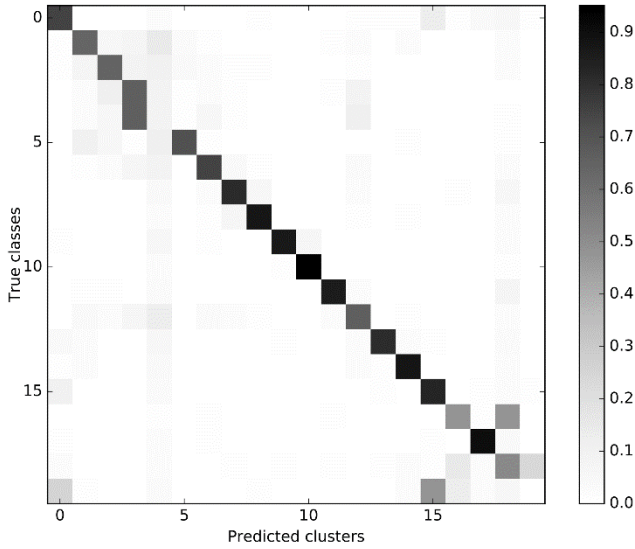


Рис. 1. Матрица ошибок PV-DBOW (k-means) на 20 Newsgroups
Fig. 1. Confusion matrix for PV-DBOW (k-means) on 20 Newsgroups

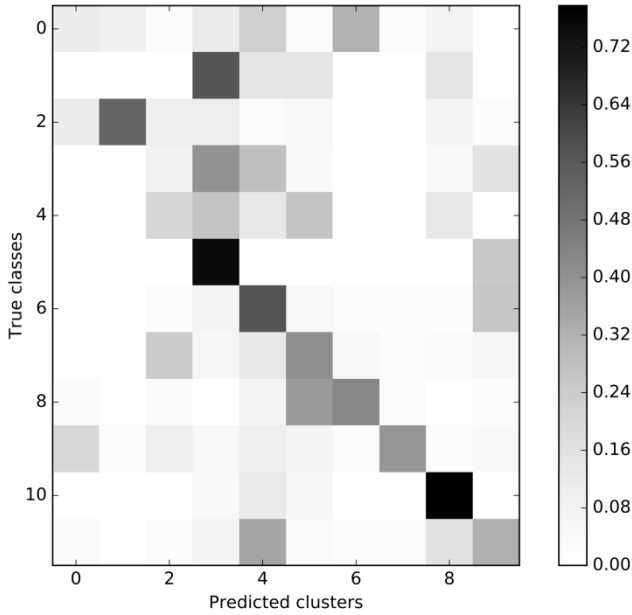


Рис. 2. Матрица ошибок BM25 (k-means) на Krapivin
Fig. 2. Confusion matrix for BM25 (k-means) on Krapivin

На рисунке 2 изображена матрица ошибок для метода PV-DM на наборе данных Krapivin. Несмотря на то, что для этой матрицы отсутствует явная структура, для нее можно выделить некоторые закономерности. Например, можно наблюдать соответствие между классами и кластерами: практически для каждой строки можно выделить столбец, в пересечении с которым содержится темный квадрат. Кластер 3 содержит значительную часть объектов класса 1 (тема верхнего уровня классификации ACM CCS: Social and professional topics) и класса 5 (General and reference); кластеру 5 соответствуют классы 7 (Theory of computation) и 8 (Mathematics of computing).

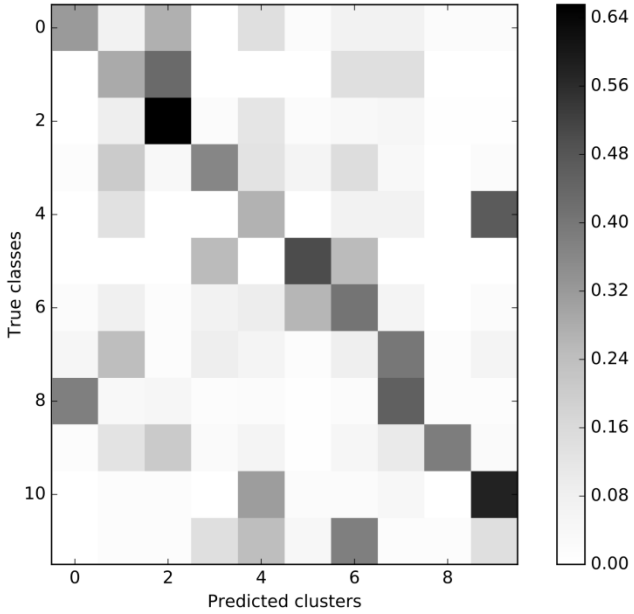


Рис. 3. Матрица ошибок BM25 (k-means) на Krapivin-abstracts
Fig. 3. Fig. 2. Confusion matrix for BM25 (k-means) on Krapivin-abstracts

На рисунке 3 изображена матрица ошибок для метода BM25 на наборе данных Krapivin-abstracts. Аналогично набору данных Krapivin, у матрицы ошибок тяжело выделить ясную структуру.

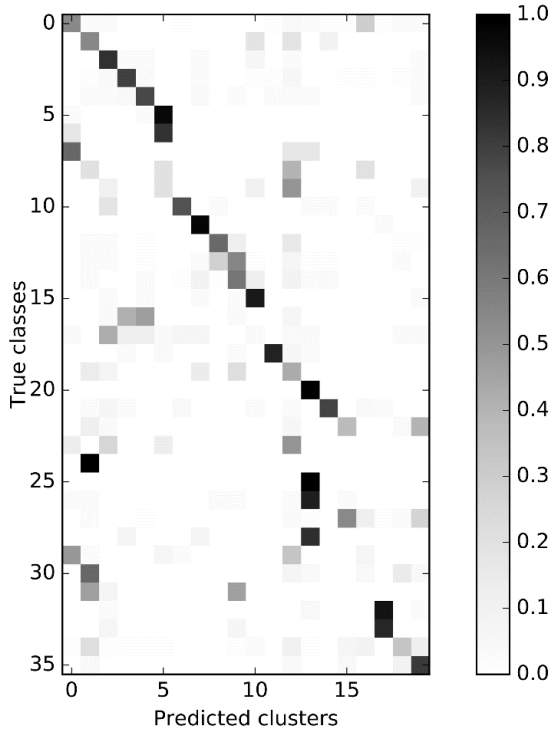


Рис. 4. Матрица ошибок PV-DM (k-means) на TREC GEN 2007
Fig. 4. Confusion matrix for PV-DB (k-means) on TREC GEN 2007

На рисунке 4 изображена матрица ошибок для метода PV-DM на наборе данных TREC GEN 2007. Количество кластеров (20) намного меньше количества классов (36). В связи с этим, в некоторых кластерах содержится большая часть объектов сразу нескольких классов. Значительная часть объектов практически каждого класса была отнесена ровно в 1 кластер. Такое поведение метода может быть объяснено тем, что количество классов слишком большое и некоторые из них содержат тексты, описывающие схожие темы.

5.3 Внутренние меры эффективности

Во многих реальных задачах для исследуемых наборов данных не существует информации о распределении документов по классам, из-за этого возникают трудности с выбором модели и параметров с наибольшей эффективностью.

Одним из подходов для решения этих проблем является оптимизация внутренних мер эффективности (в частности, Silhouette Coefficient и Calinski-Harabaz Index).

Табл. 2. Значение AMI (k-means) при подборе параметров с помощью оптимизации Silhouette
Table 2. AMI values (k-means) when selecting parameters using Silhouette optimization

	20NG	KR	KRabs	TG2007
BinaryBOW	0.0378	0.0343	0.0108	0.0946
CountBOW	0.2114	0.1589	-0.0012	0.0863
TermBOW	0.0692	0.0203	0.0161	0.2878
TF-IDF	0.0451	0.2404	0.0096	0.1565
BM25	0.0765	0.1769	0.1977	0.1241
NMF	0.0217	0.1301	0.006	0.1941
LDA	0.1371	0.2021	0.1561	0.2616
WVAvgPool	0.0706	0.1194	0.1074	0.1657
PV-DM	0.4757	0.2351	0.1774	**0.4716**
PV-DBOW	**0.6551**	**0.2515**	**0.2437**	0.4467
WordClustering	0.0421	0.1333	0.0394	0.1756
WVClustering	0.026	0.0181	0.0415	0.0152

Табл. 3. Значение AMI (k-means) при подборе параметров с помощью оптимизации Calinski-Harabaz Index
Table 3. AMI values (k-means) when selecting parameters using Calinski-Harabaz Index optimization

	20NG	KR	KRabs	TG2007
BinaryBOW	0.2581	0.1687	0.0023	0.3327
CountBOW	0.0355	0.0103	0.0145	0.189
TermBOW	0.2961	0.0723	0.0151	0.02
TF-IDF	0.0264	0.0789	0.0015	0.2267
BM25	0.3197	0.2155	0.1148	0.3886
NMF	0.4365	0.2285	**0.211**	0.39
LDA	0.2195	0.1765	0.1545	0.2651
WVAvgPool	0.141	0.1573	0.1608	0.2671
PV-DM	0.449	**0.2315**	0.1804	**0.4712**
PV-DBOW	**0.5962**	0.2044	0.1987	0.427
WordClustering	0.1991	0.1985	0.1899	0.3971
WVClustering	0.0363	0.0161	0.0496	0.0352

В таблице 2 содержатся значения меры эффективности AMI при подборе параметров с помощью оптимизация Silhouette Coefficient; в таблице 3 — аналогичные значения для Calinski-Harabaz Index.

Исходя из приведенных таблиц можно сделать вывод, что разные методы хорошо оптимизируются разными внутренними мерами эффективности. К примеру, для поиска параметров PVDBOW лучше подходит Silhouette Coefficient, а для BM25 и NMF — Calinski-Harabaz Index.

Для оценки связи внутренних метрик эффективности и внешних была посчитана ранговая корреляция Кендалла[13] [63].

В приложении В находятся таблицы с результатами вычислений ранговой корреляции (21 и 24), таблицы с оптимальными значениями внутренних мер эффективности (таблицы 19 и 22) и таблицы с отношением полученного значения AMI, оптимизированного с помощью внутренних мер эффективности, к максимально возможному значению AMI (таблицы 20 и 23).

Для большинства методов корреляция близка к нулю либо значительно отличается в зависимости от набора данных, однако в некоторых случаях использование внутренних мер эффективности позволяет подобрать параметры метода, при которых значение меры эффективности AMI близко к оптимальному.

5.4 Внешние меры эффективности

Как было отмечено выше, в разделе 4.5, для задачи кластеризации не существует общепринятой внешней меры эффективности: наравне с AMI часто используются Normalized Mutual Information(NMI), Adjusted Rand Index (ARI), Vmeasure и другие.

Для оценки того, существенно ли влияет выбор внешней меры эффективности на определение лучшего метода, также были вычислены значения мер NMI, ARI, V-measure и корреляция Кендалла между ними и AMI, см. приложение А.

На основе проведенных экспериментов можно сделать вывод, что корреляция между AMI и остальными мерами эффективности достаточно высока.

Также можно отметить, что на трех из четырех наборов данных (кроме набора данных Krapivin) все внешние меры эффективности имеют максимальное значение при использовании одних и тех же методов. На наборе данных Krapivin NMI, ARI и V-measure принимают оптимальное значение при использовании метода PV-DM, а NMI — при BM25 (впрочем, разность значений мер для этих методов не превосходит 0.02).

[13] В данной задаче ранговая корреляция предпочтительнее обычной, так как больший интерес представляет относительный порядок методов для двух исследуемых мер; при этом корреляция Кендалла предоставляет более надежную оценку, чем ранговая корреляция Спирмена, особенно в случае небольших размеров выборки [62].

Таким образом можно считать, что выбор внешней меры эффективности не оказывает значительного влияния на определение наиболее эффективного метода.

5.5 Другие методы кластеризации

Помимо k-means также были исследованы агломеративная кластеризация и спектральная кластеризация. В связи с ограниченностью вычислительных ресурсов было принято решение исследовать другие способы кластеризации только для методов, имеющих высокое значение меры AMI для k-means, а именно: PV-DBOW, PV-DM, BM25, TF-IDF и NMF.

Максимальные возможные значения AMI при использовании агломеративной кластеризации содержатся в таблице 4. Аналогичные результаты для спектральной кластеризации содержатся в таблице 5.

Табл. 4. Максимальное значение AMI (агломеративная кластеризация)
Table 4. Maximum values of AMI (agglomerate clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.4281	0.2642	0.2264	0.4481
BM25	0.4999	0.2918	**0.2495**	0.5092
NMF	0.2837	0.2334	0.1918	0.4361
PV-DM	0.5089	**0.3024**	0.2204	**0.5112**
PV-DBOW	**0.5883**	0.2928	0.2173	0.5029

Табл. 5. Максимальное значение AMI (спектральная кластеризация)
Table 5. Maximum values of AMI (spectral clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.4086	0.2455	0.2355	0.4348
BM25	0.4812	0.253	**0.2832**	0.4914
NMF	0.3905	0.2519	0.2323	0.4401
PV-DM	0.5092	**0.2781**	0.2281	**0.5209**
PV-DBOW	**0.6072**	0.2519	0.2316	0.4964

В таблицах 6 и 7 содержатся значения меры эффективности AMI при применении агломеративной кластеризации и при оптимизации мер эффективности Silhouette Coefficient и Calinski-Harabaz Index, соответственно. Аналогичные данные для спектральной кластеризации см. в таблицах 8 и 9. Оптимальные значения внутренних мер эффективности, их коэффициенты корреляции с AMI и отношения значений AMI, полученных с помощью

оптимизации внешних мер, к максимальному значению AMI представлены в таблицах в приложении В.

Табл. 6. Значение AMI (агломеративная кластеризация) при подборе параметров с помощью оптимизации Silhouette Coefficient
Table 6. AMI values (agglomerate clustering) when selecting parameters using Silhouette Coefficient optimization

	20NG	KR	KRabs	TG2007
TF-IDF	0.0385	0.0158	0.0155	0.0911
BM25	0.1516	0.1949	**0.2057**	0.1755
NMF	0.0517	0.0598	0.007	0.2559
PV-DM	0.4126	**0.2751**	0.1481	**0.4638**
PV-DBOW	**0.4836**	0.2353	0.1413	0.4626

Табл. 7. Значение AMI (агломеративная кластеризация) при подборе параметров с помощью оптимизации Calinski-Harabaz Index
Table 7. AMI values (agglomerate clustering) when selecting parameters using Calinski-Harabaz Index optimization

	20NG	KR	KRabs	TG2007
TF-IDF	0.0482	0.0111	0.0129	0.227
BM25	0.4141	**0.266**	**0.1973**	0.3804
NMF	-0.0	0.0003	-0.0005	0.0007
PV-DM	0.3608	0.2076	0.1281	0.4406
PV-DBOW	**0.5256**	0.2226	0.1794	**0.4431**

Табл.8. Значение AMI (спектральная кластеризация) при подборе параметров с помощью оптимизации Silhouette Coefficient
Table 8. AMI values (spectral clustering) when selecting parameters using Silhouette Coefficient optimization

	20NG	KR	KRabs	TG2007
TF-IDF	0.0347	0.05	0.0135	0.0936
BM25	0.0787	0.1704	0.1155	0.2172
NMF	0.0306	0.0814	0.0092	0.2632
PV-DM	0.4176	**0.2411**	**0.1785**	**0.4528**
PV-DBOW	**0.5945**	0.2409	0.1726	0.446

Табл. 9. Значение AMI (спектральная кластеризация) при подборе параметров с помощью оптимизации Calinski-Harabaz Index
Table 9. AMI values (spectral clustering) when selecting parameters using Calinski-Harabaz Index optimization

	20NG	KR	KRabs	TG2007
TF-IDF	0.0247	0.1218	0.0058	0.2122
BM25	0.4383	**0.2303**	**0.2266**	0.388
NMF	-0.0001	0.0017	-0.0014	0.0006
PV-DM	0.2991	0.2225	0.141	**0.4549**
PV-DBOW	**0.5372**	0.2298	0.2041	0.4205

Для каждого набора данных максимальные значения AMI при использовании k-means (табл. 1) выше, чем при использовании и агломеративной кластеризации (табл. 4) и спектральной (табл. 5).

Сравнение значений AMI, полученных с помощью оптимизации внутренних мер эффективности, показывает, что набору данных Krapivin соответствует большее значение AMI при использовании агломеративной кластеризации; для остальных наборов данных предпочтительнее использовать k-means.

Отметим, что агломеративная кластеризация может быть полезной в случае, когда требуется изменять число кластеров и не пересчитывать при этом всю кластеризацию, поскольку агломеративная кластеризация строит дендрограмму для всех объектов и позволяет производить разбиение по разным порогам.

Сравнение значений AMI, вычисленных при оптимизации внутренних мер эффективности, демонстрирует преимущество k-means перед спектральной кластеризацией.

Аналогичные выводы можно сделать при сравнении агломеративной кластеризации и спектральной.

5.6 Время работы

Время работы методов, в случае применения кластеризации k-means, на исследуемых наборах данных описано в таблице 10. Оно было получено путем путем усреднения трех запусков; значения параметров методов соответствовали значениям, при которых максимизируется функция эффективности AMI.

В таблице 11 содержится аналогичная информация для агломеративной кластеризации, в таблице 12 — для спектральной.

Конфигурация вычислительного устройства: Intel Xeon E312xx (8 CPU), 2GHz, 64GB RAM.

Табл. 10. Время работы методов (в секундах) при применении k-means
Table 10. Method running time (in seconds) when using k-means

	20NG	KR	KRabs	TG2007
BinaryBOW	334	130	9	251
CountBOW	342	169	14	273
TermBOW	130	171	13	327
TF-IDF	340	132	11	501
BM25	264	225	5	265
NMF	4443	820	561	1384
LDA	225	169	13	303
WVAvgPool	202	189	8	390
PV-DM	291	1050	24	2038
PV-DBOW	468	668	32	1666
WordClustering	882	808	69	775
WVClustering	566	355	27	916

Табл. 11. Время работы методов (в секундах) при применении агломеративной кластеризации
Table 11. Method running time (in seconds) when using agglomerate clustering

	20NG	KR	KRabs	TG2007
TF-IDF	1094	130	9	261
BM25	1296	266	9	260
NMF	5212	874	594	1373
PV-DM	357	482	22	1134
PV-DBOW	645	687	19	1527

Таблица 12. Время работы методов (в секундах) при применении спектральной кластеризации
Table 12. Method running time (in seconds) when using spectral clustering

	20NG	KR	KRabs	TG2007
TF-IDF	163	121	4	239
BM25	177	123	4	307
NMF	2821	840	535	1291
PV-DM	419	904	20	1953
PV-DBOW	521	382	27	848

6. Заключение

В данной работе рассмотрены и экспериментально исследованы методы кластеризации текстовых документов, в том числе, научных статей. Каждый метод состоял из трех последовательных этапов: предварительная обработка текста (см. раздел 4.1); векторизация предобработанного текста (см. раздел 4.2); кластеризация векторов (см. раздел 4.3).

Экспериментальное исследование показало, что лучшим методом (при условии оптимизации параметров с помощью внутренней меры эффективности) является k-means с векторизацией Paragraph Vectors для всех наборов данных; кроме Krapivin, для которого лучше оказалась агломеративная кластеризация. Стоит отметить, что эффективность разных модификаций метода Paragraph Vectors (DBOW и DM) сильно зависит от набора данных: для новостных текстов и аннотаций научных статей DBOW значительно превосходит DM, в то время как на двух остальных наборах данных DM несколько лучше, чем DBOW. При этом кластеризация аннотаций научных статей оказалась менее эффективной, чем кластеризация полных статей, которая, в свою очередь, показала значительно меньшую эффективность по сравнению с кластеризацией научных статей.

Также в данной работе были исследованы меры эффективности кластеризации: как внешние (Adjusted Mutual Information, Normalized Mutual Information, Adjusted Rand Index, V-measure), так и внутренние (Silhouette Coefficient, Calinski-Harabaz Index). В частности, была посчитана корреляция между внешними мерами эффективности; полученные значения позволяют сделать вывод об относительно высокой взаимозаменяемости этих мер. Внутренние меры показали достаточно высокую эффективность (хотя и не самую высокую стабильность) для задачи оптимизации параметров методов: результаты методов, оптимизированных с помощью меры Silhouette параметры, составили от 82% до 97% от лучших результатов. Кроме того, выяснилось, что для оптимизации разных методов лучше подходят разные внутренние меры эффективности: так, для BM25 выше корреляция у меры Calinski-Harabaz, в то время как для Paragraph Vectors – Silhouette.

Наиболее перспективными направлениями дальнейшей работы представляется улучшение эффективности кластеризации научных статей за счет использования дополнительной информации, такой как граф цитирования и мета-данные статей (авторы, год и место издания), а также проведение экспериментальных исследований на других наборах данных.

Список литературы

[1]. Liu Xiaoyong, Croft W Bruce. Cluster-based retrieval using language models. Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval. ACM. 2004, pp. 186–193.

[2]. Sasaki Minoru, Shinnou Hiroyuki. Spam detection using text clustering. 2005 International Conference on Cyberworlds (CW'05). IEEE. 2005, pp. 316-319.

[3]. Sergio Decherchi, Simone Tacconi, Judith Redi et al. Text clustering for digital forensics analysis. Computational Intelligence in Security for Information Systems. Springer, 2009, pp. 29–36.
[4]. E Dransfield, G Morrot, J-F Martin et al. The application of a text clustering statisticalanalysis to aid the interpretation of focus group interviews.Food Quality and Preference. 2004. T. 15, № 5, pp. 477–488.
[5]. Bader Aljaber, Nicola Stokes, James Bailey et al. Document clustering of scientific texts using citation contexts. Information Retrieval. 2010. T. 13, № 2, pp. 101–131.
[6]. Marchionini Gary. Exploratory search: from finding to understanding. Communications of the ACM. 2006. T. 49, № 4, pp. 41–46.
[7]. Andrews Nicholas O, Fox Edward A. Recent developments in document clustering: Tech. Rep.: Technical report, Computer Science, Virginia Tech, 2007.
[8]. Huang Anna. Similarity measures for text document clustering. Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand. 2008, pp. 49–56.
[9]. Sathiyakumari K, Manimekalai G, Preamsudha V. A survey on various approaches in document clustering.
[10]. Popat Shraddha K, Emmanuel M. Review and comparative study of clustering techniques.
[11]. Anastasiu David C, Tagarelli Andrea, Karypis George. Document Clustering: The Next Frontier. 2013.
[12]. Aggarwal Charu C, Reddy Chandan K. Data clustering: algorithms and applications. CRC Press, 2013.
[13]. Aggarwal Charu C, Zhai Cheng Xiang. Mining text data. Springer Science & Business Media, 2012.
[14]. Saiyad Nagma Y, Prajapati Harshadkumar B, Dabhi Vipul K. A Survey of Document Clustering using Semantic Approach.
[15]. Salton Gerard, Buckley Christopher. Termweighting approaches in automatic text retrieval. Information processing & management. 1988. T. 24, № 5, pp 513–523.
[16]. Whissell John S, Clarke Charles LA. Improving document clustering using Okapi BM25 feature weighting. Information retrieval. 2011. T. 14, № 5, pp. 466–487.
[17]. Голомазов Д. Д. Методы и средства управления научной информацией с использованием онтологий. Диссертация кандидата физико-математических наук. Москва. 2012.
[18]. Pinto David, Jim´enez-Salazar H´ector, Rosso Paolo. Clustering abstracts of scientific texts using the transition point technique. International Conference on Intelligent Text Processing and Computational Linguistics. Springer. 2006, pp. 536–546.
[19]. Scott Deerwester, Susan T Dumais, George W Furnas et al. Indexing by latent semantic analysis. Journal of the American society for information science. 1990. T. 41, № 6, pp. 391.
[20]. Xu Wei, Liu Xin, Gong Yihong. Document clustering based on non-negative matrix factorization. Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. ACM. 2003, pp. 267–273.
[21]. Hofmann Thomas. Probabilistic latent semantic indexing. Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. ACM. 1999, pp. 50–57.
[22]. Blei David M, Ng Andrew Y, Jordan Michael I. Latent dirichlet allocation. Journal of machine Learning research. 2003. T. 3, № Jan., pp. 993–1022.

[23]. Tomas Mikolov, Kai Chen, Greg Corrado et al. Efficient estimation of word representationsin vector space. arXiv preprint, arXiv:1301.3781. 2013.
[24]. Chao Xing, Dong Wang, Xuewei Zhang et al. Document classification with distributions ofword vectors. Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific. IEEE. 2014, pp. 1–5.
[25]. Le Quoc V, Mikolov Tomas. Distributed Representations of Sentences and Documents. ICML. T. 14. 2014, pp. 1188– 1196.
[26]. Slonim Noam, Tishby Naftali. Document clustering using word clusters via the information bottleneck method. Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval. ACM. 2000, pp. 208–215.
[27]. Cao Qimin, Guo Qiao, Wang Yongliang et al. Text clustering using VSM with feature clusters. Neural Computing and Applications. 2015. T. 26, № 4, pp. 995–1003.
[28]. Hotho Andreas, Maedche Alexander, Staab Steffen. Ontology-based text document clustering.
[29]. Choudhary Bhoopesh, Bhattacharyya Pushpak. Text clustering using semantics. Proceedings of the 11th International World Wide Web Conference. 2002, pp. 1–4.
[30]. Jayarajan Dinakar, Deodhare Dipti, Ravindran B. Lexical Chains as Document Features. Third International Joint Conference on Natural Language Processing. Citeseer. 2008, pp. 111.
[31]. Enrique Amigo´, Julio Gonzalo, Javier Artiles et al. A comparison of extrinsic clustering evaluationmetrics based on formal constraints. Information retrieval. 2009. T. 12, № 4, pp. 461–486.
[32]. Zhao Ying, Karypis George, Du Ding-Zhu. Criterion functions for document clustering: Tech. Rep.: Technical Report, 2005.
[33]. Meil˘a Marina. Comparing clusterings by the variation of information. Learning theory and kernel machines. Springer, 2003, pp. 173–187.
[34]. Hubert Lawrence, Arabie Phipps. Comparing partitions. Journal of classification. 1985. T. 2, № 1, pp. 193–218.
[35]. Bakus J, Hussin MF, Kamel M. A SOM-based document clustering using phrases. Neural Information Processing, 2002. ICONIP'02. Proceedings of the 9th International Conference on. IEEE. T. 5. 2002, pp. 2212–2216.
[36]. Vinh Nguyen Xuan, Epps Julien, Bailey James. Information theoretic measures for clusterings comparison: is a correction for chance necessary?. Proceedings of the 26th Annual International Conference on Machine Learning. ACM. 2009, pp. 1073–1080.
[37]. Strehl Alexander, Ghosh Joydeep. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. Journal of machine learning research. 2002. T. 3, № Dec., pp. 583–617.
[38]. Rosenberg Andrew, Hirschberg Julia. VMeasure: A Conditional Entropy-Based External Cluster Evaluation Measure. EMNLP-CoNLL. T. 7. 2007, pp. 410–420.
[39]. Bagga Amit, Baldwin Breck. Entity-based cross-document coreferencing using the vector space model. Proceedings of the 17th international conference on Computational linguistics-Volume 1. 1998, pp. 79–85.
[40]. Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza et al. An extensive comparative study of cluster validity indices. Pattern Recognition. 2013. T. 46, № 1, pp. 243–256.
[41]. Yanchi Liu, Zhongmou Li, Hui Xiong et al. Understanding of internal clustering validation measures. 2010 IEEE International Conference on Data Mining. IEEE. 2010, pp. 911–916.

[42]. Er´endira Rend´on, Itzel Abundez, Alejandra Arizmendi et al. Internal versus external cluster validation indexes.. International Journal of computers and communications. 2011. Т. 5, № 1, pp. 27–34.
[43]. Rousseeuw Peter J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics. 1987. Т. 20, pp. 53–65.
[44]. Davies David L, Bouldin Donald W. A cluster separation measure. IEEE transactions on pattern analysis and machine intelligence. 1979. № 2, pp. 224–227.
[45]. Calin´ski Tadeusz, Harabasz Jerzy. A dendrite method for cluster analysis. Communications in Statistics-theory and Methods. 1974. Т. 3, № 1, pp. 1–27.
[46]. Bezdek James C, Pal Nikhil R. Some new indexes of cluster validity. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics). 1998. Т. 28, № 3, pp. 301–315.
[47]. Ibai Gurrutxaga, In˜aki Albisua, Olatz Arbelaitz et al. SEP/COP: An efficient method to find the bestpartition in hierarchical clustering based on a new cluster validity index.Pattern Recognition. 2010. Т. 43, № 10, pp. 3364–3373.
[48]. Halkidi Maria, Vazirgiannis Michalis. Clustering validity assessment: Finding the optimal partitioning of a data set. Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on. IEEE. 2001, pp. 187–194.
[49]. Bird Steven. NLTK: the natural language toolkit. Proceedings of the COLING/ACL on Interactive presentation sessions. Association for Computational Linguistics. 2006, pp. 69–72.
[50]. Scikit-learn: Machine Learning in Python. F. Pedregosa, G. Varoquaux, A. Gramfort [и др.]. Journal of Machine Learning Research. 2011. Т. 12, pp. 2825–2830.
[51]. Astrakhantsev N.A., Fedorenko D.G., Turdakov D.Yu. Methods for automatic term recognition in domain-specific text collections: A survey. Programming and Computer Software. 2015. Т. 41, № 6, pp. 336–349.
[52]. Astrakhantsev Nikita. ATR4S: Toolkit with State-of-the-art Automatic Terms Recognition Methods in Scala. arXiv preprint, arXiv:1611.07804. 2016.
[53]. Reh˚u˘rek R., Sojka P.˘ Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. Valletta, Malta: ELRA, 2010, pp. 45–50.
[54]. Martin Ester, Hans-Peter Kriegel, J¨org Sander Er´endira Rend´on, Itzel Abundez, Alejandra Arizmendi et al. A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd. Т. 96. 1996, pp. 226–231.
[55]. Arthur David, Vassilvitskii Sergei. kmeans++: The advantages of careful seeding. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics. 2007, pp. 1027–1035.
[56]. Lang Ken. Newsweeder: Learning to filter netnews. Proceedings of the 12th international conference on machine learning. 1995, pp. 331–339.
[57]. Krapivin M., Autaeu A., Marchese M. Large dataset for keyphrases extraction. 2009. URL: http://eprints.biblio.unitn.it/1671/1/disi09055krapivin-autayeu-marchese.pdf.
[58]. William Hersh, Aaron Cohen, Lynn Ruslen et al. TREC 2007 Genomics Track Overview. 2007.
[59]. Xie Pengtao, Xing Eric P. Integrating document clustering and topic modeling. arXiv preprint, arXiv:1309.6874. 2013.
[60]. Simone Romano, Nguyen Xuan Vinh, James Bailey et al. Adjusting for Chance Clustering Comparison Measures. arXiv preprint, arXiv:1512.01286. 2015.

[61]. Van Craenendonck Toon, Blockeel Hendrik. Using internal validity measures to compare clustering algorithms. AutoML Workshop at ICML 2015, pp. 1–8.
[62]. Field Andy. Discovering statistics using IBM SPSS statistics. Sage, 2013.
[63]. Kendall Maurice G. A new measure of rank correlation. Biometrika. 1938. Т. 30, № ½, pp. 81–93.

Приложение А. Внешние меры

Табл. 13. Максимальное значение NMI
Table 13. Maximum value of NMI

	20NG	KR	KRabs	TG2007
BinaryBOW	0.2652	0.2608	0.2246	0.4288
CountBOW	0.3009	0.2833	0.1964	0.4885
TermBOW	0.327	0.2978	0.1504	0.5418
TF-IDF	0.5136	0.304	0.2963	0.5611
BM25	0.4411	0.3268	**0.3058**	0.5829
NMF	0.4631	0.2878	0.2751	0.5244
LDA	0.3531	0.3162	0.2484	0.4755
WVAvgPool	0.1509	0.1975	0.1922	0.3618
PV-DM	0.5951	**0.338**	0.2701	**0.5938**
PV-DBOW	**0.6816**	0.3099	0.2758	0.5779
WordClustering	0.2274	0.2519	0.2301	0.4742
WVClustering	0.1218	0.1145	0.0918	0.2357

Табл. 14. Максимальное значение ARI
Table 14. Maximum value of ARI

	20NG	KR	KRabs	TG2007
BinaryBOW	0.1318	0.1687	0.1347	0.1701
CountBOW	0.1126	0.1655	0.0921	0.2358
TermBOW	0.1487	0.1868	0.0785	0.3209
TF-IDF	0.292	0.1957	0.2148	0.3043
BM25	0.1707	0.2113	**0.2295**	0.3285
NMF	0.1955	0.1673	0.2063	0.295
LDA	0.1878	0.2209	0.1738	0.2698
WVAvgPool	0.0551	0.1029	0.0762	0.143
PV-DM	0.4572	**0.2211**	0.2098	**0.3418**
PV-DBOW	**0.5677**	0.1993	0.1727	0.2829
WordClustering	0.0757	0.1464	0.1255	0.2183
WVClustering	0.0309	0.0462	0.0457	0.0797

Табл. 15. Максимальное значение V-measure
Table 15. Maximum value of V-measure

	20NG	KR	KRabs	TG2007
BinaryBOW	0.2651	0.2606	0.2244	0.4266
CountBOW	0.3009	0.2817	0.1954	0.4862
TermBOW	0.3268	0.2971	0.1494	0.5416
TF-IDF	0.5132	0.3038	0.2925	0.5593
BM25	0.441	0.3266	**0.3052**	0.5818
NMF	0.4594	0.2876	0.2751	0.5242
LDA	0.3503	0.3145	0.2472	0.4737
WVAvgPool	0.1507	0.1963	0.1917	0.3599
PV-DM	0.5951	**0.3371**	0.2698	**0.5921**
PV-DBOW	**0.6804**	0.3077	0.2739	0.5745
WordClustering	0.2272	0.2513	0.2297	0.4733
WVClustering	0.1217	0.1138	0.0842	0.2353

Табл. 16. Корреляция NMI и AMI
Table 16. Correlation between NMI and AMI

	20NG	KR	KRabs	TG2007
BinaryBOW	0.8732	0.8936	0.8435	0.8322
CountBOW	0.8146	0.885	0.8795	0.8667
TermBOW	0.8504	0.9248	0.8553	0.8971
TF-IDF	0.873	0.8965	0.8802	0.8675
BM25	0.8543	0.8159	0.8361	0.8414
NMF	0.9888	0.9839	0.9831	0.9683
LDA	0.9272	0.8805	0.8925	0.7455
WVAvgPool	0.9556	0.6429	1.0	0.3778
PV-DM	0.6958	0.8092	0.807	0.7053
PV-DBOW	0.7591	0.7557	0.7281	0.8148
WordClustering	0.9171	0.8467	0.9009	0.8481
WVClustering	0.6904	0.8701	0.7316	0.8282

Табл. 17. Корреляция ARI и AMI
Table 17. Correlation between ARI and AMI

	20NG	KR	KRabs	TG2007
BinaryBOW	0.8044	0.705	0.6674	0.6839

CountBOW	0.7926	0.82	0.7872	0.6469
TermBOW	0.6179	0.8523	0.6653	0.7201
TF-IDF	0.804	0.7125	0.7862	0.7533
BM25	0.6993	0.5223	0.6292	0.6537
NMF	0.9632	0.9396	0.8919	0.9307
LDA	0.8447	0.7526	0.7281	0.4306
WVAvgPool	0.9556	0.1429	0.5714	0.4667
PV-DM	0.8454	0.6851	0.5785	0.5199
PV-DBOW	0.8409	0.5855	0.6952	0.8459
WordClustering	0.704	0.6146	0.6279	0.603
WVClustering	0.6919	0.5548	0.511	0.6823

Табл. 18. Корреляция V-measure и AMI
Table 18. Correlation between V-measure and AMI

	20NG	KR	KRabs	TG2007
BinaryBOW	0.9163	0.903	0.885	0.8432
CountBOW	0.8607	0.8959	0.9296	0.8707
TermBOW	0.8792	0.9322	0.8769	0.9196
TF-IDF	0.9114	0.907	0.9222	0.8864
BM25	0.8711	0.8312	0.8498	0.8695
NMF	0.9902	0.9843	0.9833	0.9714
LDA	0.9336	0.8904	0.8986	0.7898
WVAvgPool	1.0	0.7143	1.0	0.3778
PV-DM	0.7241	0.8232	0.8224	0.7255
PV-DBOW	0.7922	0.7702	0.7553	0.823
WordClustering	0.9267	0.8652	0.9079	0.8724
WVClustering	0.8267	0.867	0.8091	0.8192

Приложение B. Внутренние меры

Табл. 19. Максимальное значение Silhouette
Table 19. Maximum value of Silhouette

	20NG	KR	KRabs	TG2007
BinaryBOW	0.0558	0.0358	0.0294	0.0906
CountBOW	-0.2097	-0.0487	0.0262	0.0429
TermBOW	0.7047	0.4581	0.5768	0.3947

TF-IDF	0.1663	0.0432	0.2043	0.1179
BM25	0.0042	0.0305	0.0124	0.0609
NMF	0.1116	0.3627	0.1915	0.4467
LDA	0.3224	0.5262	0.5115	0.5448
WVAvgPool	0.0575	0.086	0.0496	0.1133
PV-DM	0.0287	0.0301	0.0309	0.0612
PV-DBOW	0.0209	0.0194	0.0241	0.0667
WordClustering	0.179	0.187	0.1601	0.1893
WVClustering	0.9943	0.9139	0.9953	0.9953

Табл. 20. Silhouette: доля от лучшего AMI
Table 20. Silhouette: share from the best AMI

	20NG	KR	KRabs	TG2007
BinaryBOW	0.1462	0.1428	0.0529	0.2642
CountBOW	0.7149	0.6478	0	0.2148
TermBOW	0.2216	0.0763	0.1272	0.5713
TF-IDF	0.0918	0.8507	0.0355	0.3098
BM25	0.1795	0.5764	0.7001	0.2345
NMF	0.0489	0.4924	0.0229	0.3976
LDA	0.4043	0.7139	0.6978	0.6296
WVAvgPool	0.5007	0.6862	0.6679	0.5820
PV-DM	0.8061	0.7800	0.7145	0.8421
PV-DBOW	0.9727	0.9070	0.9709	0.8888
WordClustering	0.1924	0.5806	0.1914	0.4188
WVClustering	0.2243	0.2069	0.6770	0.0929

Табл, 21. Корреляция Silhouette и AMI
Table 21. Correlation between Silhouette and AMI

	20NG	KR	KRabs	TG2007
BinaryBOW	-0.5274	0.1205	-0.0203	-0.3558
CountBOW	**0.5133**	0.4969	0.1228	-0.2095
TermBOW	-0.5763	-0.1145	-0.2712	-0.0788
TF-IDF	-0.048	0.0594	0.0475	-0.0838
BM25	-0.4191	-0.0713	-0.1088	-0.1785
NMF	-0.3425	0.0162	-0.4528	0.0746
LDA	-0.0394	0.3323	**0.1509**	-0.047

WVAvgPool	-0.2889	0.0	-0.2857	-0.1556
PV-DM	0.1476	0.3013	0.0118	0.191
PV-DBOW	-0.1241	**0.5092**	0.0961	**0.3275**
WordClustering	-0.2188	-0.4654	-0.3628	-0.398
WVClustering	-0.3363	-0.1768	-0.1346	-0.4956

Табл. 22. Минимальное значение CHI
Table 22. Minimal values of CHI

	20NG	KR	KRabs	TG2007
BinaryBOW	47.07	4.75	1.79	7.13
CountBOW	0.13	0.79	1.38	13.86
TermBOW	0.23	3.08	5.39	4.8
TF-IDF	0.07	0.76	0.67	4.74
BM25	23.28	4.3	1.92	5.28
NMF	43.9	10.96	6.03	16.05
LDA	733.99	173.67	154.37	162.48
WVAvgPool	47.1	32.21	39.87	27.14
PV-DM	37.66	7.49	10.76	7.67
PV-DBOW	41.41	5.63	9.93	6.52
WordClustering	122.93	24.97	9.79	26.79
WVClustering	3.16	4.08	11.29	1.0

Табл, 23. CHI: доля от лучшего AMI
Table 23. CHI: share from the best AMI

	20NG	KR	KRabs	TG2007
BinaryBOW	0.9981	0.7023	0.0113	0.9291
CountBOW	0.1201	0.0420	0.0907	0.4704
TermBOW	0.9481	0.2719	0.1193	0.0397
TF-IDF	0.0538	0.2792	0.0055	0.4488
BM25	0.7503	0.7022	0.4065	0.7345
NMF	0.9836	0.8649	0.8053	0.7989
LDA	0.6473	0.6235	0.6907	0.6380
WVAvgPool	1.0000	0.9040	1.0000	0.9382
PV-DM	0.7609	0.7681	0.7265	0.8414
PV-DBOW	0.8852	0.7371	0.7916	0.8496
WordClustering	0.9100	0.8645	0.9223	0.9471
WVClustering	0.3132	0.1840	0.8091	0.2152

Табл. 24. Корреляция CHI и AMI
Table 24. Correlation between CHI and AMI

	20NG	KR	KRabs	TG2007
BinaryBOW	0.4554	0.1761	-0.2598	0.7134
CountBOW	-0.3365	-0.2167	-0.421	0.4362
TermBOW	0.5576	0.5057	0.4136	0.4147
TF-IDF	0.2611	-0.1347	-0.2672	0.394
BM25	0.3607	0.1787	0.0412	0.3957
NMF	0.674	0.6169	0.6563	0.4737
LDA	0.0706	-0.1319	-0.1211	0.1978
WVAvgPool	0.3333	0.0714	0.5	0.3778
PV-DM	-0.1389	0.182	0.0952	0.2801
PV-DBOW	0.2415	-0.1614	0.143	0.1151
WordClustering	0.3194	0.5193	0.5287	0.6116
WVClustering	0.3388	-0.0908	0.1729	0.2168

Табл. 25. Максимальное значение Silhouette (агломеративная кластеризация)
Table 25. Maximum values of Silhouette (agglomerate clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.1527	0.0398	0.2064	0.103
BM25	-0.0043	0.0244	0.0067	0.0832
NMF	0.1168	0.3687	0.2088	0.3005
PV-DM	0.0214	0.0219	0.0204	0.0531
PV-DBOW	0.0105	0.0163	0.0153	0.0581

Табл. 26. Silhouette: доля от лучшего AMI (агломеративная кластеризация)
Table 26. CHI: share from the best AMI (agglomerate clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.0899	0.0598	0.0685	0.2033
BM25	0.3033	0.6679	**0.8244**	0.3447
NMF	0.1822	0.2562	0.0365	0.5868
PV-DM	0.8108	**0.9097**	0.6720	0.9073
PV-DBOW	**0.8220**	0.8036	0.6503	**0.9199**

Табл. 27. Корреляция Silhouette и AMI (агломеративная кластеризация)
Table 27. Correlation between Silhouette and AMI (agglomerate clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.012	0.0991	0.0595	-0.0959
BM25	-0.2995	0.1244	-0.1574	-0.1661
NMF	-0.3646	0.0037	-0.4792	0.0569
PV-DM	**0.0373**	0.3053	0.0781	0.0347
PV-DBOW	0.0331	**0.343**	**0.2689**	**0.3275**

Табл. 28. Минимальное значение CHI (агломеративная кластеризация)
Table 28. Minimal values of CHI (agglomerate clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.0837	0.7698	1.1986	5.6026
BM25	24.5886	5.4675	2.9236	6.4625
NMF	1.0	0.7676	1.0	1.0
PV-DM	27.4417	6.6093	7.5673	7.3344
PV-DBOW	32.5286	5.1623	7.7801	6.3808

Табл. 29. CHI: доля от лучшего AMI (агломеративная кластеризация)
Table 29. CHI: share from the best AMI (agglomerate clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.1126	0.0420	0.0570	0.5066
BM25	0.8284	**0.9116**	0.7908	0.7471
NMF	0	0.0013	0	0.0016
PV-DM	0.7090	0.6865	0.5812	0.8619
PV-DBOW	**0.8934**	0.7602	**0.8256**	**0.8811**

Табл. 30. Корреляция CHI и AMI (агломеративная кластеризация)
Table 30. Correlation between CHI and AMI (agglomerate clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.3629	-0.0076	-0.088	**0.4156**
BM25	**0.4256**	**0.1569**	**0.2492**	0.388
NMF	-0.4744	-0.5008	-0.5042	-0.503
PV-DM	-0.1619	0.0978	-0.1447	0.2549
PV-DBOW	0.3022	-0.0272	-0.1425	0.0557

Табл. 31. Максимальное значение Silhouette (спектральная кластеризация)
Table 31. Maximum values of Silhouette (spectral clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.1712	0.0405	0.2113	0.1061
BM25	0.0067	0.0297	0.0111	0.0504
NMF	0.1651	0.3372	0.1822	0.3306
PV-DM	0.0234	0.0265	0.0308	0.0552
PV-DBOW	0.018	0.0084	0.0221	0.0643

Табл. 32. Silhouette: доля от лучшего AMI (спектральная кластеризация)
Table 32. Silhouette: share from the best AMI (spectral clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.0849	0.2037	0.0573	0.2153
BM25	0.1635	0.6735	0.4078	0.4420
NMF	0.0784	0.3231	0.0396	0.5980
PV-DM	0.8201	0.8670	**0.7826**	0.8693
PV-DBOW	**0.9791**	**0.9563**	0.7453	**0.8985**

Табл. 33. Корреляция Silhouette и AMI (спектральная кластеризация)
Table 33. Correlation between Silhouette and AMI (spectral clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.0163	0.0857	0.0577	-0.052
BM25	-0.494	0.0711	-0.4639	-0.0254
NMF	-0.4847	-0.1423	-0.5149	-0.0264
PV-DM	**0.3644**	**0.2689**	**0.2092**	0.1384
PV-DBOW	-0.0532	0.1237	-0.0496	**0.3322**

Табл. 34. Минимальное значение CHI (спектральная кластеризация)
Table 34. Minimal values of CHI (spectral clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.0887	1.4317	0.4466	5.9291
BM25	29.3736	5.8227	3.2461	6.6132
NMF	1.0	0.2563	1.0	1.0
PV-DM	32.3986	7.3227	10.2393	7.943
PV-DBOW	39.4728	5.6117	9.7885	6.8773

Табл. 35. CHI: доля от лучшего AMI (спектральная кластеризация)
Table 35. CHI: share from the best AMI (spectral clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	0.0605	0.4961	0.0246	0.4880
BM25	**0.9108**	0.9103	0.8001	0.7896
NMF	0	0.0067	0	0.0014
PV-DM	0.5874	0.8001	0.6181	**0.8733**
PV-DBOW	0.8847	**0.9123**	**0.8813**	0.8471

Табл. 36. Корреляция CHI и AMI (спектральная кластеризация)
Table 36. Correlation between CHI and AMI (spectral clustering)

	20NG	KR	KRabs	TG2007
TF-IDF	**0.3472**	-0.0455	-0.0785	0.4048
BM25	0.3287	0.1476	**0.3762**	**0.3665**
NMF	-0.4579	-0.4662	-0.4673	-0.4908
PV-DM	-0.4174	**0.2206**	-0.1289	0.2824
PV-DBOW	0.1277	-0.0259	0.1781	0.0714

A survey and an experimental comparison of methods for text clustering: application to scientific articles

[1,2] *P.A. Parhomenko <parhomenko @ispras.ru>*
[1,3] *A.A. Grigorev <agrigorev@ispras.ru>*
[1] *N.A. Astrakhantsev <astrakhantsev@ispras.ru>*
[1] *Institute for System Programming of the RAS,*
25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[3] *National Research University Higher School of Economics (HSE)*
20 Myasnitskaya Ulitsa, Moscow, 101000, Russia

Abstract. Text documents clustering is used in many applications such as information retrieval, exploratory search, spam detection. This problem is the subject of many scientific papers, but the specificity of scientific articles in regards to the clustering efficiency remains to be studied insufficiently; in particular, if all documents belong to the same domain or if full texts of articles are unavailable. This paper presents an overview and an experimental comparison of text clustering methods in application to scientific articles. We study methods based on bag of words, terminology extraction, topic modeling, word embedding and document embedding obtained by artificial neural networks (word2vec, paragraph2vec).

Keywords: text documents clustering; bag of words; terminology extraction; topic modeling; word and document embedding; artificial neural networks

DOI: 10.15514/ISPRAS-2017-29(2)-6

For citation: Parhomenko P.A., Grigorev A.A., Astrakhantsev N.A. A survey and an experimental comparison of methods for text clustering: application to scientific articles. Trudy ISP RAN/Proc. ISP RAS, 2017, vol. 29, issue 2, pp. 161-200 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-6

References

[1]. Liu Xiaoyong, Croft W Bruce. Cluster-based retrieval using language models. Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval. ACM. 2004, pp. 186–193.
[2]. Sasaki Minoru, Shinnou Hiroyuki. Spam detection using text clustering. 2005 International Conference on Cyberworlds (CW'05). IEEE. 2005, pp. 316-319.
[3]. Text clustering for digital forensics analysis. Sergio Decherchi, Simone Tacconi, Judith Redi [и др.]. Computational Intelligence in Security for Information Systems. Springer, 2009, pp. 29–36.
[4]. The application of a text clustering statisticalanalysis to aid the interpretation of focus group interviews. E Dransfield, G Morrot, J-F Martin [и др.]. Food Quality and Preference. 2004. Т. 15, № 5, pp. 477–488.
[5]. Document clustering of scientific texts using citation contexts. Bader Aljaber, Nicola Stokes, James Bailey [и др.]. Information Retrieval. 2010. Т. 13, № 2, pp. 101–131.
[6]. Marchionini Gary. Exploratory search: from finding to understanding. Communications of the ACM. 2006. Т. 49, № 4, pp. 41–46.
[7]. Andrews Nicholas O, Fox Edward A. Recent developments in document clustering: Tech. Rep.: Technical report, Computer Science, Virginia Tech, 2007.
[8]. Huang Anna. Similarity measures for text document clustering. Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand. 2008, pp. 49–56.
[9]. Sathiyakumari K, Manimekalai G, Preamsudha V. A survey on various approaches in document clustering.
[10]. Popat Shraddha K, Emmanuel M. Review and comparative study of clustering techniques.
[11]. Anastasiu David C, Tagarelli Andrea, Karypis George. Document Clustering: The Next Frontier. 2013.
[12]. Aggarwal Charu C, Reddy Chandan K. Data clustering: algorithms and applications. CRC Press, 2013.
[13]. Aggarwal Charu C, Zhai Cheng Xiang. Mining text data. Springer Science & Business Media, 2012.
[14]. Saiyad Nagma Y, Prajapati Harshadkumar B, Dabhi Vipul K. A Survey of Document Clustering using Semantic Approach.
[15]. Salton Gerard, Buckley Christopher. Termweighting approaches in automatic text retrieval. Information processing & management. 1988. Т. 24, № 5, pp 513–523.
[16]. Whissell John S, Clarke Charles LA. Improving document clustering using Okapi BM25 feature weighting. Information retrieval. 2011. Т. 14, № 5, pp. 466–487.

[17]. Golomazov D.D. Methods and tools for managing scientific information with ontologies. PhD Thesis. Moscow, 2012 (in Russian).
[18]. Pinto David, Jim´enez-Salazar H´ector, Rosso Paolo. Clustering abstracts of scientific texts using the transition point technique. International Conference on Intelligent Text Processing and Computational Linguistics. Springer. 2006, pp. 536–546.
[19]. Scott Deerwester, Susan T Dumais, George W Furnas et al. Indexing by latent semantic analysis. Journal of the American society for information science. 1990. T. 41, № 6, pp. 391.
[20]. Xu Wei, Liu Xin, Gong Yihong. Document clustering based on non-negative matrix factorization. Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. ACM. 2003, pp. 267–273.
[21]. Hofmann Thomas. Probabilistic latent semantic indexing. Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval. ACM. 1999, pp. 50–57.
[22]. Blei David M, Ng Andrew Y, Jordan Michael I. Latent dirichlet allocation. Journal of machine Learning research. 2003. T. 3, № Jan., pp. 993–1022.
[23]. Tomas Mikolov, Kai Chen, Greg Corrado et al. Efficient estimation of word representationsin vector space. arXiv preprint, arXiv:1301.3781. 2013.
[24]. Chao Xing, Dong Wang, Xuewei Zhang et al. Document classification with distributions ofword vectors. Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific. IEEE. 2014, pp. 1–5.
[25]. Le Quoc V, Mikolov Tomas. Distributed Representations of Sentences and Documents. ICML. T. 14. 2014, pp. 1188– 1196.
[26]. Slonim Noam, Tishby Naftali. Document clustering using word clusters via the information bottleneck method. Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval. ACM. 2000, pp. 208–215.
[27]. Cao Qimin, Guo Qiao, Wang Yongliang et al. Text clustering using VSM with feature clusters. Neural Computing and Applications. 2015. T. 26, № 4, pp. 995–1003.
[28]. Hotho Andreas, Maedche Alexander, Staab Steffen. Ontology-based text document clustering.
[29]. Choudhary Bhoopesh, Bhattacharyya Pushpak. Text clustering using semantics. Proceedings of the 11th International World Wide Web Conference. 2002, pp. 1–4.
[30]. Jayarajan Dinakar, Deodhare Dipti, Ravindran B. Lexical Chains as Document Features. Third International Joint Conference on Natural Language Processing. Citeseer. 2008, pp. 111.
[31]. Enrique Amigo´, Julio Gonzalo, Javier Artiles et al. A comparison of extrinsic clustering evaluationmetrics based on formal constraints. Information retrieval. 2009. T. 12, № 4, pp. 461–486.
[32]. Zhao Ying, Karypis George, Du Ding-Zhu. Criterion functions for document clustering: Tech. Rep.: Technical Report, 2005.
[33]. Meil˜a Marina. Comparing clusterings by the variation of information. Learning theory and kernel machines. Springer, 2003, pp. 173–187.
[34]. Hubert Lawrence, Arabie Phipps. Comparing partitions. Journal of classification. 1985. T. 2, № 1, pp. 193–218.
[35]. Bakus J, Hussin MF, Kamel M. A SOM-based document clustering using phrases. Neural Information Processing, 2002. ICONIP'02. Proceedings of the 9th International Conference on. IEEE. T. 5. 2002, pp. 2212–2216.

[36]. Vinh Nguyen Xuan, Epps Julien, Bailey James. Information theoretic measures for clusterings comparison: is a correction for chance necessary? Proceedings of the 26th Annual International Conference on Machine Learning. ACM. 2009, pp. 1073–1080.
[37]. Strehl Alexander, Ghosh Joydeep. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. Journal of machine learning research. 2002. T. 3, № Dec., pp. 583–617.
[38]. Rosenberg Andrew, Hirschberg Julia. VMeasure: A Conditional Entropy-Based External Cluster Evaluation Measure. EMNLP-CoNLL. T. 7, 2007, pp. 410–420.
[39]. Bagga Amit, Baldwin Breck. Entity-based cross-document coreferencing using the vector space model. Proceedings of the 17th international conference on Computational linguistics-Volume 1. 1998, pp. 79–85.
[40]. Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza et al. An extensive comparative study of cluster validity indices. Pattern Recognition. 2013. T. 46, № 1, pp. 243–256.
[41]. Yanchi Liu, Zhongmou Li, Hui Xiong et al. Understanding of internal clustering validationmeasures. 2010 IEEE International Conference on Data Mining. IEEE. 2010, pp. 911–916.
[42]. Er´endira Rend´on, Itzel Abundez, Alejandra Arizmendi et al. Internal versus external cluster validation indexes. International Journal of computers and communications. 2011. T. 5, № 1, pp. 27–34.
[43]. Rousseeuw Peter J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics. 1987. T. 20, pp. 53–65.
[44]. Davies David L, Bouldin Donald W. A cluster separation measure. IEEE transactions on pattern analysis and machine intelligence. 1979. № 2, pp. 224–227.
[45]. Calin´ski Tadeusz, Harabasz Jerzy. A dendrite method for cluster analysis. Communications in Statistics-theory and Methods. 1974. T. 3, № 1, pp. 1–27.
[46]. Bezdek James C, Pal Nikhil R. Some new indexes of cluster validity. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics). 1998. T. 28, № 3, pp. 301–315.
[47]. Ibai Gurrutxaga, In˜aki Albisua, Olatz Arbelaitz et al. SEP/COP: An efficient method to find the bestpartition in hierarchical clustering based on a new cluster validity index. Pattern Recognition. 2010. T. 43, № 10, pp. 3364–3373.
[48]. Halkidi Maria, Vazirgiannis Michalis. Clustering validity assessment: Finding the optimal partitioning of a data set. Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on. IEEE. 2001, pp. 187–194.
[49]. Bird Steven. NLTK: the natural language toolkit. Proceedings of the COLING/ACL on Interactive presentation sessions. Association for Computational Linguistics. 2006, pp. 69–72.
[50]. Scikit-learn: Machine Learning in Python. F. Pedregosa, G. Varoquaux, A. Gramfort et al. Journal of Machine Learning Research. 2011. T. 12, pp. 2825–2830.
[51]. Astrakhantsev N.A., Fedorenko D.G., Turdakov D.Yu. Methods for automatic term recognition in domain-specific text collections: A survey. Programming and Computer Software. 2015. T. 41, № 6, pp. 336–349.
[52]. Astrakhantsev Nikita. ATR4S: Toolkit with State-of-the-art Automatic Terms Recognition Methods in Scala. arXiv preprint, arXiv:1611.07804. 2016.
[53]. Reh˚u˘rek R., Sojka P.˘ Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. Valletta, Malta: ELRA, 2010, pp. 45–50.

[54]. Martin Ester, Hans-Peter Kriegel, J¨org Sander et al. A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd. T. 96. 1996, pp. 226–231.
[55]. Arthur David, Vassilvitskii Sergei. kmeans++: The advantages of careful seeding. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics. 2007, pp. 1027–1035.
[56]. Lang Ken. Newsweeder: Learning to filter netnews. Proceedings of the 12th international conference on machine learning. 1995, pp. 331–339.
[57]. Krapivin M., Autaeu A., Marchese M. Large dataset for keyphrases extraction. 2009. URL: http://eprints.biblio.unitn.it/1671/1/disi09055krapivin-autayeu-marchese.pdf.
[58]. William Hersh, Aaron Cohen, Lynn Ruslen et al. TREC 2007 Genomics Track Overview. 2007.
[59]. Xie Pengtao, Xing Eric P. Integrating document clustering and topic modeling. arXiv preprint, arXiv:1309.6874. 2013.
[60]. Simone Romano, Nguyen Xuan Vinh, James Bailey et al. Adjusting for Chance Clustering Comparison Measures. arXiv preprint, arXiv:1512.01286. 2015.
[61]. Van Craenendonck Toon, Blockeel Hendrik. Using internal validity measures to compare clustering algorithms. AutoML Workshop at ICML 2015, pp. 1–8.
[62]. Field Andy. Discovering statistics using IBM SPSS statistics. Sage, 2013.
[63]. Kendall Maurice G. A new measure of rank correlation. Biometrika. 1938. T. 30, № ½, pp. 81–93.

Fractal Analysis of Growing Cities and its Relationship with Health Centre Distribution

[1, 2 ,4] C.E. Leyton-Pavez <cleyton@ubiobio.cl>
[2] J.M. Redondo <jose.manuel.redondo@upc.edu>
[3] A.M. Tarquis-Alfonso <anamaria.tarquis@upm.es>
[4] J.C. Gil-Martín < joan.carles.gil@upc.edu>
[2,5] J.D. Tellez-Alvarez < jackson.david.tellez@upc.edu>
[1]Dept. Business Management, U. Bio Bio,
Av. Andres Bello 720, E-3800708, Chillan, Chile
[2]Dept. Physical, UPC Barcelona Tech,
B5 Campus Nord, E-08034, Barcelona, Spain
[3]CEIGRAM, UPM, Campus of Practice, University City, E- 28040, Madrid, Spain
[4]Dept. Business Organization, UPC Barcelona Tech,
C5 Campus Nord, E-08034, Barcelona, Spain
[5]Dept. Civil and Environmental Engineering (Institute Flumen),
UPC Barcelona Tech, D1 Campus Nord, E-08034, Barcelona, Spain

Abstract. During recent years, many research group interested in the study of organizational networks from different fields, e.g. studies of behavior of cities growth following the fractal theory. Considering the fractal analysis as a strong tools to analyse the relation between grow of cities and its relation with health center distribution. The propose of this paper is to develop a methodology based on a fractal analysis framework of Chilean public health networks, the population growth of cities and the location of health centers. To produce empirical approach for the analysis of the locations of health centers and its effects. On the other hand the health has been related to other sciences, a clear example is the physics and the mathematics, where we found many studies related with the geometry to understood the nature of the multiple natural shapes, its developed was called fractal nature or mathematic fractal model, this approach consider the different scales. For this reason, we present in this paper some preliminary ideas and some initial results of the analysis of the complex and fractal behavior of urban expansion, and its relation to the location of public health centers through simple counters algorithms and spectral methods using ImaCal software. The study health patterns will become a useful tool for understanding human flow and behavior of health center, where using a traditional measurement as equipment has serious problems and limitations.

Keywords: Multi-Fractals; Urban growth; City hospitals; Health centres; Nonlinear process; Chile health centres

DOI: 10.15514/ISPRAS-2016-29(2)-7

For citation: Leyton-Pavez C.E, Redondo J.M., Tarquis-Alfonso A.M. Gil-Martín J.C., Tellez-Alvarez J.D. Fractal Analysis of Growing Cities and its Relationship with Health Centre Distribution. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 2, 2017, pp. 201-214. DOI: 10.15514/ISPRAS-2016-29(2)-7

1. Introduction

The objective of this work is to propose a methodology based on a multi-fractal analysis framework of Chilean health networks and the relationship between human flow, population growth of cities and he location of health centers, seeking to produce new ideas for the empirical study of the health situation and its scaling and geographic determinants using an open software application ImaCalc.

The establishment of inter organizational [1] relations has been one of the characteristics of the business of the last decades; these relations are constituted as relatively resistant flows and unions that occur between an organization and one or several organisations of its environment. These long-term relationships allow deeper understanding of the structures and how they may maintaining competitive advantages, as they offer the opportunity to work together in a shared knowledge or physical environment. The theoretical advantage is that each organization can concentrate on doing what it does best and hire independent companies to do the rest of its activities, but this simple approach does not take into consideration even scaling effects.

2. Networks in the Health Systems

The Pan American Health Organization (PAHO, 2010) [2] in response to the consequences of heavily fragmented and segmented health systems, has expressed the need to implement Integrated Health Services Networks in the health systems of Latin American and Caribbean countries.

The concept of integrated health services has been favored for decades in the health discourse, however, discrepancies in the scope of implementing real inter-organizational coordination and effective cooperation measures, have shown the difficulty to reverse obstacles, based on the local social, economic and political reality that often do not contribute to the sustainability of integrated systems.

3. Multi-Fractal Ideas and Analysis

Fractal objects are irregular in shape but their irregularity is similar across many scales [3], enabling them to be described mathematically and to be generated computationally. Following the theory of chaos and fractal and multifractal geometry theory, we apply these ideas using the advances of fluid visualization [4], to investigate urban human behaviour, urban growth and relationship with public

health networks in Chile. The vision of complex natural things has changed in recent years [5], because in science, the nonlinear and fractal evolution of plants, feathers, clouds, flowers, rocks, mountains, tapestries and many other things has helped their understanding [5,6,7]. We present some preliminary ideas and some initial results of the analysis of the complex; multifractal behaviour of urban, expansion and their relationship with the location of health centers public through simple box-counting algorithms and spectral methods (Digiflow, ImaCalc, Matlab).These are open distribution, free, or limited academic license programs.

Here we consider multifractals as a statistic distribution that yields useful information even if the underlying structure does not show a complete self-similar or self-affine behaviour, but generalised spaces may help. A clear example of how fractal mathematics benefits health is to occupy it in various structures of the human body, such as in nervous networks, in blood vessels, in lung alveolar structure and even in the distribution of health centers. Chaos and non-linearity, fractal geometry, emergency and catastrophe theory, dynamic systems, and network theory constitute new methods that represent significant potential in the production of scientific knowledge and technological development in the area of health [8, 9].

For a long time, health has been related to other sciences, clear examples are chemistry, physics and mathematics, and now, from a human point of view: topology, when studying geometry multiple natural forms appear triangles, squares, hexagons, etc., but in practice irregular forms of fractal or multi-scale nature dominate. Within the methodology to study the flow location or distribution of health centers in a complex linked network, it is important to consider that the mathematical theory of fractal growth behavior, depends on of the diversity of parameters used in the description *{Xi}*, their time derivative *{dXi/dt}*, acceleration, spatial gradients etc.; with different viewpoints and considerations in a generalized system of coordinates, from a dimensional point of view, the flux of property *Xi* is the derivative in time per unit area [10,11].

4. Study of the Central Chile Area (CCA) Health Structure

Recent research has focused on the management of health networks, which, because of their establishment, growth and development, have formed relationships that are relatively resistant, with quite stable flows and unions that occur between an organization and one or more organizations of its neighborhood, which determine the characteristics of their success, business efficiency and their management [12].

The fundaments of this analysis are to consider the network as objects of health/disease propose a fractal structure of network on the base of the fractal unit, and in each point, to be double or increase the size in the same way. In figure 1, we represent the network of health as groups of four elements as cultural diversity, social development, economic development and the environment [9]. Figure 2 represents the evolution since 2010 and projection of the urban growth of Chillan.

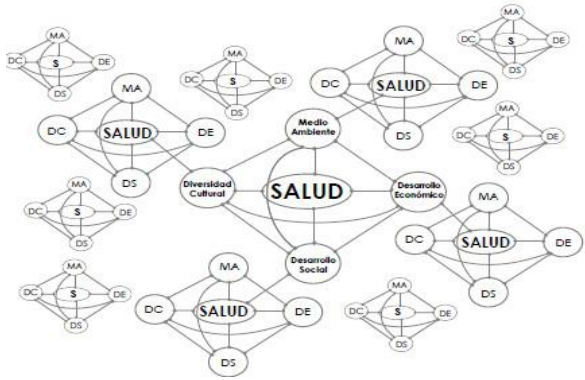


Fig. 1. Health as Fractal Network [9]

4.1 Urban Growth Behavior in Central Zone Chile

The accumulation, now seen from its growth projections, yields quite important figures, almost comparable to the projections the growth of the entire region (Institute Nacional de Statistical - INE 2002). According to projections, without substantial changes in future growth, growth will occur almost entirely in the most attractive cities, therefore, in the central area.

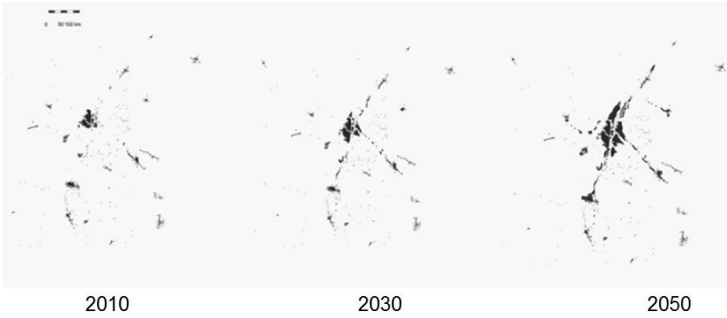


Fig.2. Urban growth – projections Chillan-Chile

There is a great amount of small towns, villages and small villages dispersed in most of the territory, with population voids at greater distances of the central area. These entities are close to large arable areas, natural areas, or simply abandoned areas, which could be characterized as the edges of the supra-region. This, in turn, shows a high rural percentage in the territory and consequently it may be described

as a ring of unprotected areas, for example, in terms of planning, transport, infrastructures, etc.

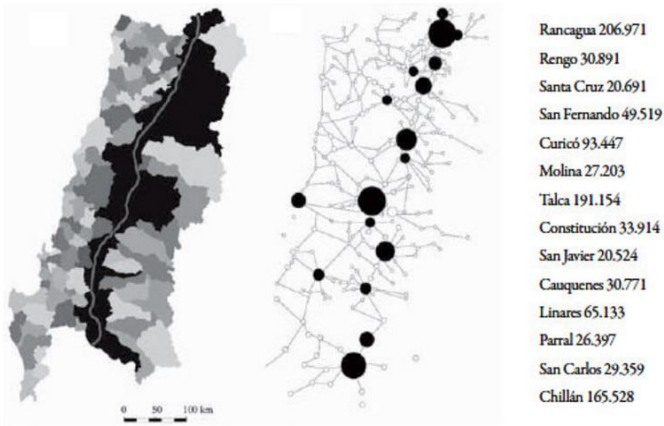


Fig.3. Geographical information and population of the study region, the size of cities is proportional to the number of inhabitants

The urban population of this region is concentrated mainly along the centre of Chile, in a few major cities, which are administrative provincial or regional capitals, including the province of Ñuble. The rest of the population is dispersed throughout the regional territory in innumerable villages, hamlets and even smaller centres. Figure 3 shows the visual information of the population distribution, while figure 4, and also presents the main human flows, both associated to the city structure, the transport infrastructures and the local conditions.

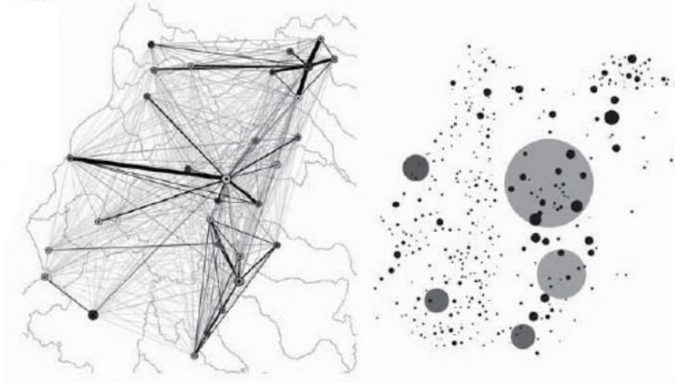


Fig.4. Human flow, population – location near the study area

4.2. The Integrated Networks of Health Services in Central Chile

A network of organizations that provide, or arrange to provide, equitable and comprehensive health services to a defined population, and which is accountable for its clinical and economic results and for the health status of the population at large. We combine here the population structure and the health infrastructures of the area.

Fig.5. Hospital network in the province of Ñuble

The micro-regional capabilities and their integrations, connect areas of the same hierarchy, that is, connecting micro-regional areas with others, in theory, a series of longitudinal corridors are configured. As a result of the geographic conditions and real physical possibilities that the territory offers, alternative areas are developed to the central corridor, which finally acquires the scale of the entire macro-region. Figure 5 represents the main hospital distribution in the CCA.

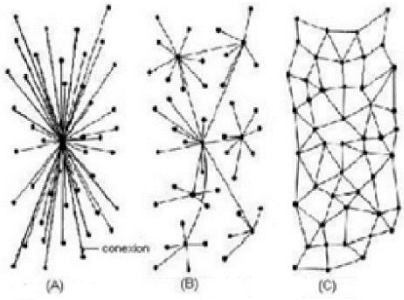


Fig. 6. (A) Centralized, (B) De-centralized y (C) Distributed. The sciences of complexity and medical innovation, Institute of Physics of the Center for Interdisciplinary Research in Sciences and Humanities, National Autonomous University of Mexico, Grama Editora, S.A., 2006

Fractal geometry studies and classifies several possible distributions both in fixed or in dynamical coordinates, for example figure 6 shows three different network (or flux) possibilities of the same geographical data (UNAM), so complex systems allow a wide range of possibilities. From a topological point of view in figure 7, we show how statistically random, but simple fractal geometry, and the evolution of the mono-fractal or (on-off) black-white structure is described with *Do= 1.89.* This is the Sierpinski Carpet in a random set, to show the structure and the classification of complex geometry and how it is possible to determine in this case a single fractal dimension.

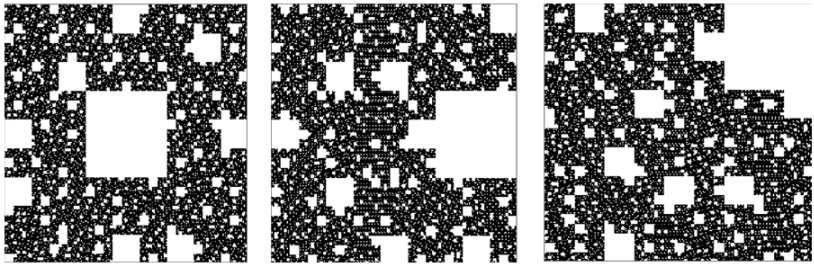


Fig.7. Three types of Random Sierpinski Carpets with five size levels of recurrence. They represent possible levels of Health Centers in an urban area. In spite of their different Networks, the spatial Fractal dimension D = log8/log 3 = 1.8928 is the same but in a statistical sense [13, 14]

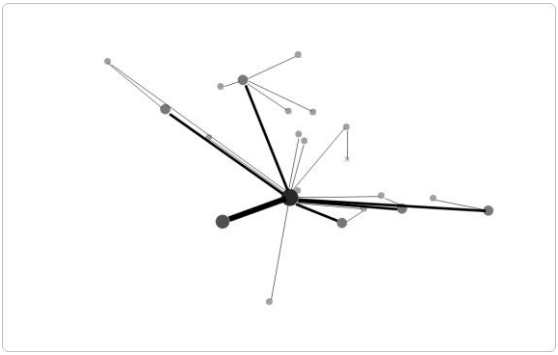


Fig.8. Geographical composition of human flows between the main hospitals in the province of Ñuble. Calculation of hospital levels, high-level complexity (8), medium level complexity (4) and low complexity (1).

Programs coupled to complex non-linear behaviour have been used since R. Thom (1988) [15,17,18], using concepts such as evolution Entropy, but advanced image processing for fluid mechanics to Flux-Force ideas is recent. As an example, the thickness and intensity of the marked roads are proportional to the transit not unlike a blood vessel or a Thermo-Magnetic Flux. In order to show the complexity of the

network relations and interactions, we use the Fractal Dimension as a function of the traffic intensity *i*, we can thereby define the fractal dimension *D (i)* also as a function of the scale *e* of the image. The fractal dimension usually is calculated following this equation:

$$D(i) = -Log(N(i)) / Log(e, i),$$

where *N(i)* is the number of boxes of size e, needed to cover the image contour of intensity *i*. The algorithm used by ImaCalc 1.5 [16, 18] (available in free access from: https://www.academia.edu/420566/ImaCalc_Executable_Program), this program is oriented to calculate the fractal dimension of images using the Box-Counting algorithm. It also includes simple tools for processing and analysis of images operates dividing the 2D surface into smaller and smaller square boxes and counting the number of them that have values close to the level under study, for different iterations, *n*). For each box of size *l/n* it is then decided if the convoluted line, is intersecting that box. The slope of *N* versus the size of the box *e* in a log-log plot, within experimental limits, gives the fractal dimension for a single intensity or for the condition of small boxes: *Lim e -> 0* can be measured by counting the number N of boxes needed to cover the Flux Map under investigation for decreasing box sizes and estimating the limit of the slope.

There are several methods for implementing multifractal analysis; the moment method uses mainly three functions: The mass group of similar transport is conditional to many specific health related aspects, such as size, speciality distribution, etc. this is similar to traffic intensities described in [11, 17]. This method of box-counting in ImaCalc software is used to detect the self-similar behaviour. This analysis can be made for the Scalar marker, for a vector, such as a Vorticity or a generalised variable. In a monofractal object, the number *N* of features of a certain size *e* varies as

$$N(e) = c(e) \exp\{-Do\},$$

where the fractal dimension *Do* at the limit of small size exponent function $\tau(q)$ the coarse Holden exponent, α and the multifractal spectrum, $f(\alpha)$ as seen in figure 8. We have only used a grey scale between 0 to 255, but with more data, it is possible to improve the histogram of the analysed images.

A complex network could be decomposed spatially in terms of infinitely many intertwined sets of fractal dimensions. If that is the case, one fractal dimension cannot characterise all the complexity and several fractal dimensions will be estimate depending on the position and other relevant parameters, such as the patient type of ailment or condition.

Applying box counting “up-scaling” partitioning process we can obtain the partition function relating *m* as the mass of the measure, *q*, the mass exponent, *e* the length size of the box and *N* is the number of boxes in which the statistical moment of the measure, defined as a group of non-overlapping boxes of the same size partitioning the area studied. *Dq (i)* are related to $f(\alpha)$ as multifractal measures are the concept

of generalized dimensions Dq. This set of descriptor corresponds to the scaling exponents for the moment of the measure.

Fractal Dimension as a function of the growing population map of province de Ñuble as intensity *i* of, we can thereby define the fractal dimension *D (i)* also as a function of the scale e of the image. This dimension is usually calculated using: *D (i) = - Log (N(i)) / Log (e, i),* Where *N(i)* is the number of boxes of size *e*, needed to cover the image contour of intensity *i.*

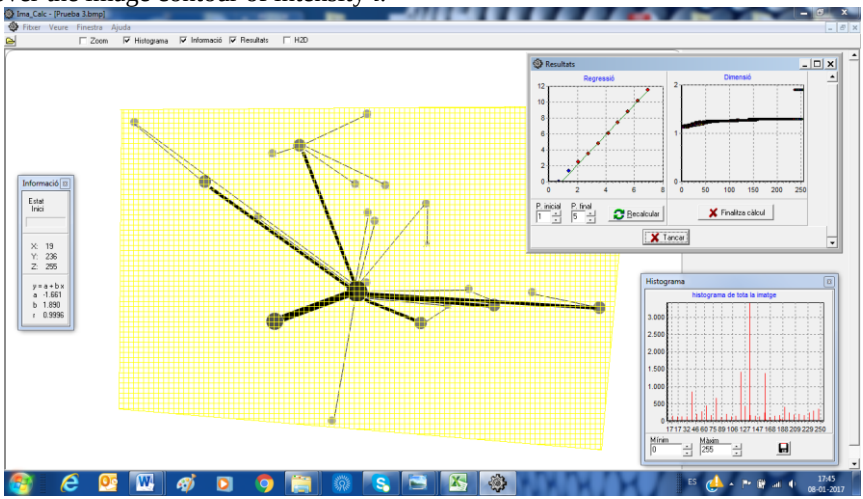


Fig.9. Visualization of ImaCalc applied on the human flow in the province of Ñuble

The algorithm used by ImaCalc operates dividing the 2D surface into smaller and smaller square boxes and counting the number of them that have values close to the level under study, for different iterations [14]. The Sierpinski carpet is used as a heavily urbanized city simple model, with fractal dimension D_{max}= log 8 / log 3 = 1.892789, but using multifractal generation at different intensities, the differences between the spectra of the city structure and that of the sanitary network may be quantified as D_{max} *(i,e)- D_{health} (i,e).*For the multifractal spectra shown in the bottom right corner of figure 8 the urban flow in the province of Ñuble shows a variation of the fractal dimension between D_{max} (i) = 1.15 and 1.25.

This follows the methodology of [17, 19], where use of program ImaCalc is explained further.

5. Conclusions

In this case, the methodology carried out can become a useful tool to understand the human behaviour of the flow approaching the inlet where the traditional measuring equipment has serious problems and limitations.

An important achievement in human studies is the development of new techniques for the measurement and prediction of complex network relations and interactions. The technological advances in "Big Data" digital processing networks and the advances in image processing techniques give the researchers an enormous potential to measure and study the behavior of human population flows, in this context, it is possible to use these non-linear network techniques to study health patterns and patient fluxes related to the scale of the hospitality or road network. The results presented in figure 9, corresponding to the general patient mobility could be upgraded to include the different medical specialties as well as different Indexes associated to the health practice.

References

[1]. *Sánchez, V, Ramírez, R.* (2006). Approach to a Framework for Analysis and Systems Development Management Control interorganizational relationships, Iberoamericana Management Accounting, no. 8, 155-176.

[2]. PAHO (2010). The renewal of primary health care in the Americas: Integrated Networks and Health Services Concepts. Policy Options and a Road Map for Implementation in the Americas. Washington D.C.

[3]. *Rodriguez-Milagros, E.* (2011). Mathematics and its Relationship with the Sciences as a Pedagogical Resource. Magazine Numbers, no. 77, 35–49.

[4]. *Adrian, R.J.* (1991). Particle-Imaging techniques for experimental fluid mechanics, Annual Review of Fluid Mechanics, no. 23, 261-304.

[5]. *Salingaros, N.A.* (1999). Urban Space and its Information Field, Journal of Urban Design 4, 29-49. Reprinted as Chapter 2 of Principles of Urban Structure, Techne Press, Amsterdam, Holland, 2005.

[6]. *Mandelbrot, B.* (1997). New methods in statistical economics. Fractals and Scaling in Finance. Springer, New York, 79-104.

[7]. *Castilla R., Redondo J.M., Gamez P.J. & Babiano A.* (2007), Non Linear Processes in Geophysics, no. 14, 139.

[8]. *Ciuchi, F, Sorriso-Valvo, L, Mazzulla, A, Redondo J.M (2009). Fractal aggregates evolution of methyl red in liquid crystal. The European Physical Journal E 07/2009*, no. *29(2), 139-47.*

[9]. *Almeida-Filho, N.* (2006). Complexity and Trans-disciplinarily in the Collective Health Field: Concepts' Evaluation and Applications. Salud Colectiva, Buenos Aires, no. 2(2), 123-146.

[10]. *Tijera, M., Cano, J. L., Cano, D., Bolster, D., & Redondo, J.M. (2008).* Filtered deterministic waves and analysis of the fractal dimension of the components of the wind velocity. Nuovo Cimento C. Geophysics and Space Physics, no. 31, 653-667.

[11]. *Redondo, J.M.* (1993). Fractal models of density interfaces. In: IMA Conf. Ser. 13, Farge, M., Hunt, J.C.R., Vassilicos J. C. (eds.) Oxford: Clarendon Press, 353–370.

[12]. *Huerta-Riveros, P, Paul-Espinoza, I, Leyton-Pavez, C.* (2012). The impact of health management indicators on a public health service's strategies. Public Health Magazine, no. 14 (2), 248-259.

[13]. *Tarquis, A.M., Platonov, A, Matulka, A, Grau, J, Sekula, E, Diez, M, & Redondo, J.M.* (2014). Application of multifractal analysis to the study of SAR features and oil spills on the ocean surface. Nonlinear Processes in Geophysics, no. 21(2), 439-450.

[14]. *Redondo-Buitrago, Haro-Delicado M.J.* (2004). Fractal Geometry Activities in the Secondary Classroom (I). SUMA, no. 47, 19-28.
[15]. *Thom, R.* (1988). Sketch of a semi physics. Paris: Inter-Editions.
[16]. *Platonov, A., Redondo, J.M., Grau, J.B.* (2008). Aplicación de análisis fractal al estudio de las estructuras dinámicas en fluidos medioambientales. Ingeniería del Agua, vol 15, no 3 pp. 163-174.
[17]. *Leyton, C., Redondo, J. M. Gonzalez-Nieto, P.L. & Tarquis, A.M.* (2016). Fractal Behaviour of Human Fluxes, *Waves and vortices in complex media,* Ishlinski IPM RAS. Moscow. P. 2, 230-345.
[18]. *Grau J.* (2005) Analysis of the Meteosat image sequences using the digital processing method. Ph D. Thesis, UPC, Barcelona.
[19]. Redondo, J.M. (2009). ImaCalc Executable Program https://www.academia.edu/420566/ImaCalc_Executable_Program

Фрактальный анализ растущих городов и его взаимосвязь с распределением центров здоровья

[1, 2 ,4] К.Е. Лейтон-Павес <cleyton@ubiobio.cl>
[2]Х.М. Редондо <jose.manuel.redondo@upc.edu>
[3] А.М. Таркус-Альфонсо <anamaria.tarquis@upm.es>
[4]Х.К. Джил-Мартин <joan.carles.gil@upc.edu>
[2,5] Дж.Д. Теллес-Альварес <jackson.david.tellez@upc.edu>
[1]Dept. Business Management, U. Bio Bio,
Av. Andres Bello 720, E-3800708, Chillan, Chile
[2]Dept. Physical, UPC Barcelona Tech, B5 Campus Nord, E-08034, Barcelona, Spain
[3]CEIGRAM, UPM, Campus of Practice, University City, E- 28040, Madrid, Spain
[4]Dept. Business Organization, UPC Barcelona Tech,
C5 Campus Nord, E-08034, Barcelona, Spain
[5]Dept. Civil and Environmental Engineering (Institute Flumen),
UPC Barcelona Tech, D1 Campus Nord, E-08034, Barcelona, Spain

Аннотация. В последние годы многие научные коллективы исследуют поведение организованных сообществ в различных областях, например, изучают поведение растущих городов в соответствии с теорией фракталов. Теория фракталов является мощным инструментом для анализа взаимосвязи между ростом городов и распределением сети центров здравоохранения для населения. Целью данной работы является разработка методологии, основанной на теории фракталов, для изучения сообщества общественных организаций в области здравоохранения в государстве Чили, а также для изучения роста населения и эффективного расположения медицинских центров для обслуживания населения. Авторами статьи предложен эмпирический подход для анализа ситуации в области здравоохранении и различных эффектов приложения данного подхода. Известно, что фрактальная теория применима

к задачам физики и математики. Имеется много примеров, в которых изучается геометрия объектов для обеспечения понимания многообразных форм в природе. В связи с этим, мы представляем в этой работе ряд идей и свои первые результаты анализа фрактального поведения процесса урбанизации в городах, влияния урбанизации на расположение центров здравоохранения с использованием простейших счетных алгоритмов и спектральных методов с использованием программного обеспечения ImaCal. Результаты исследования позволят лучше понимать разные аспекты миграции населения в связи с размещением центров здоровья людей.

Ключевые слова: сообщества; рост городов; центры здравоохранения; население; теория фракталов; урбанизация; нелинейные процессы.

DOI: 10.15514/ISPRAS-2017-29(2)-7

Для цитирования: Лейтон-Павес К.Е., Редондо Х.М., Таркус-Альфонсо А.М., Джил-Мартин Х.К., Теллес-Альварес Дж.Д. Фрактальный анализ растущих городов и его взаимосвязь с распределением центров здоровья. *Труды ИСП РАН*, том. 29, вып. 2, 2017, стр. 201-214 (на английском). DOI: 10.15514/ISPRAS-2017-29(2)-7

Список литературы:

[1]. *Sánchez, V, Ramírez, R.* (2006). Approach to a Framework for Analysis and Systems Development Management Control interorganizational relationships, Iberoamericana Management Accounting, no. 8, 155-176.

[2]. PAHO (2010). The renewal of primary health care in the Americas: Integrated Networks and Health Services Concepts. Policy Options and a Road Map for Implementation in the Americas. Washington D.C.

[3]. *Rodriguez-Milagros, E*. (2011). Mathematics and its Relationship with the Sciences as a Pedagogical Resource. Magazine Numbers, no. 77, 35–49.

[4]. *Adrian, R.J*. (1991). Particle-Imaging techniques for experimental fluid mechanics, Annual Review of Fluid Mechanics, no. 23, 261-304.

[5]. *Salingaros, N.A.* (1999). Urban Space and its Information Field, Journal of Urban Design 4, 29-49. Reprinted as Chapter 2 of Principles of Urban Structure, Techne Press, Amsterdam, Holland, 2005.

[6]. *Mandelbrot, B.* (1997). New methods in statistical economics. Fractals and Scaling in Finance. Springer, New York, 79-104.

[7]. *Castilla R., Redondo J.M., Gamez P.J. & Babiano A*. (2007), Non Linear Processes in Geophysics, no. 14, 139.

[8]. *Ciuchi, F, Sorriso-Valvo, L, Mazzulla, A, Redondo J.M (2009). Fractal aggregates evolution of methyl red in liquid crystal. The European Physical Journal E 07/2009,* no. *29(2), 139-47.*

[9]. *Almeida-Filho, N.* (2006). Complexity and Trans-disciplinarily in the Collective Health Field: Concepts' Evaluation and Applications. Salud Colectiva, Buenos Aires, no. 2(2), 123-146.

[10]. *Tijera, M., Cano, J. L., Cano, D., Bolster, D., & Redondo, J.M. (2008).* Filtered deterministic waves and analysis of the fractal dimension of the components of the wind velocity. Nuovo Cimento C. Geophysics and Space Physics, no. 31, 653-667.

[11]. *Redondo, J.M.* (1993). Fractal models of density interfaces. In: IMA Conf. Ser. 13, Farge, M., Hunt, J.C.R., Vassilicos J. C. (eds.) Oxford: Clarendon Press, 353–370.

[12]. *Huerta-Riveros, P, Paul-Espinoza, I, Leyton-Pavez, C.* (2012). The impact of health management indicators on a public health service's strategies. Public Health Magazine, no. 14 (2), 248-259.

[13]. *Tarquis, A.M., Platonov, A, Matulka, A, Grau, J, Sekula, E, Diez, M, & Redondo, J.M.* (2014). Application of multifractal analysis to the study of SAR features and oil spills on the ocean surface. Nonlinear Processes in Geophysics, no. 21(2), 439-450.

[14]. *Redondo-Buitrago, Haro-Delicado M.J.* (2004). Fractal Geometry Activities in the Secondary Classroom (I). SUMA, no. 47, 19-28.

[15]. *Thom, R.* (1988). Sketch of a semi physics. Paris: Inter-Editions.

[16]. *Platonov, A., Redondo, J.M., Grau, J.B.* (2008). Aplicación de análisis fractal al estudio de las estructuras dinámicas en fluidos medioambientales. Ingeniería del Agua, vol 15, no 3 pp. 163-174.

[17]. *Leyton, C., Redondo, J. M. Gonzalez-Nieto, P.L. & Tarquis, A.M.* (2016). Fractal Behaviour of Human Fluxes, *Waves and vortices in complex media*, Ishlinski IPM RAS. Moscow. P. 2, 230-345.

[18]. *Grau J.* (2005) Analysis of the Meteosat image sequences using the digital processing method. Ph D. Thesis, UPC, Barcelona.

[19]. *Redondo, J.M.* (2009). ImaCalc Executable Program https://www.academia.edu/420566/ImaCalc_Executable_Program

Turbulent convection by thermoelectricity in a cooling-heating didactive device

[1,3] *J.M. Redondo <jose.manuel.redondo@upc.edu>*
[1,2,3] *J.D. Tellez-Alvarez <jackson.david.tellez@upc.edu>*
[3] *J.M. Sanchez <berotza@berotza.org>*
[1] *Dept. Física, UPC BarcelonaTech, Barcelona, Spain*
[2] *Dept. of Civil and Environmental Engineering (Institute Flumen), UPC Barcelona, Barcelona, Spain*
[3]*Berotza S.L., Noain, Pamplona, Navarra, Spain*

Abstract. Local Diffusion and the topological structure of vorticity and velocity fields is measured in the transition from a homogeneous linearly stratified fluid to a cellular or layered structure by means of convective cooling and/or heating. Patterns arise by setting up a convective flow generated by an array of Thermoelectric devices (Peltier/Seebeck cells) these are controlled generating a buoyant heat flux. The experiments described here investigate high Prandtl number mixing using brine and fresh water in order to form density interfaces and low Prandtl number mixing with temperature gradients. The set of dimensionless parameters define conditions of numeric and small scale laboratory modeling of environmental flows. Fields of velocity, density and their gradients were computed and visualized using the open software tools of DigiFlow. When convective heating and cooling takes place in the side wall of a stratified enclosed cell, the combination of internal waves and buoyancy driven turbulence is much more complicated if the Rayleigh and Reynolds numbers are high. Higher order moments calculations and intermittency are important in order to study mixing in complex flows Here some examples are shown using the Thermoelectric Convection Didactive Device (TCDD) built by BEROTZA, mainly in a symmetric two dimensional pattern, but many other combinations, using heating-cooling and angles with the vertical are possible in order to validate more complex numerical experiments.

Keywords: convection; thermoelectricity; peltier effect; experiments; numerical simulation; K-e Model; turbulence; digiFlow.

DOI: 10.15514/ISPRAS-2017-29(2)-8

For citation: Redondo J.M., Tellez J.D., Sanchez J.M. Turbulent convection by thermoelectricity in a cooling-heating didactive device. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017. pp. 215-230. DOI: 10.15514/ISPRAS-2017-29(2)-8

1. Introduction

The combination of laboratory experiments and direct numerical simulations of the same experiments is an important source of information when studying the role of body forces in the turbulent cascade that takes place at large Reynolds numbers.

Richardson proposed fully developed turbulence as a hierarchy of eddies of different size. He assumed a cascade process of eddies breaking down. At eddies of size L energy is injected, then energy is transmitted to smaller and smaller eddies and finally it is dissipated in small eddies of scale η where viscosity plays a dominant role. A central role in this scheme is played by the mean rate of energy transfer per unit mass [1-3]. Rotation and Convection due to buoyancy are clear examples of the inverse cascade behavior [4,5]. We propose a combination of these effects in new experiments placing the device described below (Figures 1, 2) TC DD built by BEROTZA , further experiments using a rotating table will be reported elsewhere [6,7].

Here, we will describe the Thermoelectric, Thermal Convection Apparatus that may simulate easily Dirichlet and Newmann side and bottom heated or cooled wall boundary conditions. The Numerical validation of Environmental Turbulence programs with Laboratory Experiments as well as with field data and numerical models is fundamental to build student confidence. The possibilities of using the thermoelectric driven convective device in complex set-ups is also very interesting for didactive purposes, using student teams to model and validate different (but similar) configurations (varying the heat flux, its direction, the angle with gravity, using rotation,...) are very wide.

We first describe the Thermoelectric device and present some results obtained with DigiFlow [5] for 2D basic configurations, Then we introduce a RANS approach with K-e turbulence's model with side-wall heating-cooling and finally we discuss scaling in the 2D-3D transition leading to Buoyant-Convection, which is similar to the Rayleigh-Taylor flows multifractal scaling [8,9].

2. Thermoelectric control and experiments

The possibilities of heating and or cooling, are coupled with new methods of flow visualization exploited by DigiFlow [2, 5], and these new techniques allow us to map velocity and vorticity fields for these types of flows in the desired planes within the TCDD, both in 2D and 3D. We have modified existing methods based on techniques such as Laser Induced Fluorescence(LIF) or Correlation Image Velocimetry (CIV), Shadowgraph and Schiliering, in order to also allow comparisons of higher order descriptors of the turbulence and of the scale to scale transfer of (Scalar Tracers, Energy, Enstrophy, etc.). These methods involve the use of multi-fractal analysis as described in [9, 10].

The possibility of heating-cooling in the full side, by means ow two Peltier cells acting on 5 cm x 5 cm inox steel plates as seen in figure 2 with a constant heat flux regulated by Voltage and Intensity from a versatile control, now allows to set the same conditions as in the numerical simulations. The simple flow used here is just heating in one lateral side and cooling in the opposite one with the same heat flux to avoid large changes in the temperature. An additional complication is to use brine in order to strongly stratify the flow and allow to propagate internal waves driven by

the convective flow. The asymmetry of the flow is reflected by the differences in vorticity between the bottom and top of the TCDD.

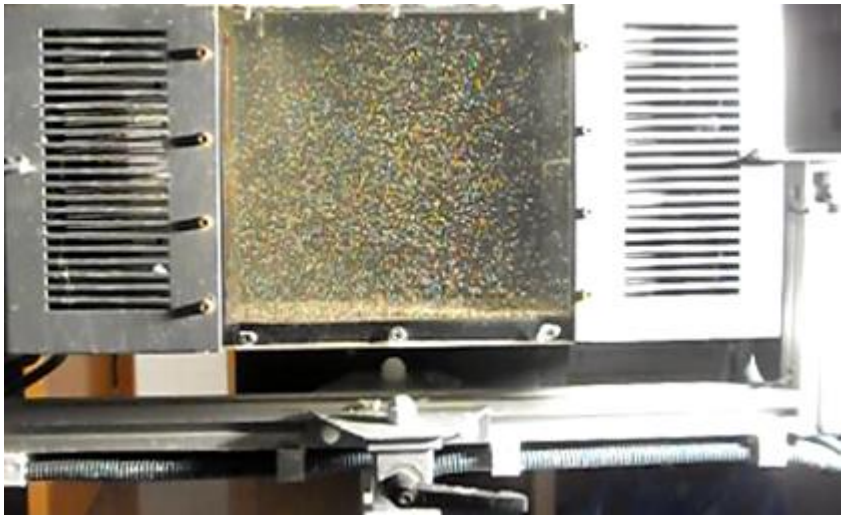


Fig. 1. Experimental Thermoelectric Driven Didactic Apparatus, used to generate cooling and heating at side or bottom boundary conditions. Here seeding for PIV is shown.

Convective mixing takes place, depending on the many possible boundary conditions regulating the cooling or heating fluxes with a transient phase, in most of the high thermal flux set-tings, the sides of the thermal elements produce fast ascending or descending plumes, It is very didactive to observe the different parts of the flow near regions of positive or negative divergence Here, both the temperature (measured by probes) and velocity (measured by plane PIV [2, 5] show different shapes of scalar spectra, the energy spectra, and the relationship be-tween vorticity and stream functions, when only plane visualizations are available. These phenomena are similar in other flows where, the large coherent structures or Super-Cells, blobs or spikes [5, 8] are produced by an inverse cascade or by baroclinic effects that are very common in geophysical driven flows. To present the high resolution of the velocity and vorticity Lagrangian data, Figures 3 to 6 show aspects of the flow obtained by DigiFlow and ImaCalc software: (http://www.winsite.com/Multimedia/Image-Editors/DigiFlow/) (https://www.academia.edu/420566/ImaCalc_Executable_Program).

The DigiFlow application was designed to provide a range of image processing features designed specifically for analyzing fluid flows. The package is designed to be easy to use, yet flexible and efficient, and includes a powerful yet flexible macro language. Whereas most image processing systems are intended for analyzing or processing single images, DigiFlow is designed from the start for dealing with

sequences or collections of images in a straightforward manner. Some key features of DigiFlow have been designed from the outset to provide a powerful yet efficient environment for acquiring and processing a broad range of experimental flows to obtain both accurate quantitative and qualitative output. Central to design philosophy is the idea that an image stream may be processed as simply as a single image. Image streams may consist of a sequence of images and eefficiency is obtained through the use of advanced algorithms (many of them unique to DigiFlow/DigImage). Power and flexibility are obtained through an advanced fully integrated macro interpreter providing a similar level of functionality to standard applications such as MatLab. This interpreter is available to the user either to directly run macros, or as part of the various DigiFlow tools to allow more flexible and creative use. DigiFlow retains the potential to control a frame grabber, which greatly simplify the process of running experiments, acquiring images, processing them, extracting and plotting data, as it also enables real-time processing of particle streaks, PIV and synthetic schlieren, as shown in figures 3 to 7.

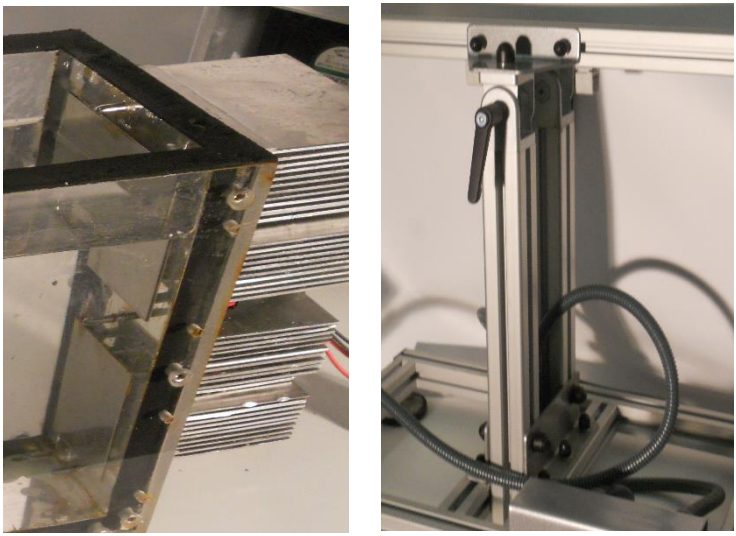


Fig. 2. Details of the Convective Thermoelectric Device produced by BEROTZA with details of the TECs dissipaters (left) and of the versatile position and angle holds [3, 9].

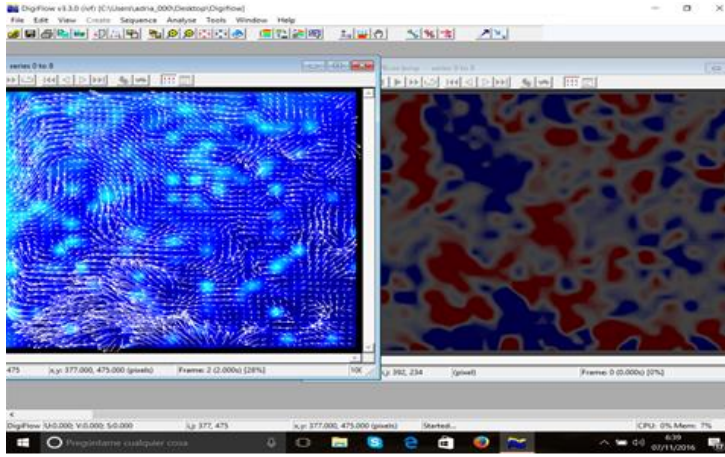


Fig. 3. Velocity and Vorticity patterns formed by sudden heating of a distributed TEC with uniform heat flux and non-homogeneous density [8, 10]. The DigiFlow Environment Program is shown, with some of the open software icons on top of the figure [5, 17]

Open Foam and some 2D RANS with K-e turbulence's model simulations agree with the experiments in the very simple single convective cell situations of low Rayleigh numbers, but it is difficult to model the more complex 3D flows, what is not well understood is the role of the local mixing on the flows [11-16]. The combination of experiments and simulations is important when giving further insight on the different direct and inverse cascades processes that take place in the flow. New experiments, such as the one shown (Fig 3, 4, 5.) show the velocity and vorticity structure. By adding dye, the tracer structure and density spectra, may be studied. The vorticity spectra may control the complex structure evolution (Fig 4). which clearly shows a fractal structure in the appearance of connective local structures, when the TCDD is heated from below, even in a uniform (constant density experiment) There are few examples to compare between the experiment and the numerical simulation of even a simple theoretically model of the growth of the Benard–Taylor mixing layer in the limit of very high Reynolds number (Malkus regime, where the Nusselt number scales as one third of the Rayleigh number). The self-similar mixing layer in unstable flows is predicted to grow with a limit given by buoyancy (Ozmidov) scale so the Nusselt number depends on the Rayleigh number, these types of strong heating are difficult to realize in the laboratory [8, 9], as shown in figures 5 and 6. The presentation of such results as cascade slope or intermittency in wider parametric spaces (Akin to Poincare maps) [10-12] are expected to help in the understanding of turbulent mixing (Fig. 6).

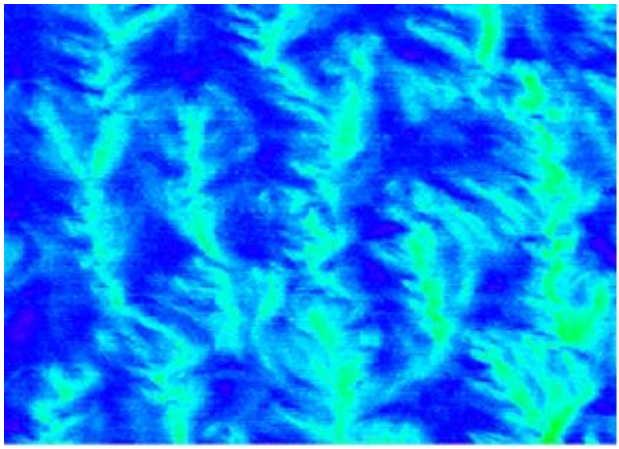


Fig. 4. Centers of Convective cells formed by sudden heating at the base of a distributed TEC with uniform heat flux [3, 9]

3. Numerical Model

Convective mixing takes place, mostly in the sides of ascending and descending plumes, different parts of the flow show shapes of the multi-fractal spectra, but most cases it is not easy to interpret, in some sense, it is similar in other flows where, the large coherent structures, i.e. blobs and spikes [7, 11].

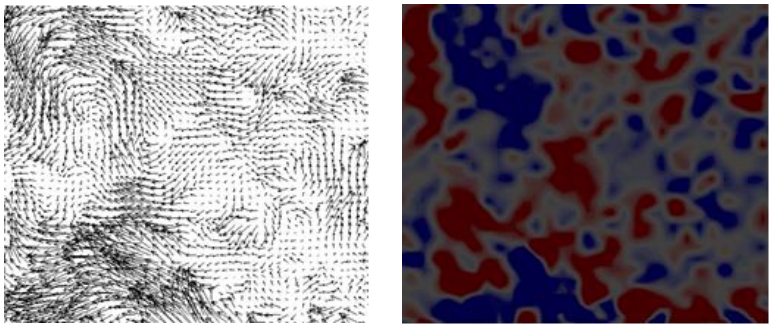
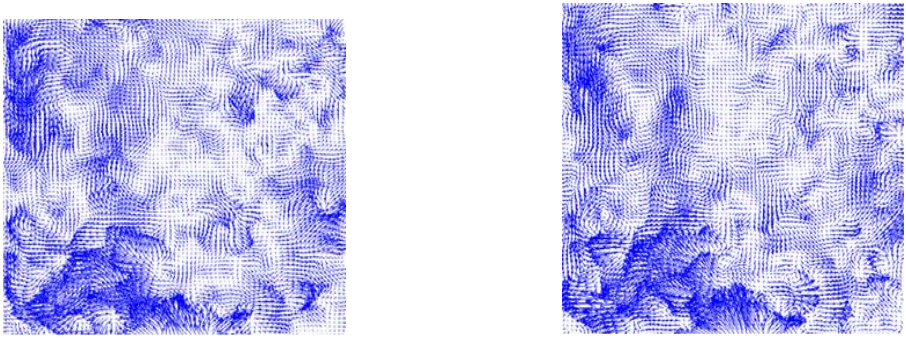


Fig. 5. Thermoelectric Convection induced flow Velocity (left), Vorticity (right) bottom Distribution of the corner driven vorticity complex 3D structure [13]

A key feature is that dominant flows of the CTDD take place in the center or symmetry plane (Figure 6). So far only flat, experiments, with no angle between gravity and the side walls have been performed, but a wide range of new configurations are possible [3, 6].



a) b)

Fig. 6. Sequence of Convective induced flow Velocity (left). The corner driven flow is the most energetic creating a vorticity 3D structure [13,14]

A Reynolds-averaged Navier-Stokes (RANS) approach with a k-e closure scheme was used to simulate the evolution of the rise of the thermoelectric driven convective structure. This is an interesting problem to investigate the mixing process, The model solves the Boussinesq equations in a two dimensional grid and is based in Cantalapiedra and Redondo [15], Versteegh [16] with updrafts and downdrafts clearly separated. Here 1:1 aspect ratio cells heated at the side walls were compared with the experiments, but other configurations were used, with a significant interest in the maximum growth flow at 2.7:1 ratio. At aspect ratios of 8:1 a complex convective pattern develops showing some asymmetry as in the experiments .

The present study is confined to the analysis of aspect ratio convection of 1:1 in an incompressible fluid using a Reynolds-averaged approach. The governing equations for the conservation of mass and momentum as:

$$\frac{\partial \overline{u_j}}{\partial x_j} = 0, \quad \frac{D\overline{u_i}}{Dt} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\nu \frac{\partial \overline{u_i}}{\partial x_j}\right] - \frac{\partial \overline{u_i' u_j'}}{\partial x_j} \quad \text{with}$$

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \overline{u_j}\frac{\partial}{\partial x_j}$$

Here, p, ρ, ν and $\overline{u_i}$ denote the pressure, density, viscosity and the Reynolds-averaged velocity based on Cartesian tensor notation, respectively. A turbulence closure model for the unknown Reynolds stresses $\overline{u_i' u_j'}$ is necessary to obtain a closed system of equations, the election of the method is linked to the degree of mixing in the process. DNS, LES, KS, RANS include closures featuring different degrees of complexity and predictive quality. Implicit second-moment closures utilize individual transport equations for each component of $\overline{u_i' u_j'}$, which is

computationally demanding. Other applications use explicit Reynolds-stress closures, which consists of two parts, a stress-strain relation and a background model. The stress-strain relation describes the Reynolds stresses as a function of the mean-velocity gradients and the considered unknown turbulent scalars. The background model comprises the transport equations for the considered, relevant turbulent parameters.

The most common approach is a two-equation model, based on two transport equations for the unknown scalars, i.e. the turbulence energy $k = 0.5\overline{u_i' u_i'}$ and the energy-dissipation rate ε. Various alternative formulations exist, e.g. $k-\omega$, $k-l$, $k-\tau$, which might yield a change of the Reynolds-stress magnitudes but do not alter the structure of the stress tensor. The active components of the Reynolds-stress tensor – in particular the degree of stress anisotropy – are primarily governed by the employed stress-strain relation for a given strain field. The anisotropy tensor is – of course – also influenced by the background model since, it globally scales with the turbulent time scale. The latter is, however, only a scalar, which carries no structural or tensorial information and thus creates no anisotropy on its own. The present paper is thus confined to a specific RANS approach with low-Re $k-\varepsilon$ turbulence's model that solves:

$$\frac{Dk}{Dt} - \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\mathrm{Pr}_k}\right)\frac{\partial k}{\partial x_j}\right] = \mathrm{P} - \varepsilon,$$

$$\frac{D\varepsilon}{Dt} - \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\mathrm{Pr}_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_j}\right] = \frac{\varepsilon}{k}\left(C_{\varepsilon 1}\mathrm{P} - C_{\varepsilon 2}\varepsilon\right),$$

with

$$C_{\varepsilon 1} = 1.44 f_1,\quad C_{\varepsilon 2} = 1.92 f_2,\quad \mathrm{Pr}_k = 1,\quad \mathrm{Pr}_\varepsilon = 1.3,$$

$$R_t = \frac{k^2}{\nu\varepsilon},\quad R_k = \frac{\sqrt{k}n}{\nu},$$

$$f_1 = 1 + \frac{\mathrm{P}}{\mathrm{P}^*},\ f_2 = 1 - 0.3e^{-R_t^2},\ \text{and}\ f_\mu = \frac{1 - e^{-\alpha_\mu R_k}}{1 - e^{-\alpha_\varepsilon R_k}},\ \mathrm{P} = -\overline{u_i' u_j'} S_{ij},$$

$$\mathrm{P}^* = \frac{f_2 C_{\varepsilon 2} k^{1.5}}{C_{\varepsilon 1} L_\varepsilon} e^{-\alpha_d R_k^2},\ \text{with}\ L_\varepsilon = \kappa \overline{c_\mu}^{(-0.75)} n\left(1 - e^{-\alpha_\varepsilon R_k}\right),\ \text{and}$$

$$\overline{c_\mu} = 0.09, \alpha_d = 0.0022,\ \alpha_\varepsilon = 0.263,\ \alpha_\mu = 0.016.$$

The most popular representative of an explicit Reynolds-stress closure is the well known linear Boussinesq-viscosity model where: $b_{ij} = -\overline{c_\mu} T_t S_{ij}$.

In above equation, $b_{ij} = \left(\overline{u_i u_j} - \frac{2}{3} k \delta_{ij} \right) / 2k$ is the Reynolds-stress anisotropy tensor, $S_{ij} = \left(\partial \bar{u}_i / \partial x_j + \partial \bar{u}_j / \partial x_i \right) / 2$ and $T_t = k / \varepsilon$ denote the strain-rate tensor and the turbulent time scale respectively. The main reason for the popularity of this model is the similarity to the definition of viscous stresses, which simplifies the numerical implementation [15-18].

Under the assumption of a steady 2D boundary layer flow (i.e., u>>v, $\partial / \partial y >> \partial / \partial x$), the governing equations for the velocity and temperature, in forced convection, are:

Conservation of mass:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

Conservation of momentum:

$$u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = \frac{\partial}{\partial y} \left(\nu \frac{\partial u}{\partial y} - \overline{u' v'} \right)$$

Conservation of energy:

$$u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} = \frac{\partial}{\partial y} \left(\alpha \frac{\partial T}{\partial y} - \overline{v' T'} \right)$$

As this system of equations contains more unknowns than equations, so it is an open system. Here the closure of averaged above equations is ensured by different low Reynolds number $k - \bar{\varepsilon}$ turbulence models. The mean turbulent kinetic energy and its dissipation rate may be written in the following form for Newman conditions [16,17]:

$$u \frac{\partial k}{\partial x} + v \frac{\partial k}{\partial y} = \frac{\partial}{\partial y} \left(\left(\nu + \frac{\nu_t}{\sigma_k} \right) \frac{\partial k}{\partial y} \right) + \nu_t \left(\frac{\partial u}{\partial y} \right)^2 - \tilde{\varepsilon} - D$$

$$u \frac{\partial \tilde{\varepsilon}}{\partial x} + v \frac{\partial \tilde{\varepsilon}}{\partial y} = \frac{\partial}{\partial y} \left(\left(\nu + \frac{\nu_t}{\sigma_\varepsilon} \right) \frac{\partial \tilde{\varepsilon}}{\partial y} \right) + C_{\varepsilon 1} f_1 \frac{\tilde{\varepsilon}}{k} \nu_t \left(\frac{\partial u}{\partial y} \right)^2 - C_{\varepsilon 2} f_2 \frac{\tilde{\varepsilon}^2}{k} + F$$

The Reynolds stress $\overline{u' v'}$ and the heat flux $\overline{v' T'}$ appearing in the system of conservation of momentum and conservation of energy equations, may be expressed as:

$$\overline{u' v'} = -\nu_t \frac{\partial u}{\partial y}, \qquad \overline{v' T'} = -\frac{\nu_t}{\mathrm{Pr}_t} \frac{\partial T}{\partial y}$$

The eddy viscosity ν_t is related to k and $\tilde{\varepsilon}$ through the Kolmogorov-Prandtl relations as:

$$\nu_t = C_\mu f_\mu \frac{k^2}{\tilde{\varepsilon}}$$

$\sigma_k, \sigma_\varepsilon, C_{\varepsilon 1}$ and $C_{\varepsilon 2}$ are empirical constants and f_1, f_2 and f_μ are the turbulence model functions for the near wall formulation. For low Reynolds numbers and the simple side wall heated experiments, good qualitative average 2D flows were obtained, even showing the asimetría of corner vortex formation as seen in the experiments (figure 7) and in the numerical simulations (figure 8). The problem with these models is detected when higher order moments and intermittency is evaluated. Mixing evaluations are very poor.

4. Discussion and Conclusions

The calibration of fluids dynamics (CFD) source codes such as (OpenFoam, Imcompact3D, DigiFlow) for numerical simulation of different experiments is important at different resolutions, here spectral and fractal measurements in large Reynold numbers water or wind tunnels or experiments like the ones performed in Multiflow and EU Hit [21-23] are fundamental to write new solvers for different problems in tensorial partial differential equation from [11, 12], but considering fractal aspects. A flow field may be solved by an adaptation of original turbulence solvers for incompressible fluid (simpleFoam, pisoFoam, pimpleFoam). The core of these solver is the finite volume discretization of the governing equations. Almost all kinds of differential operators possible in a partial differential equation, such as temporal derivative, divergence, Laplacian operator, vorticity, etc. Even in the case of stratified flow [13] DNS and LES codes allow to predict and calculate structure functions and local intermittency [4,10], but detailed calibrations are also needed , in [19] it was noted how the transition between 2D and 3D flows produced different scale to scale cascades which departed from Kolmogorov local processes. By the use of experimental benchmarks such as the TCDD described here, it is possible, thanks to advanced visualization software such as DigiFlow, ImaCalc [20-22] to have similar resolution in laboratory experiments and numerical simulations as seen in figures 8 and 9 [14, 15].

By using the dimensionless variables, most of the times in a very large and complex parametric space, the experiments and simulations fit. The challenge is not just to model the system of conservation of mass, conservation of momentum, conservation of energy and the mean flow and turbulent kinetic energy, but also model and measure the structure functions, the fractal aspects and the intermittency, as in [23] for wind generator wakes.

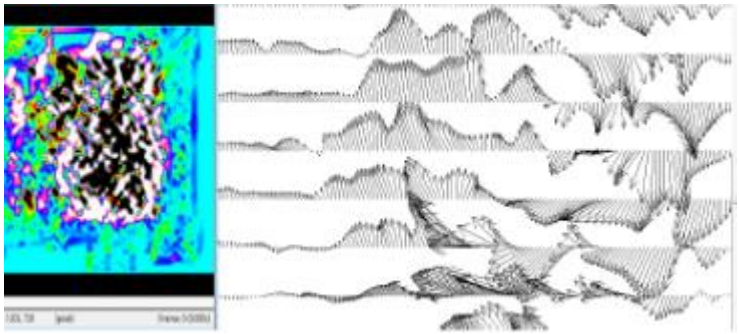


Fig 7. Dominant Vortical structure and Velocity fluxes(left), The Large scale structure is seen on the right, Structure functions and fractal structure may be calculated as in [10, 15]

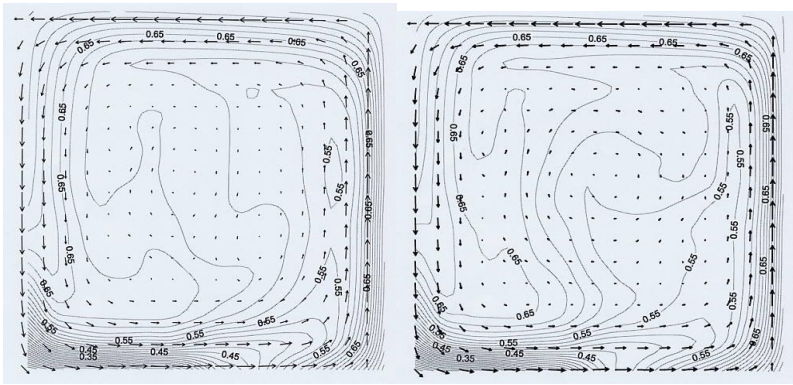


Fig 8 . K-e model of a 2D connective cell in an 1:1 ratio enclosure. The trecirculating average flow may be clearly seen [13]

Figures 7 and 8 show the global flow and the acceleration produced due to boundary layer induced buoyancy in an enclosed flow. When buoyancy acts on a density interface, Baroclinic vorticity is also generated internally and Rayleigh-Taylor flows advance in a non-linear fashion [8,9, 14], here only DNS models are able to predict mixing. But there is still a need to evaluate the multi-fractal and intermittent nature of the different forcing conditions. The detailed comparisons of experiments and numerical simulations, both in physical and spectral domains is necessary to understand the complex flows at high Reynolds numbers.

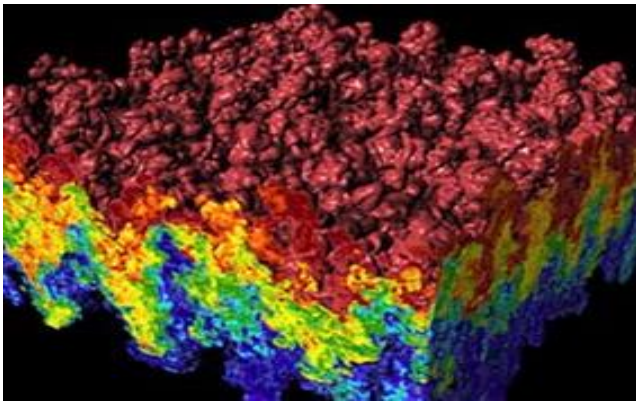


Fig 9 . DNS of a Rayleigh-Taylor front where the filamen tation may be seen[13]

Even considering intermittency in Kolmogorov's turbulent cascade, such complex flows are known to generate non-equilibrium turbulence, and probably also non-local turbulence. This produces different turbulence properties and modified mixing in strong convection as shown here with the TCDD. Also downstream of fractal grids (EU Hit proposal) the flow exhibits different mixing as compared with the classical uniform grids. Applications to enhanced mixing and drag reduction are still being investigated [21, 22], even in other types of fractal fluxes. How do the turbulence and mixing properties change with Rayleigh and Reynolds numbers in different parameter spaces. In order to answer this, it seems necessary to investigate both numerically and experimentally the following properties: The turbulence intensity; the Reynolds stresses; The energy budget. The scalar gradients. The scalar fluctuations. The Skeweness and the Kurthosis of flow velocity and vorticity, kinetic energy and enstrophy both for scalar tracer and flow. With the didactic apparatus and the software described above, we believe that the role of intermittency and of the structure functions may be compared and evaluated, which is very important in complex and environmental flows [23].

Acknowledgments

Support from European Union through ISTC-1481 and from MCT-FTN2001-2220 , DURSI XT2000-0052 projects are acknowledged. PELNoT-ERCOFTAC and BEROTZA-CTT local projects as well as MULTIFLOW and EU Hit 2016 have funded some of this work. We thank Profs. M.G. Velarde, P. Fraunie and Y. Chashechkin for interesting discussions.

References

[1]. Sekula E., Redondo J.M. The structure of Turbulent Jets, Vortices and Boundary Layers. Il Nuovo Cimento. 2008. V. 31. P. 893 – 907.

[2]. Redondo J.M. Introduction to Vortex Theory in Turbulence Laboratory Visualization, UPC. Lecture Notes Dept. Fisica. PELNHoT, 2015. P. 160
[3]. Redondo J.M., Garriga J.N. Convection driven by thermoelectric heat fluxes. Berotza, Report UPC, 1995. P. 98.
[4]. Stepanova E.V. Study of the rotating fluid surface form in the cylindrical container. Selected papers of international conference “Fluxes and structures in fluids”. Moscow. 2006. P. 313–319.
[5]. Dalziel, S.D., Redondo J.M. New visualization and self-similar analysis in experimental turbulence studies. Models, Experiments and Computation in Turbulence. CIMNE, Barcelona.
[6]. Kuramitsu, M., Redondo, J.M., Noriega, G. Measurements of anysotropy, thermoelectric Proceedings of the IEEE 2003, V. 22, P. 541-545.
[7]. Castilla R., Onate E. and Redondo J.M. Models, Experiments and Computations in Turbulence. CIMNE, Barcelona. 2007. P. 255.
[8]. Redondo, J. M., Sanchez, M. A., Garriga, J., Castilla, R., Convective and Rayleigh-Taylor Instabilities in Stratified Fluids. Advances in Turbulence V. 1995. P. 428-434.
[9]. Vila T., Tellez J., Sánchez M. J., Sotillos L., Diez M. and Redondo J.M. Diffusion in fractal wakes and convective thermoelectric flows. GRA– EGU. 2014.V.16, p.2011.
[10]. Tarquis, A. M., Platonov A., Matulka A., Grau J., Sekula E., Diez M., Redondo J. M.. Application of multifractal analysis to the study of SAR features and oil spills on the ocean surface. Nonlinear Processes in Geophysics 21, no. 2 2014: P. 439-450.
[11]. Redondo J.M., Turbulence Entropy and Dynamics, UPC Lecture Notes. 2014.
[12]. Redondo, J.M., The structure of density interfaces. Ph.D. Thesis. Univ. of Cambridge. Cambridge. DAMTP 1990.
[13]. Redondo J.M. and Metais O. Mixing in Geophysical Flows. CIMNE, Barcelona. 1995, p. 415.
[14]. Linden P.F., Redondo. J.M. Turbulent mixing in Geophysical Flows. CIMNE, Barcelona, 2001. p. 319.
[15]. Cantalapiedra I.R. And Redondo J.M. Mixing in zero-mean-flow turbulence, Mixing in Geophysical Flows (Redondo J.M. and Metais O. Eds.) CIMNE, 1995, 127-146.
[16]. Versteeg T.A.M. And Nieuwstadt F.T.M. Numerical simulation of buoyancy driven flow in enclosures. Mixing in Geophysical Flows (Redondo J.M. and Metais O. Eds.) CIMNE, 1995, 329-342
[17]. Redondo J.M. Mixing efficiency of different kinds of turbulent processes and instabilities; Appplications to the environment. CIMNE, 2001, 131-157.
[18]. Redondo J.M. Termodinámica de los procesos irreversibles, efectos termoeléctricos. Rev. Termoelectricidad, 1995 V.2.
[19]. Mahjoub O.B. Granata T. and Redondo J.M. Scaling Laws in Geophysical Flows. Phys. Chem. Earth (B) V 26, no.4 p281-285, 2001.
[20]. Dalziel R. P. DigiFlow User Guide. Cambridge, England: Department of Applied Mathematics and Theoretical Physics (DAMTP) - University of Cambridge. 2012.
[21]. Tellez J., Gómez. M., Russo M., Redondo, J.M. Surface Flow Image Velocimetry (SFIV) for Hydraulics Applications In 18th International Symposium on the Application of Laser and Imaging Techniques to Fluid Mechanics. July 4-7 2016. Lisbon, Portiugal.
[22]. Leyton C., Redondo J.M, Gonzalez-Nieto P.L. and Tarquis A.M. Fractal behaviour of human fluxes.Waves and vortices in complex media, Moscow, IPM RAS. 2016. V.2 p.230-235.

[23]. Strijhak S. Redondo J.M. And Tellez J. Multifractal Analysis of a Wake for a Single Wind Turbine Proceedings Topical Problems of Fluid Mechanics 2017, Prague, 2017 Edited by David Šimurda and Tomáš Bodnár, pp. 275-284.

Турбулентная конвекция термоэлектричеством в охладительно-нагревательном устройстве

[1,3] *Х.М. Редондо <jose.manuel.redondo@upc.edu>*
[1,2,3] *Дж.Д. Теллес-Альварес <jackson.david.tellez@upc.edu>*
[3] *Х.М. Санчес <berotza@berotza.org>*
[1] *Dept. Física, UPC BarcelonaTech, Barcelona, Spain*
[2] *Dept. of Civil and Environmental Engineering (Institute Flumen), UPC Barcelona, Barcelona, Spain*
[3] *Berotza S.L., Noain, Pamplona, Navarra, Spain*

Аннотация. Локальная диффузия и топологическая структура завихрения и поля скоростей измерены в переходном режиме от гомогенной линейно стратифицированной жидкости до ячеистой или многоуровневой структуры посредством конвективного охлаждения и/или нагревания. Подобные структуры возникают, создавая конвективный поток, при использовании массива термоэлектрических устройств (элементы Peltier/Seebeck), которые генерируют значительный тепловой поток. Описанные в статье эксперименты направлены на изучение различных чисел Прандтля с использованием пластовой и пресной воды, чтобы сформировать градиенты плотности и низкие числа Прандтля для режимов смешения с градиентами температуры. Набор безразмерных параметров определяет условия численнего и мелкомасштабного лабораторного моделирования для различных геофизических течений. Поля скорости, плотности и их градиенты были вычислены и визуализированы с использованием открытого программного пакета DigiFlow и численного моделирования на базе уравнений Рейнольдса с k-e моделью турбулетности. Когда конвективный нагрев и охлаждение происходят на стороне стратифицированной вложенной ячейки (комбинации внутренних волн и плавучести) управляемая турбулентность намного более сложна, если числа Релея и числа Рейнольдса вылики. Вычисления моментов высшего порядка и перемежаемость важны для изучения процессов перемешивания в сложных течениях. В данной работе представлены некоторые примеры с использованием Thermoelectric Convection Didactive Device (TCDD), созданного в BEROTZA, главным образом в симметричного двумерного образца. Но существует много других технических вариантов устройства, например, с использованием охлаждения и нагревания, создания стенок под углом с вертикалью, позволяющих протестировать более сложные варианты с применением численных экспериментов.

Ключевые слова: конвекция; термоэлектричество; элементы Пелетье; эксперименты, k-e модель турбулентности; турбулетность; программа DigiFlow.

DOI: 10.15514/ISPRAS-2017-29(2)-8

Для цитирования: Редондо Х.М., Теллес-Альварес Дж.Д., Санчес Х.М. Турбулентная конвекция термоэлектричеством в охладительно-нагревательном устройстве. Труды ИСП РАН, том 29, вып. 2, 2017. pp. 215-230 (на английском). DOI: 10.15514/ISPRAS-2017-29(2)-8

References

[1]. Sekula E., Redondo J.M. The structure of Turbulent Jets, Vortices and Boundary Layers. Il Nuovo Cimento. 2008. V. 31. P. 893 – 907.

[2]. Redondo J.M. Introduction to Vortex Theory in Turbulence Laboratory Visualization, UPC. Lecture Notes Dept. Fisica. PELNHoT, 2015. P. 160

[3]. Redondo J.M., Garriga J.N. Convection driven by thermoelectric heat fluxes. Berotza, Report UPC, 1995. P 98.

[4]. Stepanova E.V. Study of the rotating fluid surface form in the cylindrical container // Selected papers of international conference “Fluxes and structures in fluids”. Moscow. 2006. P. 313–319.

[5]. Dalziel, S.D., Redondo J.M. New visualization and self-similar analysis in experimental turbulence studies. Models, Experiments and Computation in Turbulence. CIMNE, Barcelona.

[6]. Kuramitsu, M., Redondo, J.M., Noriega, G. Measurements of anysotropy, thermoelectric Proceedings of the IEEE 2003, V. 22, P. 541-545.

[7]. Castilla R., Onate E., Redondo J.M. Models, Experiments and Computations in Turbulence. CIMNE, Barcelona. 2007. P. 255.

[8]. Redondo, J. M., Sanchez, M. A., Garriga, J., Castilla, R., Convective and Rayleigh-Taylor Instabilities in Stratified Fluids. Advances in Turbulence V. 1995. P. 428-434.

[9]. Vila T., Tellez J., Sánchez M. J., Sotillos L., Diez M. and Redondo J.M. Diffusion in fractal wakes and convective thermoelectric flows. GRA– EGU. 2014.V.16, p.2011.

[10]. Tarquis, A. M., Platonov A., Matulka A., Grau J., Sekula E., Diez M., Redondo J. M. Application of multifractal analysis to the study of SAR features and oil spills on the ocean surface. Nonlinear Processes in Geophysics 21, no. 2 2014: P. 439-450.

[11]. Redondo J.M., Turbulence Entropy and Dynamics, UPC Lecture Notes. 2014.

[12]. Redondo, J.M., The structure of density interfaces. Ph.D. Thesis. Univ. of Cambridge. Cambridge. DAMTP 1990.

[13]. Redondo J.M., Metais O. Mixing in Geophysical Flows. CIMNE, Barcelona. 1995, p. 415.

[14]. Linden P.F., Redondo. J.M. Turbulent mixing in Geophysical Flows CIMNE, Barcelona, 2001. p. 319.

[15]. Cantalapiedra I.R., Redondo J.M. Mixing in zero-mean-flow turbulence, Mixing in Geophysical Flows (Redondo J.M. and Metais O. Eds.) CIMNE, 1995, 127-146.

[16]. Versteeg T.A.M., Nieuwstadt F.T.M. Numerical simulation of buoyancy driven flow in enclosures. Mixing in Geophysical Flows (Redondo J.M. and Metais O. Eds.) CIMNE, 1995, 329-342

[17]. Redondo J.M. Mixing efficiency of different kinds of turbulent processes and instabilities; Appplications to the environment. CIMNE, 2001, 131-157.

[18]. Redondo J.M. Termodinámica de los procesos irreversibles, efectos termoeléctricos. Rev. Termoelectricidad, 1995 V.2.

[19]. Mahjoub O.B. Granata T., Redondo J.M. Scaling Laws in Geophysical Flows. Phys. Chem. Earth (B) V 26, no.4 p281-285, 2001.

[20]. Dalziel R. P. DigiFlow User Guide. Cambridge, England: Department of Applied Mathematics and Theoretical Physics (DAMTP) - University of Cambridge. 2012.

[21]. Tellez J., Gómez. M., Russo M., Redondo, J.M. Surface Flow Image Velocimetry (SFIV) for Hydraulics Applications In 18th International Symposium on the Application of Laser and Imaging Techniques to Fluid Mechanics. July 4-7 2016. Lisbon, Portiugal.

[22]. Leyton C., Redondo J.M, Gonzalez-Nieto P.L. and Tarquis A.M. Fractal behaviour of human fluxes.Waves and vortices in complex media, Moscow, IPM RAS. 2016. V.2 p.230-235.

[23]. Strijhak S. Redondo J.M. And Tellez J. Multifractal Analysis of a Wake for a Single Wind Turbine Proceedings Topical Problems of Fluid Mechanics 2017, Prague, 2017 Edited by David Šimurda and Tomáš Bodnár, pp. 275-284.

Математическая формализация задач проектного планирования в расширенной постановке

[1]А.С. Аничкин <anton.anichkin@ispras.ru>
[1,2]В.А. Семенов <sem@ispras.ru>
[1]Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2]Московский физико-технический институт,
141700, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. Задачи теории расписаний и проектного планирования находят широкое применение в научных и индустриальных областях. В статье обсуждаются возможности обобщенной математической постановки задач проектного планирования и их эффективного решения эвристическими алгоритмами полиномиальной сложности.

Ключевые слова: теория расписаний; проектное планирование.

DOI: 10.15514/ISPRAS-2017-29(2)-9

Для цитирования: Аничкин А.С., Семенов В.А. Математическая формализация задач проектного планирования в расширенной постановке. Труды ИСП РАН, том 29, вып. 2, 2017 г., стр. 231-256. DOI: 10.15514/ISPRAS-2017-29(2)-9

1. Введение

Теория расписаний и методы проектного планирования находят широкое применение в научных и индустриальных областях, связанных с управлением производством, организацией транспортных потоков, управлением вычислительными ресурсами. Однако многообразие существующих и перманентное появление новых математических моделей приводят к необходимости обобщения классов прикладных задач и применения универсальных подходов к их решению. Необходимость подобных обобщений возникает, в частности, при создании перспективных систем календарно-сетевого планирования индустриальных проектов, в которых задачи составления расписаний решаются в расширенной постановке с учетом разнообразных факторов, влияющих на ход выполнения проектных работ. В

такой постановке учитываются не только типовые отношения предшествования между работами и простые ресурсные ограничения, но и директивные сроки, рабочие календари, условия финансового и логистического обеспечения проектных работ, специфические требования их пространственно-временной согласованности. Примерами подобных требований могут служить особенности монтажа элементов конструкций возводимого сооружения, условия резервирование рабочих зон при организации проектных работ, правила размещения и использования оборудования. Учет всех перечисленных выше факторов крайне важен для масштабных индустриальных программ, в которых риски организационных и технологических ошибок чрезвычайно высоки, а сроки и бюджеты жестко ограничены.

Задачи теории расписаний обычно описываются общепринятой нотацией Грэхема $\alpha|\beta|\gamma$, которая представляет собой комбинацию трёх основных характеристик, определяющих тип условий задачи. Кратко поясним ее, используя привычные для теории расписаний термины «задание», «операция» и «машина». В проектном планировании для этих понятий применяются термины «составная работа», «работа» и «ресурс».

Первая характеристика α описывает модель операций и модель машин (модель исполнения работ и модель ресурсов для проектного планирования). Предопределенные значения задают, например, постановки «рабочего цеха», «потоковой линии», «открытой линии». Дополнительный параметр обычно определяет число машин. В ресурсном планировании он применяется также для уточнения общего количества доступных ресурсов и их типа как возобновимых, невозобновимых, частично возобновимых, логистических или непрерывно разделяемых.

Вторая характеристика β описывает модель исполнения операций. Она обычно задается одним или несколькими значениями, которые определяют допустимость прерываний, наличие отношений предшествования, задание директивных сроков на начало и завершение операций, а также возможность пакетирования операций.

Наконец, третья обязательная характеристика γ определяет целевую функцию, минимум которой должен достигаться на составленном расписании. При поиске допустимого расписания целевая функция может использоваться для оценки качества найденного приближенного решения. Типовые значения γ соответствуют целевой функции для минимизации времени завершения всех операций, наибольшей задержки завершения операции, общего взвешенного времени завершения операций, общего взвешенного запаздывания операций и общих взвешенных опережений и запаздываний операций. Выбор целевых функций в задачах проектного планирования существенно шире, поскольку оптимизационная задача может ставиться как задача выравнивания используемых ресурсов или как задача минимизации общей или приведенной стоимости проекта. Допустим выбор целевых функций, направленных на достижение многокритериальных показателей проекта.

Примечательно, что задачи проектного планирования RCPSP (Resource-Constrained Project Scheduling Problem) в значительной степени обобщают постановки теории расписаний и поэтому алгоритмы проектного планирования часто рассматриваются в качестве универсальных инструментов для их решения. Тем не менее, в зависимости от индивидуальных характеристик задачи вычислительная сложность составления расписания может существенно варьироваться и поэтому для частных постановок обычно применяют специальные алгоритмы. Например, задача проектного планирования RCPSP, являющаяся NP-полной, редуцируется к частным постановкам «открытой линии», «рабочего цеха» или «потоковой линии», имеющим полиномиальную сложность при небольшом числе машин, простых моделях обслуживания и отсутствии директивных сроков.

Вместе с тем, на практике возникает необходимость в обобщении задач проектного планирования, обусловленная особенностями осуществления проектной деятельности разными организациями и индустриальными сообществами, различиями в представлении проектных данных, а также альтернативными способами математической формализации задач проектного планирования. Большое число существующих программных приложений для календарно-сетевого планирования и управления проектами с близкими функциями подтверждает этот факт. В качестве таких приложений следует указать популярные программные системы Oracle Primavera, MS Project, Synchro, Spider Project, Gemini, Merlin, Zoho Projects, ManagePro, обеспечивающие автономную работу планировщиков и управленческого персонала на изолированных компьютерах или групповую работу в корпоративной сети. Ряд систем конфигурируется в виде универсальных Интернет-сервисов и web-клиентов к ним. К подобным решениям относятся сервисы Smartsheet, GanttPro, Asana, Acunote, Teamweek, Bitrix24, Jira, ISETIA. Примечательно, что почти все приложения реализуют алгоритмы для поиска критического пути или вычисления критических работ, а большая часть приложений — алгоритмы решения задач RCPSP в расширенных постановках. В типовых приложениях речь идет о проектных планах, содержащих десятки тысяч работ, поэтому для решения задач RCPSP применяются приближенные, основанные на эвристиках алгоритмы полиномиальной сложности. Применение точных методов при составлении оптимальных расписаний для проектных планов с числом работ выше сотни не представляется возможным из-за экстремально высокой вычислительной сложности задач RCPSP.

В представленной работе предпринимается попытка математического обобщения и формализации задач проектного планирования с учетом особенностей расширенных постановок RCPSP, встречаемых в программных приложениях. Ожидается, что полученные результаты позволят разработать программно-инструментальную среду общего назначения, обеспечивающую задание условий и эффективное решение разнообразных классов задач проектного планирования. Развитые инструментальные возможности среды

позволят относительно просто адаптировать ее к новым постановкам прикладных задач.

В разделе 2 приводится классическая постановка задач RCPSP, и кратко упоминаются алгоритмы их приближенного решения. Недостатки и ограничения математической постановки детально обсуждаются в разделе 3, в котором особое внимание уделяется вопросам структуризации проектного плана и моделям исполнения работ. Кратко обсуждаются вопросы математической формализации, связанные с заданием альтернативных целевых функций оптимизационной задачи и возможным переопределенным характером системы наложенных алгебраических ограничений. Раздел 4 посвящен математической формализации обобщенной задачи проектного планирования GCPSP (Generally Constrained Project Scheduling Problem), а также доказательству некоторых утверждений о достаточных условиях существовании решения. В разделе 5 описывается алгоритм приближенного решения задач GCPSP, который можно рассматривать в качестве развития известного алгоритма последовательной диспетчеризации. Формулируется утверждение об эквивалентности алгоритмов в случаях, когда обобщенная задача редуцируется к классической постановке RCPSP. В заключении кратко резюмируются результаты и определяются возможные направления для дальнейших исследований.

2. Классическая постановка RCPSP

Рассмотрим классическую постановку задач RCPSP, следуя работам [[1], [2], [3]].

Пусть проект состоит из N работ, R возобновимых ресурсов и L связей с соответствующими индексами $i = 1,2,\dots,N$, $r = 1,2,\dots,R$ и $l = 1,2,\dots,L$ соответственно. Пусть задано время старта проекта t_0, а для каждой работы i определено ее время выполнения $d_i \geq 0$ и количества потребляемых ей ресурсов $q_{i,r} \geq 0$. Требуется найти расписание проекта (время старта каждой работы x_i, $i = 1,2,\dots,N$), при котором минимизируется время выполнения всего проекта.

Уточним условия задачи. Потребление работой i ресурса r в количестве $q_{i,r}$ означает, что с началом выполнения работы данное количество ресурса становится недоступным для использования в других работах, а с ее окончанием — высвобождается. Одна работа может потреблять несколько разных ресурсов. Один ресурс может одновременно использоваться несколькими работами при условии, что его суммарное потребление не превышает доступное количество $Q_r > 0$. Пусть $I(t,r) = \{i = 1,..,n \mid x_i \leq t < x_i + d_i \ \& \ q_{i,r} \neq 0\}$ — множество индексов работ, выполняемых в момент времени $t \geq t_0$ и использующих ресурс r. Тогда условие корректного потребления ресурса r и необходимое условие существования решения задачи может быть выражено следующим образом:

$$\sum_{i \in I(t,r)} q_{i,r} \le Q_r$$

Наконец, связи определяют бинарные отношения предшествования между работами таким образом, что работа-последователь не может стартовать до того, как работа-предшественник завершится. Пусть $i_{(l)}$ — индекс работы-предшественника, участвующего в связи l, а $i^{(l)}$ — индекс работы-последователя, участвующего в связи l. Тогда данное отношение имеет вид $x_{i_{(l)}} + d_{i_{(l)}} \le x_{i^{(l)}}$. В дальнейшем будем записывать $i \to j$, если существует связь l, такая что $i = i_{(l)}$ и $j = i^{(l)}$.

Тогда задача RCPSP формализуется следующим образом:

$$\min \left(\max_{i=1,\dots N} (x_i + d_i) \right)$$

$$\text{при } x_i \ge t_0, i = 1, \dots, N$$

$$x_{i_{(l)}} + d_{i_{(l)}} \le x_{i^{(l)}}, l = 1,2, \dots, L$$

$$\forall\, t \ge t_0 \text{ имеет место } \sum_{i \in I(t,r)} q_{i,r} \le Q_r, r = 1,2, \dots, R$$

Неизвестные переменные в постановке задачи обычно целочисленные, поскольку на практике обычно используется дискретное представление временных параметров. Задача считается корректно поставленной, если заданные связи не образуют циклов и уровни потребления ресурсов индивидуальными работами не превышают их общедоступное количество. В противном случае система алгебраических ограничений может оказаться переопределенной, а задача — не иметь решений. В случае отсутствия ресурсных ограничений задача RCPSP естественным образом редуцируется к задаче поиска критического пути или вычисления критических работ.

На сегодняшний день существует несколько популярных алгоритмов приближенного решения задач RCPSP. Наиболее известным является последовательный алгоритм составления расписания, иногда называемый алгоритмом последовательной диспетчеризации [[3], [4]]. Он позволяет за фиксированное количество шагов построить согласованное расписание или убедиться в его отсутствии, если в проектном плане есть неразрешимые ограничения. На каждом шаге алгоритма выбирается одна из работ, предшественники которой уже обработаны и для них определены времена старта и завершения. Для выбранной на основе эвристических правил работы вычисляется наиболее ранее допустимое время старта и осуществляется переход к следующей работе. В предположении отсутствии циклов последовательность работ с обработанными предшественниками всегда существует, а для работ всегда может быть определено время старта, согласованное со всеми наложенными ограничениями.

В классическом варианте алгоритм предполагает использование множества обработанных работ S и множества необработанных работ D, все предшественники которых входят в S [[1]]. На каждом шаге алгоритма множества корректируются в соответствии с приведенным ниже псевдокодом:

Begin
$S \coloneqq \emptyset$;
While $|S| < N$ **do**
$D \coloneqq \{i = 1,..,N | i \notin S \,\&\, j \in S \;\forall\, j \to i\}$;
For $r \coloneqq 1$ **to** R **do**
$K_r \coloneqq \left\{Q_r - \sum_{i \in I(\tau,r)} q_{i,r} \,\middle|\, \tau = t_0, t_0 + 1, \dots, T\right\}$;
End for;
$i^* \coloneqq \mathrm{argmax}_{i \in D}\{v(i)\}$;
If $\nexists\, j \to i^*$ **then**
$t_{min} \coloneqq t_0$;
Else
$t_{min} \coloneqq \max_{j \to i^*}(x_j + d_j)$;
End if;
$x_{i^*} \coloneqq \min\{t | t \geq t_{min} \,\&\, q_{i,r} \leq K_r(\tau), \tau = t, t+1, \dots, t + d_{i^*}, r = 1,2, \dots, R\}$;
$S \coloneqq S \cup \{i^*\}$;
End while;
End;

Здесь T — время, на протяжении которого составляется расписание, K_r — вспомогательное множество значений доступного ресурса в дискретные моменты времени, $v(i)$ — эвристическое правило, определяющее приоритет и очередность планирования работы i.

Алгоритм относительно прост и допускает многочисленные варианты, что делает его привлекательным для программной реализации. Один из вариантов заключается в попытке составить расписание в предположении отсутствия ресурсных ограничений и определить ранние и поздние даты выполнения работ, как это делается при поиске критического пути [[4]]. Затем проводится анализ ресурсных ограничений с учетом допустимых задержек индивидуальных работ. Можно ожидать, что для большей части работ ресурсные ограничения разрешатся таким способом. При этом следует предусмотреть возможность перезапустить или продолжить процесс составления расписания для оставшихся конфликтных работ и их последователей. Иначе применение алгоритма на практике может оказаться довольно рискованным. Другой известный вариант алгоритма основан на выделении групп несвязанных между собой работ и на их одновременном анализе [[1]]. Было установлено, что в ряде случаев он приводит к более качественным приближенным решениям. Однако реализация данного варианта сопряжена с дополнительными расходами на перераспределение работ по группам и согласование частных расписаний, что может оказаться критичным для больших проектных планов.

Все упомянутые выше алгоритмы имеют полиномиальную сложность и обеспечивают поиск приближенных решений для индустриально значимых задач, размерность которых может составлять десятки и сотни тысяч

неизвестных переменных. На практике часто требуется удостовериться в качестве получаемых решений, для чего желательно получить оптимальные решения, хотя бы для подобных задач низкой размерности. В этом случае можно применить точные комбинаторные алгоритмы с известными оптимизационными приемами типа «альфа-бета-отсечения» или «метода ветвей и границ» [[5]]. Например, лучший из известных точных алгоритмов Брукера за приемлемое время может решать задачи размерности не больше 60 [[6]]. По результатам сравнения приближенных и точных решений можно скорректировать параметры применяемых алгоритмов и настроить их эвристики с тем, чтобы обеспечить надлежащее качество результатов и при решении задач высокой размерности. К сожалению, теоретические исследования дают слишком грубые оценки, не позволяющие судить о качестве приближенных решений, полученных известных алгоритмами.

3. Ограничения классической постановки

Хотя приведенная математическая постановка RCPSP является классической для теории расписаний и прикладных дисциплин, связанных с календарно-сетевым планированием и управлением проектной деятельностью, она имеет ряд принципиальных ограничений для широкого практического использования. На необходимость решения задач проектного планирования в расширенной постановке указывают авторы работы [[3]]. В нашей обзорной статье [[7]] достаточно детально рассмотрены особенности прикладных задач, не учитываемые классической постановкой. В настоящем разделе мы неформально определяем обобщенный класс задач проектного планирования, который мог бы допускать альтернативные критерии и сложные модели исполнения работ, наложения связей, привлечения ресурсов, календарей, финансов при составлении расписаний. В конечном итоге подобные расширения приводят к новым видам целевых функций и функциональных ограничений для решаемых оптимизационных задач.

Опишем главные отличия, характеризующие обсуждаемый класс задач, более систематическим образом.

Во-первых, в нем допускается задание работ различных видов, таких как простые активности, вехи начала и завершения работ, «короткие гамаки», «длинные гамаки», а также составные работы, обеспечивающие иерархическую многоуровневую структуризацию проектного плана. При этом предполагается, что работы могут исполняться в обычном режиме, в режиме с прерываниями, с профильным использованием ресурсов, с учетом накладных временных затрат, в альтернативных мультимодальных режимах, а также с учетом компромиссов. При задании условий задачи может указываться также статус работ, как планируемых, стартованных, прерванных, возобновленных или завершенных, а также процент выполнения для незавершенных работ.

Во-вторых, взаимосвязи работ могут устанавливать отношения предшествования не только между окончанием предшествующей работы и

началом последующей работы, но также и между любыми событиями, связанными с их началом и завершением. Для формируемых взаимосвязей «начало-начало», «начало-окончание», «окончание-начало» и «окончание-окончание» могут быть установлены минимальные и максимальные величины задержки, пересчитанные в календарные даты с использованием рабочих календарей проекта. Примечательно, что величины задержек могут отрицательными, а взаимосвязи могут устанавливаться и для составных работ, обеспечивая тем самым возможность составления расписаний при крупноблочной структуризации проектного плана и при его последующей детализации. Важным аспектом составления расписания для актуализируемого проектного плана является учет нарушенных взаимосвязей и возможность восстановить отношения предшествования, хотя бы для оставшихся частей незавершенных работ. Подобные ситуации приводят к неоднозначной интерпретации ограничений и нуждаются в более строгой формализации.

В-третьих, исполнение работ и привлечение ресурсов в рамках проектной деятельности обычно осуществляется на основе календарей, определяющие графики работы. В обсуждаемых задачах предполагается использование модели календарей, в которой рабочие интервалы могут задаваться на основе регулярных и исключительных правил с требуемой точностью и дискретизацией представления временных параметров. При фиксировании трудоемкости работы и доминирующего ресурса ее продолжительность может определяться доступным количеством ресурса, а не только рабочим календарем.

В-четвертых, для работ могут быть заданы явные временные ограничения, определяющие алгебраические условия их начала и завершения. Ограничения могут задаваться условиями типа «начать не раньше», «завершить не позже», «начать в фиксированную дату» и т.п., так и спецификаторами «выполнить как можно раньше» или «выполнить как можно позже». Кроме того, каждой работе могут быть приписаны правила выравнивания, устанавливающие условия начала или завершения работы строго в начале или конце рабочего интервала (часа, дня, недели, месяца, года и т.п.) независимо от возможности исполнить работу несколько раньше или позже. Важной особенностью обсуждаемых задач является возможность задания временных ограничений для составных работ, что, как и в случае с взаимосвязями, обеспечивает возможность составления расписаний при крупноблочной структуризации проектного плана. Наконец, система наложенных алгебраических ограничений может оказаться переопределенной и не иметь решений. Это означает, что должен быть задан непротиворечивый способ разрешения подобных ситуаций, например, с использованием приоритетов, приписанных индивидуальным ограничениям.

В-пятых, ресурсная модель должна допускать использование невозобновимых, ограничено-возобновимых, частично возобновимых, логистических, непрерывно разделяемых, исключительных ресурсов, а также ресурсов с переменной доступностью. Возобновимые ресурсы обычно

моделируют использование персонала и техники, которые могут объединяться в «бригады». Ресурсы, связанные с персоналом определенной специализации и квалификации, могут формировать соответствующие «компетенции». Невозобновимые ресурсы обычно ассоциируют с расходными материалами, для которых могут определяться цепочки поставок. Финансовое обеспечение проектной деятельности также может моделироваться невозобновимыми ресурсами, ассоциируемыми, например, с банковскими счетами участвующих в проекте организаций. Количество доступных ресурсов может быть нефиксированным и зависеть от времени, например, как в случае поставок материалов или привлечения дополнительных инвестиций в ходе реализации проекта.

В-шестых, допускается выбор альтернативных целевых функций для минимизации временных показателей проекта, обеспечения консервативности и устойчивости расписания к задержкам, минимизации затрат на возобновимые ресурсы, минимизации невозобновимых ресурсов, минимизации общей стоимости проекта, максимизации чистой приведенной стоимости, а также достижения многокритериальных показателей проекта.

В-седьмых, задача проектного планирования может решаться в инкрементальной постановке, когда требуется скорректировать расписание с учетом измененных условий исходной задачи или при актуализации данных непосредственно на проектной площадке в режиме реального времени.

4. Математическая формализация задач GCPSP

Обсудим предлагаемый способ математической формализации задач проектного планирования. Следуя ему, оптимизационная задача ставится на множестве допустимых решений, связанных с частной задачей удовлетворения ограничений с приоритетами (или предпочтениями). В отличие от RCPSP мы не уточняем семантику неизвестных переменных, например, как времен начала, завершения, прерывания и возобновления работ или их продолжительностей. Также не конкретизируем вид целевой функции и функциональных ограничений, например, в виде явных временных или ресурсных условий. Вместо этого определяем обобщенный класс задач GCPSP в математически нейтральной форме в рамках общих предположений относительно свойств целевой функции, вида алгебраических ограничений и способа их разрешения относительно тех или иных переменных. Таким способом определяемый класс задач GCPSP расширяет условия классической постановки RCPSP и удовлетворяет основным требованиям к прикладным постановкам, перечисленным в предыдущем разделе.

Определение 1. Четверку множеств $\langle D, X, C, P\rangle$ назовем задачей в ограничениях с приоритетами, где $D = \{D_1 \times D_2 \times \ldots \times D_n\}$ — домен, определяющий область допустимых значений множества переменных задачи $X = \{x_1, x_2, \ldots, x_n\} \in D,\ x_i \in D_i,\ i = 1,2, \ldots, n;\ C = \{c_1(X_1), c_2(X_2), \ldots, c_m(X_m)\}$ — множество предикатов $c_j(X_j),\ j = 1,2, \ldots, m,$ определенных на

подмножествах переменных $X_j = \left\{x_{i_1^{(j)}}, x_{i_2^{(j)}}, \dots, x_{i_{k_j}^{(j)}}\right\} \subseteq X$ и называемых ограничениями задачи; $P = \{p_1, p_2, \dots, p_m\}$ — множество натуральных чисел $p_j \in N$, называемых приоритетами ограничений. Пусть $I(X_j) = \{i_1^{(j)}, i_2^{(j)}, \dots, i_{k_j}^{(j)}\} \subseteq \{1,2,\dots,n\}$ — множество индексов подмножества переменных X_j. Тогда будем говорить, что переменная задачи x_i, $1 \le i \le n$ участвует в ограничении $c_j(X_j)$, $1 \le j \le m$ или ограничение ассоциировано с переменной, если $i \in I(X_j)$. Ограничение $c_j(X_j)$ будем называть удовлетворённым при любом множестве значений переменных $X_j^* \subseteq X^* \in D$, при котором соответствующий предикат принимает значение «верно».

Определение 2. Пусть $\langle D, X, C, P\rangle$ — задача в ограничениях с приоритетами. Значения переменных $X^* \in D$, при которых все ограничения задачи удовлетворены, будем называть решением задачи. Пусть $C^*(X)$ — множество ограничений, которые удовлетворены при значениях переменных X. Тогда X^* является решением задачи, только если $C^*(X^*) = C$.

Определение 3. Пусть $\langle D, X, C, P\rangle$ — задача в ограничениях с приоритетами. Значения переменных $X^* \in D$ будем называть согласованным решением задачи в ограничениях с приоритетами, если, по крайней мере, хотя бы одно ограничение удовлетворено и не существует других значений переменных Y, повышающих максимальный приоритет разрешенных ограничений: $C^*(X^*) \neq \emptyset$ и для любого неразрешенного ограничения $c_j \in C \backslash C^*(X^*)$, $p_j \in P$ не существует значений $Y = \{y_1, y_2, \dots, y_n\} \in D$, $Y \neq X^*$, таких что удовлетворяется каждое ограничение $c_k \in C^*(Y)$, $p_k \in P$ с приоритетом $p_k \ge p_j$.

Определение 4. Пусть $\langle D, X, C, P\rangle$ — задача в ограничениях с приоритетами. Значения переменных $X^* \in D$ будем называть локально согласованным решением задачи в ограничениях с приоритетами, если, по крайней мере, хотя бы одно ограничение удовлетворено и согласованы приоритеты разрешенных ограничений относительно каждой переменной: $C^*(X^*) \neq \emptyset$, а также для любой переменной $x_i \in X$ и для любого ассоциированного с ней неудовлетворенного ограничения $c_j(X_j) \in C \backslash C^*(X^*)$, $p_j \in P$, $i \in I(X_j)$ не существует значения переменной $y_i \in D_i$, $y_i \neq x_i$ такого, что удовлетворяется каждое ассоциированное с переменной ограничение $c_k \in C^*(Y)$, $p_k \in P$, $i \in I(X_k)$ с приоритетом $p_k \ge p_j$, где множество значений переменных $Y = \{x_1^*, x_2^*, \dots, y_i, \dots, x_n^*\}$.

Определение 5. Обобщенной задачей проектного планирования GCPSP будем называть задачу оптимизации целевой функции проекта на множестве локально согласованных расписаний. Для определенности ограничимся следующей математической постановкой

$$\min f(X)$$

$$X \in D^*$$

где $f(X): Z^n \to R$ — целевая функция, $D^* \subseteq Z^n$ — множество локально согласованных решений системы ограничений $c_j(X_j)$, $X_j = \left\{x_{i_1^{(j)}}, x_{i_2^{(j)}}, \dots, x_{i_{k_j}^{(j)}}\right\} \subseteq X$ с приоритетами $p_j \in N, j = 1,2,\dots,m$.

Предполагается, что ограничения разрешимы относительно переменных одним из нижеприведенных способов:

(1) $c_j(X_j) \Leftrightarrow x_{i_1^{(j)}} = g_j\left(x_{i_2^{(j)}}, x_{i_3^{(j)}}, \dots, x_{i_{k_j}^{(j)}}\right)$ (ограничения A)

(2) $c_j(X_j) \Leftrightarrow x_{i_1^{(j)}} \in D_j^+\left(x_{i_2^{(j)}}, x_{i_3^{(j)}}, \dots, x_{i_{k_j}^{(j)}}\right)$ (ограничения B)

(3) $c_j(X_j) \Leftrightarrow x_{i_2^{(j)}} \in D_j^+\left(x_{i_1^{(j)}}\right) \parallel x_{i_3^{(j)}} \in D_j^+\left(x_{i_1^{(j)}}\right) \parallel \dots \parallel x_{i_{k_j}^{(j)}} \in D_j^+\left(x_{i_1^{(j)}}\right)$ (ограничения C)

(4) $c_j(X_j) \Leftrightarrow$ для любого упорядочивания переменных ограничения X_j, представимого биекцией множеств индексов $\pi^{(j)}: \{1,2,\dots,k_j\} \to \left\{i_1^{(j)}, i_2^{(j)}, \dots, i_{k_j}^{(j)}\right\}$, имеет место $x_{\pi_1^{(j)}} \in D^+_{\pi^{(j)},1}$, $x_{\pi_2^{(j)}} \in D^+_{\pi^{(j)},2}\left(x_{\pi_1^{(j)}}\right)$, $x_{\pi_3^{(j)}} \in D^+_{\pi^{(j)},3}\left(x_{\pi_1^{(j)}}, x_{\pi_2^{(j)}}\right)$, … , $x_{\pi_{k_j}^{(j)}} \in D^+_{\pi^{(j)},k_j}\left(x_{\pi_1^{(j)}}, x_{\pi_2^{(j)}}, \dots, x_{\pi_{k_j-1}^{(j)}}\right)$ (ограничения D)

(5) $c_j(X_j) \Leftrightarrow x_{i_1^{(j)}} \in D_j^-\left(x_{i_2^{(j)}}, \dots, x_{i_{k_j}^{(j)}}\right)$ (ограничения E)

где $D_j^+ \in Z$, $D^+_{\pi^{(j)},k} \in Z$ — подмножества целых чисел, содержащие открытые положительные интервалы, а $D_j^- \in Z$ — подмножества целых чисел, содержащие открытые отрицательные интервалы.

Естественным визуальным представлением задачи GCPSP является двудольный направленный граф $G\langle X, C, R\rangle$, где X — множество вершин, ассоциированных с переменными задачи, C – множество вершин, ассоциированных с ограничениями задачи, и R — множество ребер графа, соединяющих вершины ограничений с соответствующими вершинами переменных следующим образом. Если переменная задачи x_i участвует в ограничении c_j в качестве независимой переменной, то ребро является входящим в вершину ограничения (в применяемой нотации $x_i \to c_j$). Если переменная задачи x_i является зависимой переменной ограничения c_j, то ребро входит в вершину переменной ($c_j \to x_i$). Если ограничение c_j предусматривает правила разрешения относительно каждой своей переменной x_i, то ребра между соответствующими вершинами будем представлять как

двунаправленные и записывать $x_i \leftrightarrow c_j$. Будем также говорить, что переменная x_{i_1} прямо предшествует переменной x_{i_2} или $x_{i_1} \to x_{i_2}$, если существует такое ограничение задачи c_j, что $x_{i_1} \to c_j \to x_{i_2}$.

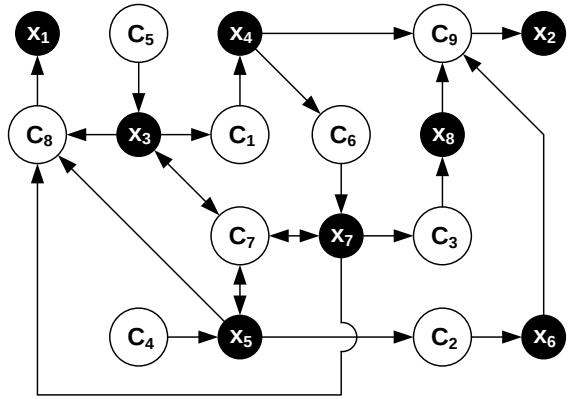


Рис. 1. Пример графа задачи GCPSP, отображающего взаимосвязи между переменными и ограничениями
Fig. 1. An example of the graph of the GCPSP task representing relationships between variables and constraints

На Рис. 1 представлен пример графа задачи GCPSP, на котором темными кружками отображены переменные задачи, а большими светлыми кругами — ограничения. Однонаправленные и двунаправленные ребра указывают характер зависимостей между переменными, который связан со способами разрешения ограничений.

В предположении, что граф задачи ацикличен, план разрешения системы ограничений представляется в виде перестановки переменных $\pi: \{1,2,\dots,n\} \to \{1,2,\dots,n\}$, в соответствии с которой они могут быть последовательно выражены друг через друга или уточнена область их допустимых значений в соответствии с наложенными ограничениями. Циклом в данном случае называется упорядоченный набор пар вершин $x_{i_1} \to c_{j_1} \to x_{i_2} \to c_{j_2}, \dots, x_{i_k} \to c_{j_k} \to x_{i_1} \subseteq G\langle X, C, R\rangle$, такой, что вершины-переменные и вершины-ограничения соединены последовательно направленными ребрами от вершины x_{i_l} к вершине c_{j_l} для всех $l = 1,..,k$, от вершины c_{j_l} к вершине $x_{i_{l+1}}$ для всех $l = 1,..,k-1$, и от вершины c_{j_k} к вершине x_{i_1}. Заметим, что двунаправленные ребра не включаются в цикл. План разрешения системы ограничений визуально будем отображать на графе задачи, приписывая индексы упорядочивания переменных соответствующим вершинам.

Сформулируем достаточные условия разрешимости системы ограничений с приоритетами и существования решения для связанной с ней обобщенной задачи проектного планирования GCPSP.

Утверждение 1. Пусть $\langle D, X, C, P\rangle$ — обобщенная задача GCPSP. Решение задачи всегда существует, если

(1) целевая функция задачи $f(X)$ монотонно неубывающая по каждой переменной;

(2) граф задачи $G\langle X, C, R\rangle$ ацикличен;

(3) ограничения (A) имеют наивысший приоритет, а приоритеты ограничений (B), (C) и (D) выше приоритета ограничений (E);

(4) ни одна переменная задачи не ассоциирована с более чем одним ограничением (A) в качестве зависимой переменной;

(5) каждая переменная задачи участвует, по крайней мере, в одном ограничении (A), (B), (C) или (D) в качестве зависимой переменной.

Доказательство.

Прежде всего, покажем, что система ограничений задачи разрешима и множество допустимых значений для переменных задачи не пусто. Допустимость значений будем интерпретировать в терминах локальной согласованности переменных в соответствии с назначенными приоритетами ограничений. В предположении ацикличности графа задачи существует упорядочивание переменных, при котором допустимые значения или область допустимых значений для переменных с большими индексами выражаются через соответствующие значения переменных с меньшими индексами. Покажем, что при обходе переменных по возрастанию индексов всегда существует непустое множество допустимых значений переменной, удовлетворяющее ограничения с входящими ребрами в вершину-переменную. Рассмотрим следующие взаимоисключающие случаи для определения области допустимых значений переменной:

- У вершины-переменной отсутствуют входящие ребра. В этом случае область допустимых значений переменной – все множество целых чисел.
- Имеется входящее ребро от унарного ограничения (*A*). В этом случае переменная принимает единственное допустимое значение в соответствии с явной функцией разрешения ограничения, независимо от других наложенных ограничений (*B*), (*C*) и (*D*), которые имеют меньший приоритет. Наличие нескольких ограничений (*A*) условиями утверждения исключаются.
- Имеются входящие ребра от ограничений (*B*) и (*C*) и отсутствует входящее ребро от ограничений (*A*). Независимо от назначенных приоритетов для переменной задачи всегда существует положительный полуинтервал, удовлетворяющий данным ограничениям. Ограничения (*D*) имеют меньший приоритет и поэтому

либо нарушаются, либо удовлетворяются путем уточнения допустимой области переменной в виде интервала или точки.

- Входящие ребра только от ограничений (D) не могут существовать в силу принятых допущений.

Таким образом, множество локально согласованных решений системы ограничений с приоритетами не пусто, причем они ограничены снизу. В силу неубывания целевой функции задачи по каждой из переменной ее минимум достигается на данном множестве и обобщенная задача проектного планирования GCPSP имеет решение. ■

Следующее утверждение определяет достаточные условия разрешимости задач проектного планирования GCPSP в случае, когда приоритеты назначаются ограничениям индивидуально независимо от их общего вида.

Утверждение 2. Пусть $\langle D, X, C, P\rangle$ — обобщенная задача GCPSP. Решение задачи всегда существует, если

(1) целевая функция задачи $f(X)$ по каждой переменной не убывает на некотором положительном полуинтервале и не возрастает на некотором отрицательном полуинтервале;

(2) граф задачи $G\langle X, C, R\rangle$ ацикличен;

(3) для каждой переменной задачи и ограничений, в которых она участвует как зависимая переменная, наивысший приоритет имеют либо одно ограничение (A), либо несколько ограничений (B), (C) и (D), либо несколько ограничений (E).

Доказательство.

Система ограничений задачи разрешима в смысле локальной согласованности переменных задачи. В самом деле, для каждой зависимой переменной всегда могут быть разрешены ограничения с наивысшим приоритетом. В случае единственного ограничения (A) это очевидно. В случае нескольких ограничений (B), (C) и (D) всегда существует положительный полуинтервал, которому удовлетворяют данные ограничения. В случае нескольких ограничений (E) существует отрицательный полуинтервал, которому удовлетворяют ограничения данного вида. Выполнимость ограничений с низкими приоритетами не имеет никакого значения для множества допустимых решений. В силу свойств монотонности целевой функции по каждой переменной на данном множестве существует, по крайней мере, одно оптимальное решение. ■

5. Алгоритм приближенного решения задач GCPSP

Опишем алгоритм приближенного решения задач проектного планирования в постановке GCPSP. Его можно рассматривать в качестве обобщения известной схемы последовательной диспетчеризации работ.

На каждом шаге алгоритма $k = 1,2,\dots,n$ определяется одна из переменных задачи x_i, $i \in I(X)$. При этом выбор очередной переменной подчиняется следующей общей схеме. Пусть $I_{PASSIVE} \subseteq I(X)$ — множество индексов переменных, значения которых определены к началу шага k, а $I_{ACTIVE} \subseteq I(X)$ – множество индексов неопределенных переменных, все прямые предшественники которых содержатся в $I_{PASSIVE}$ или вовсе не имеют предшественников. Таким образом, имеют место следующие соотношения между рабочими множествами: $I_{PASSIVE} \cap I_{ACTIVE} = \emptyset$ и если $i \in I_{ACTIVE}$, то либо не существует $j \in I(X)$ такого что $x_j \to x_i$, либо для каждого $j \in I(X)$ такого что $x_j \to x_i$, имеет место $j \in I_{PASSIVE}$.

Тем самым, множество $I(X)\backslash(I_{PASSIVE} \cup I_{ACTIVE})$ включает в себя индексы тех переменных, которые не участвовали в анализе до текущего шага и должны быть определены на следующих шагах алгоритма.

На начальном шаге инициализируются вспомогательные множества $I_{PASSIVE} = \emptyset$ и $I_{ACTIVE} = \left\{ i \in I(X) \middle| \nexists\, j \in I(X), x_j \to x_i \right\}$. В силу условий описанного выше утверждения граф задачи ацикличен, множество переменных, не имеющих предшественников, не пусто и алгоритм корректно стартует. На каждом шаге алгоритма отбирается активная переменная x_i, $i \in I_{ACTIVE}$, определяется ее значение и корректируются рабочие множества индексов таким образом, что активная переменная перемещается из I_{ACTIVE} в $I_{PASSIVE}$, а множество I_{ACTIVE} пополняется новыми переменными, все предшественники которых уже находятся в $I_{PASSIVE}$. Поскольку коррекция рабочих множеств осуществляется на каждом шаге алгоритма для поиска пополняемых переменных достаточно проанализировать только предшественников последователей переменной x_i. Таким образом, при переходе к следующему шагу рабочие множества корректируются следующим образом:

$$I_{ACTIVE} = I_{ACTIVE}\backslash\{i\},$$
$$I_{PASSIVE} = I_{PASSIVE} \cup \{i\},$$
$$I_{ACTIVE} = I_{ACTIVE} \cup \{k \in I\backslash(I_{PASSIVE} \cup I_{ACTIVE}) \mid x_i \to x_k, \forall\, x_{i'} \to x_k, i' \in I_{PASSIVE}\}.$$

Выборка активной переменной из множества I_{ACTIVE} выполняется на основе эвристических правил вида $i' H\, i''$, где $i', i'' \in I_{ACTIVE}$, определяющих линейный порядок на множестве переменных и позволяющих на текущем шаге выбрать переменную, наиболее предпочтительную для достижения оптимума целевой функцией. Иногда правила задают частичный порядок на множестве переменных. В подобных случаях можно определить иерархическую стратегию $H = \{H_l\}$, $l = 1,2,\dots,L$, заключающуюся в последовательном применении элементарных правил H_l в ожидании того, что одно из них позволит вынести окончательный вердикт о предпочтении одной переменной над другой. Если ни одно из правил не позволяет установить это, то приоритетной может считаться та переменная, которая имеет наименьший индекс. Очевидно, что порядок применения элементарных правил в рамках иерархической стратегии может влиять на упорядочивание переменных и

качество приближенного решения задачи. Алгоритмом допускается широкий набор эвристических правил, применяемым в задачах RCPSP [[8]]. Ниже мы приводим математически эквивалентную интерпретацию некоторых популярных правил, сохраняя их оригинальные названия:

- LIS (Least Immediate Successors): выбрать переменную с наименьшим количеством непосредственных последователей;
- MIS (Most Immediate Successors): выбрать переменную с наибольшим количеством непосредственных последователей;
- MTS (Most Total Successors): выбрать переменную с наибольшим количеством всех последователей;
- LTS (Least Total Successors): выбрать переменную с наименьшим количеством всех последователей;
- LSC (Longest Successors Chain): выбрать переменную с наибольшим числом переменных в цепочках последователей;
- SSC (Shortest Successors Chain) выбрать переменную с наименьшим числом переменных в цепочках последователей;
- SPT (Shortest Process Time): выбрать переменную с наименьшей разностью $x_j - x_i$ среди всех пар $x_j \to x_i$, связанных ограничениями вида (A) и (B). Здесь x_i — активная переменная, значение которой определяется в предположении ее отбора, а x_j — ее непосредственный последователь, значение которого рассчитывается на основе правила разрешения соответствующего ограничения и выбора минимально допустимого значения;
- LPT (Longest Process Time): выбрать переменную с наибольшей разностью $x_j - x_i$ среди всех пар $x_j \to x_i$, связанных ограничениями вида (A) и (B).

Существуют и более сложные правила, применение которых предполагает решение вспомогательных задач, связанных с локальной оптимизацией целевой функции. Важно, чтобы применяемые эвристики были релевантны целевой функции.

Значение для выбранной активной переменной определяется, исходя из требования минимизации целевой функции при условии удовлетворения максимального количества ассоциированных с ней ограничений. В случаях, когда система ограничений является переопределенной, ограничения разрешаются последовательно в соответствии с заданными приоритетами. В предположениях описанного выше утверждения всегда существует способ обеспечить локальную согласованность решения, при которой в первую очередь разрешаются ограничения с высокими приоритетами, а затем, если возможно, ограничения с низкими приоритетами.

В самом деле, после построения множеств $I_{PASSIVE}$ и I_{ACTIVE} и выбора активной переменной x_i, $i \in I_{ACTIVE}$ все независимые переменные

ассоциированных с ней ограничений (*A*), (*B*), (*C*), (*D*) и (*E*) находятся в множестве $I_{PASSIVE}$ и поэтому их значения уже определены и могут быть использованы для расчета допустимых значений x_i.

В случае наложенного ограничения вида (*A*) переменная принимает единственное значение в соответствии с правилом разрешения ограничения $x_i = g_j\left(x_{i_2^{(j)}}, x_{i_3^{(j)}}, \dots, x_{i_{k_j}^{(j)}}\right)$, где $\left\{i_2^{(j)}, i_3^{(j)}, \dots, i_{k_j}^{(j)}\right\} \subseteq I_{PASSIVE}$, независимо от других наложенных ограничений.

В случае ограничений вида (*B*), (*C*), (*D*) для переменной x_i вначале определяется допустимое множество значений как пересечение множеств интервалов $D_j^+\left(x_{i_2^{(j)}}, x_{i_3^{(j)}}, \dots, x_{i_{k_j}^{(j)}}\right)$ для всех j таких, что $c_j \to x_i$. Пересечение не пусто, поскольку множества включают в себя положительные полуинтервалы. Значение переменной выбирается, исходя из требования минимизации целевой функции по данной переменной. В случае неубывания целевой функции точка минимума всегда существует, а если она не единственна, то выбирается наименьшее значение. Переменные задачи обычно семантически связаны с датами исполнения и сроками, которые предпочтительно сократить даже при достижении целевой функцией минимума. Таким образом, значение выбранной переменной в этом случае определяется следующим образом:

$$x_i^* = \operatorname{argmin} f(x_i),$$

$$x_i \in \bigcap_{c_j \to x_i} D_j^+\left(x_{i_2^{(j)}}, x_{i_3^{(j)}}, \dots, x_{i_{k_j}^{(j)}}\right)$$

Анализ ограничений (*C*) и (*D*) имеет свои особенности. Чтобы удовлетворить ограничение вида (*C*) достаточно выполнить соответствующее условие для одной зависимой переменной. Действительно, попытка распространить условие $x_{i^{(j)}} \in D_j^+\left(x_{i_1^{(j)}}\right)$ на все зависимые переменные может оказаться обременительной для поиска качественного приближенного решения. Поэтому алгоритм предусматривает иную схему. Ограничение разрешается только для самой последней обрабатываемой переменной ограничения и только в том случае, если оно не было автоматически удовлетворено в результате определения значений предыдущих переменных. С этой целью для каждого ограничения вида (*C*) рассчитывается и хранится статус его выполнения, а также ведется подсчет числа обработанных ограничений. Такая схема обеспечивает разумную стратегию удовлетворения ограничений и минимизации целевой функции в случаях, когда переменные связаны между собой множественными ограничениями.

Для активной переменной все ассоциированные с ней ограничения (*D*) удовлетворяются в рамках общей схемы, поскольку они могут быть

разрешены относительно любой переменной независимо от переменных множества $I(X)\backslash I_{PASSIVE}$, значения которых на текущем шаге алгоритма еще не определены.

Ниже приводится псевдокод описанного выше обобщённого алгоритма.

Begin
    $X \coloneqq \perp$;
    $I_{PASSIVE} \coloneqq \emptyset$;
    $I_{ACTIVE} \coloneqq \{i | i \in I, \nexists\, i' \in I, x_{i'} \to x_i\}$;
    While $|I_{ACTIVE}| > 0$ **do**
        $i \coloneqq selectVariable(I_{ACTIVE})$;
        $x_i \coloneqq computeVariable(i, X)$;
        $I_{PASSIVE} \coloneqq updatePassive(I_{PASSIVE}, i)$;
        $I_{ACTIVE} \coloneqq updateActive(I_{ACTIVE}, i)$;
    End while;
End;

Function $selectVariable(I_{ACTIVE})$
Begin
    $i \coloneqq I_{ACTIVE}[1]$;
    For $k \coloneqq 2$ **to** $|I_{ACTIVE}|$ **do**
        If $I_{ACTIVE}[k]$ H i **then**
            $i \coloneqq I_{ACTIVE}[k]$;
        End if;
    End for;
    Return i;
End;
Function $computeVariable(i, X)$
Begin
    $D^* \coloneqq Z$;
    $J \coloneqq \{j | c_j \to x_i \mid c_j \leftrightarrow x_i\}$;
    $J \coloneqq sort(J, P)$;
    For $j \coloneqq 1$ **to** $|J|$ **do**
        $D \coloneqq resolve(c_j, X_j, i)$;
        If $j = 1$ **then**
            If $D = \emptyset$ **then**
                «Ошибка в постановке задачи или решения не существует!»
                Stop;
            Else
                $D^* \coloneqq D$;
            End if;

Else
If $D^* \cap D \neq \emptyset$ **then**
$D^* \coloneqq D^* \cap D$;
Else
«Ограничение не может быть удовлетворено!»
End if;
End if;
End for;
Return $\operatorname{argmin}_{x_i \in D^*}\{f(x_i)\}$;
End;

Function $updatePassive(I_{PASSIVE}, i)$
Begin
Return $I_{PASSIVE} \cup \{i\}$;
End;

Function $updateActive(I_{ACTIVE}, i)$
Begin
Return $(I_{ACTIVE} \backslash \{i\}) \cup \{k \in I \backslash (I_{PASSIVE} \cup I_{ACTIVE}) \mid x_i \to x_k, \forall\, x_{i'} \to x_k, i' \in I_{PASSIVE}\}$;
End;

Вспомогательная функция $sort(J, P)$ сортирует ограничения J по убыванию приоритетов P. Перед вызовом функции множество J формируется из ограничений, в которых x_i участвует в виде зависимой переменной. Функция $resolve(c_j, X_j, i)$ разрешает ограничение c_j относительно i переменной при фиксированных значениях остальных переменных из множества X_j. Возвращаемый результат — множество допустимых значений переменной D^*. В силу сделанных допущений относительно вида ограничений, функция допускает обобщенную реализацию. Из множества D^* переменной x_i присваивается одно из значений, при котором достигается минимум целевой функции по данной переменной.

В наихудшем случае описанный алгоритм имеет квадратичную вычислительную сложность $O(n^2)$ от количества переменных задачи n, поскольку на каждом шаге алгоритма из множества активных переменных, мощность которого ограничена n, необходимо выбирать наиболее предпочтительную переменную путем последовательного применения эвристических правил. Данная полиномиальная оценка приемлема для решения прикладных задач высокой размерности.

Приведённый алгоритм может быть легко трансформирован для поиска точного решения. Для этого достаточно подменить функцию $selectVariable$ на аналогичную, в которой выбор приоритетной переменной из множества активных I_{ACTIVE} осуществляется на основе перебора вариантов с отсечением

по верхней оценке целевой функции задачи U_f. Псевдокод функции приводится ниже.

Function $selectVariable(I_{ACTIVE}, X, U_f)$
Begin
  If $U_f \neq \perp$ **and** $f(X) > U_f$ **then**
    Return $\perp$;
  Else
    If $|I_{ACTIVE}| = 1$ then
      Return $I_{ACTIVE}[1]$;
    Else
      $i_{BEST} := \perp$;
      $f_{BEST} := \perp$;
      For $i := 1$ **to** $|I_{ACTIVE}|$ **do**
        $X' := X$;
        $x'_i := computeVariable(i, X')$;
        $I'_{ACTIVE} := updateActive(I_{ACTIVE}, i)$;
        While $|I'_{ACTIVE}| > 0$ **do**
          $i' := selectVariable(I'_{ACTIVE}, X', f_{BEST})$;
          If $i' = \perp$ **then**
            Break while;
            Next i;
          Else
            $x'_{i'} := computeVariable(i', X')$;
            $I'_{ACTIVE} := updateActive(I'_{ACTIVE}, i')$;
          End if;
        End while;
        If $i_{BEST} = \perp$ **then**
          $i_{BEST} := i$;
          $f_{BEST} := f(X')$;
        Else
          $f := f(X')$;
          If $f < f_{BEST}$ **then**
            $i_{BEST} := i'$;
            $f_{BEST} := f$;
          End if;
        End if;
      End for;
    End if;
    Return i_{BEST};

End if;

End;

Таким образом, точное решение находится путём перебора всех последовательностей назначений переменных, при которых достигается минимальное значение целевой функции на множестве локально согласованных решений. При этом алгоритм реализует метод «ветвей и границ», при котором из анализа исключаются последовательности, заведомо приводящие к неоптимальным решениям.

Утверждение 3. Класс задач RCPSP принадлежат классу задач проектного планирования GCPSP. В предположениях классической постановки RCPSP существуют решения эквивалентных задач GCPSP. Обобщенный алгоритм проектного планирования сводится к алгоритму последовательной диспетчеризации работ в случае задач RCPSP.

Доказательство. Покажем, что любая задача RCPSP удовлетворяет условиям обобщенной постановки GCPSP. Выберем в качестве неизвестных переменных задачи целочисленные даты начала работ $x = (x_1, x_2, \dots x_n)$. Очевидно, что функция минимизации проектного времени $\min(\max_{i=1,\dots N}(x_i + d_i))$ удовлетворяет условиям задачи GCPSP. Условия начала проекта $x_i \geq t_0, i = 1, \dots, N$ выражаются ограничениями вида (B), в которых отсутствуют независимые переменные, а область допустимых значений зависимых переменных определяется положительными полуинтервалами. Отношения предшествования работ $x_{i_{(l)}} + d_{i_{(l)}} \leq x_{i^{(l)}},\ l = 1,2, \dots, L$ также могут быть представлены ограничениями вида (B), если в качестве зависимых переменных использовать даты начала работ-последователей. Наконец, ресурсные условия

$$\sum_{i \in I(k)} \theta(t, x_i, d_i)\, r_{ik} \leq R_k, k = 1,2, \dots, K$$

могут быть отнесены к ограничениям (D), поскольку последовательно разрешимы относительно каждой переменной при фиксированных значениях других.

В предположениях классической постановки задач RCPSP для эквивалентных задач GCPSP также существует решение. Для доказательства воспользуемся достаточными условиями утверждения 1. Во-первых, целевая функция задачи RCPSP монотонно не убывает по каждой переменной. Во-вторых, граф задачи GCPSP ацикличен при условии, что отношения предшествования не образуют циклов. В-третьих, порождаемые условиями RCPSP ограничения вида (B) всегда разрешимы. Порождаемые ограничения вида (D) разрешимы, если уровни потребления ресурсов индивидуальными работами не превышают их доступное количество. Наконец, поскольку в эквивалентной задаче отсутствуют ограничения (A) и (E), а условия старта проекта распространяются на все переменные задачи, то при любом назначении

приоритетов на индивидуальные ограничения (B) и (D) решение задачи GCPSP существует согласно утверждению 1.

Для доказательства сводимости обобщенного алгоритма в случае задачи RCPSP достаточно адресоваться к описанию алгоритма последовательной диспетчеризации [[3]], а также проинтерпретировать работы в терминах переменных, а отношения предшествования — в терминах ограничений и соответствующих правил разрешения задачи GCPSP. При использовании упомянутых выше эвристик LIS, MIS, MTS, LTS, LSC, SSC, связанных с количеством непосредственных или общих последователей-переменных, предпочтение будет отдаваться тем же активным работам, что и в алгоритме последовательной диспетчеризации с аналогичными эвристиками, но выраженными в терминах предшествования работ. ■

Вместе с тем, область практического применения рассмотренной обобщенной постановки GCPSP существенно шире, поскольку охватываются прикладные задачи проектного планирования в расширенных постановках с учетом альтернативных критериев составления расписаний, сложных моделей исполнения работ, многопараметрических взаимосвязей, особенностей привлечения ресурсов, применения календарных графиков, финансового и логистического обеспечения проектных работ. Для краткости остановимся на некоторых примерах, характерных для расширенных постановок.

При планировании обычно применяются составные работы, которые обеспечивают иерархическую многоуровневую структуризацию проектного плана. Даты начала составных работ x_i и их продолжительности d_i связаны с параметрами дочерних работ $j \prec i$ следующим образом:

$$x_i = \min_{j=1,\dots,N \mid j \prec i} (x_j)$$

$$d_i = \max_{j=1,\dots,N \mid j \prec i} (x_j + d_j - x_i)$$

Данные условия представимы ограничениями (A) в постановке GCPSP и порождают ацикличный подграф задачи, допускающий последовательное вычисление значений переменных, начиная с параметров дочерних работ и заканчивая параметрами составных работ. Вычисления естественным образом обобщаются на случай многоуровневых составных работ, для которых обход начинается с простых работ нижнего уровня и распространяются на работы верхних уровней.

Для представления вех в проектном плане можно использовать простые работы с нулевой продолжительностью. Более содержательным является моделирование «коротких гамаков» и «длинных гамаков», которые можно описать в постановке GCPSP с помощью соответствующих ограничений (A).

Пусть дата начала «короткого гамака» x_i и его продолжительность d_i, тогда

$$x_i = \max_{j=1,\dots,N \mid x_j \to x_i} (x_j + d_j)$$

$$d_i = \min_{j=1,\dots,N \mid x_i \to x_j} (x_j - x_i)$$

Для «длинного гамака» соотношения приобретают вид

$$x_i = \min_{j=1,\dots,N \mid x_j \to x_i} (x_j + d_j)$$
$$d_i = \max_{j=1,\dots,N \mid x_i \to x_j} (x_j - x_i)$$

При условии, что взаимосвязи «гамаков» не образуют циклов, данные ограничения в постановке GCPSP также приводят к ацикличному графу и могут быть последовательно разрешены.

Другим примером расширенной постановки задач проектного планирования могут служить отношения предшествования типа «начало-начало», «начало-окончание», «окончание-начало» и «окончание-окончание», для которых могут быть установлены величины задержки. Пусть $i_{(l)}$ — индекс работы-предшественника, участвующего в связи l, $i^{(l)}$ — индекс работы-последователя, участвующего в связи l, и Δt_l — задержка связи, для которой допустимы и отрицательные значения. Тогда отношения предшествования выражаются следующими ограничениями в соответствии с перечисленными выше типами:

$$x_{i^{(l)}} \geq x_{i_{(l)}} + \Delta t_l$$
$$x_{i^{(l)}} \geq x_{i_{(l)}} - d_{i^{(l)}} + \Delta t_l$$
$$x_{i^{(l)}} \geq x_{i_{(l)}} + d_{i_{(l)}} + \Delta t_l$$
$$x_{i^{(l)}} \geq x_{i_{(l)}} + d_{i_{(l)}} - d_{i^{(l)}} + \Delta t_l$$

В постановке GCPSP данные условия выражаются ограничениями вида (B).

Другим важным аспектом является учет рабочих календарей, в соответствии с которыми исполняются работы и привлекаются ресурсы. Продолжительности, трудозатраты, задержки обычно выражаются в единицах рабочего времени, которые должны быть пересчитаны в календарные даты. Например, если x_i — дата начала простой работы и d_i — ее продолжительность, то дата завершения определяется не выражением $x_i + d_i$, как в приведенных выше формулах, а функцией календаря как $g(x_i, d_i)$. Примечательно, что общий вид ограничений в рамках обобщенной постановки GCPSP при этом не меняется.

Характерной особенностью прикладных постановок является также задание алгебраических условий для дат начала и завершения работ. Данные условия типа «начать не раньше», «завершить не раньше» или «начать не позже», «завершить не позже» выражаются в постановке GCPSP ограничениями вида (B) и (E) соответственно:

$$x_i \geq t_i$$
$$x_i \geq t_i - d_i$$
$$x_i \leq t_i$$
$$x_i \leq t_i - d_i,$$

где t_i — соответствующие директивные сроки. Согласованность ограничений обычно не контролируется пользователем и они могут оказаться

переопределенными даже в случае одной проектной работы. Для разрешения подобных ситуаций ограничениям могут быть приписаны приоритеты, а выполнимость условий проинтерпретирована в терминах локальной согласованности. Принципиально, что обобщенная постановка GCPSP предусматривает и такую возможность.

Рассмотрим более интересный случай, когда временные условия задаются пользователем для дат начала и завершения составных работ. Поскольку сами параметры составных работ являются зависимыми переменными от соответствующих параметров дочерних работ, непосредственный анализ таких ограничений не возможен. Однако подобные условия могут быть переопределены эквивалентным образом. Например, если для составной работы i задано условие «начать не раньше» в виде $x_i \geq t_i$, то оно может быть переопределено для всех дочерних работ $j \prec i$ как $x_j \geq t_i$, поскольку $x_i = \min_{j=1,\dots,N \mid j \prec i}(x_j)$. Тем самым условие в постановке GCPSP представимо ограничениями вида (B). Если для составной работы i задано условие «завершить не раньше» в виде $x_i \geq t_i - d_i$, то оно переопределяется для дочерних работ $j \prec i$ как $\bigvee_{j \prec i} x_j \geq t_i - d_j$, поскольку $d_i = \max_{j=1,\dots,N \mid j \prec i}(x_j + d_j - x_i)$. Данное условие в постановке GCPSP представляется ограничением вида (C).

Наконец, в классической постановке RCPSP рассматриваются только возобновимые ресурсы с постоянным профилем использования. В расширенных постановках обычно участвуют модели возобновимых и невозобновимых ресурсов с переменными профилями использования и доступности. Ресурсные ограничения в подобных случаях могут быть описаны следующим образом:

$$\forall\, t \geq t_0 \text{ имеет место } \sum_{i \in I(t,r)} q_{i,r}(t, x_i, d_i) \leq Q_r(t), r = 1,2, \dots, R,$$

где монотонно неубывающая ограниченная функция $Q_r(t)$ определяет доступное количество ресурса r на момент времени t в предположении, что данный ресурс не расходовался. Ограниченная функция $q_{i,r}(t, x_i, d_i)$ задает профиль использования ресурса r задачей i на момент времени t, а множество $I(t,r)$ определяет индексы работ, использовавших или использующих ресурс r на момент времени t.

Пусть для возобновимых ресурсов $q_{i,r}(t, x_i, d_i) = 0$ при $t < x_i$ или $t \geq x_i + d_i$ и $q_{i,r} = \max_{x_i \leq t < x_i + d_i} \{q_{i,r}(t, x_i, d_i)\}$, а для невозобновимых ресурсов имеет место $q_{i,r}(t, x_i, d_i) = 0$ при $t < x_i$ и $q_{i,r}(t, x_i, d_i) = q_{i,r}$ при $t \geq x_i + d_i$, где $q_{i,r} = \max_{t \geq x_i} \{q_{i,r}(t, x_i, d_i)\}$. Тогда можно показать, что условие в постановке GCPSP представляется ограничением вида (D), если для невозобновимых ресурсов выполняется $\sum q_{i,r} \leq Q_r$, а для возобновимых ресурсов имеет место $q_{i,r} \leq Q_r$, где $Q_r = \max_{t \geq t_0} \{Q_r(t)\}$.

Рассмотренные примеры демонстрируют общность предложенной постановки GCPSP для разнообразных прикладных задач проектного планирования.

Заключение

В работе предложена, формализована и обоснована обобщенная математическая постановка задач проектного планирования GCPSP, сформулированы и доказаны достаточные условия разрешимости задач данного класса, а также описан эффективный приближенный алгоритм полиномиальной сложности. В рамках дальнейших исследований предполагается реализовать данный алгоритм в составе программно-инструментальной среды общего назначения, обеспечивающей задание условий и решение разнообразных прикладных задач проектного планирования.

Список литературы

[1]. Kolisch R. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. // European Journal of Operational Research, том 90, 1996 г., стр. 320-333.
[2]. Kolisch R., Sprecher A. PSPLIB - A project scheduling library. // European Journal of Operational Research, том 96, 1996 г., стр. 205-216.
[3]. Лазарев А. А., Гафаров Е. Р. Теория расписаний. Задачи и алгоритмы. // МГУ им. М. В. Ломоносова, Москва, 2011 г., 222 стр.
[4]. Kelley James E. Jr., Walker Morgan R. Critical-Path Planning and Scheduling. // Proceedings of the eastern joint computer conference, 1959 г., стр. 160-173.
[5]. Land A. H., Doig A. G. An automatic method of solving discrete programming problems. Econometrica, том 28, выпуск 3, 1960 г., стр. 497-520.
[6]. Brucker P., Knust S. Complex scheduling. Springer, Берлин, 2006 г., 342 стр.
[7]. Аничкин А. С., Семенов В. А. Современные модели и методы теории расписаний. Труды ИСП РАН, том 26, вып. 3, 2014 г., стр. 5-50, DOI: 10.15514/ISPRAS-2014-26(3)-1.
[8]. Kolisch R. Project Scheduling under Resource Constraints: Efficient Heuristics for Several Problem Classes. Springer, Берлин, 1995 г., 212 стр.

Mathematical formalization of project scheduling problems

[1]A.S. Anichkin <anton.anichkin@ispras.ru>
[1,2]V.A. Semenov <sem@ispras.ru>
[1]Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2]Moscow Institute of Physics and Technology (State University)
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia

Abstract. Theory of scheduling and project planning is widely applied in diverse scientific and industrial areas. To effectively solve application-specific problems it is necessary to take into account a lot of factors such as task execution models, precedence relationship between tasks, resource limitations, directive deadlines, working calendars, conditions for financial and logistics support of project tasks, specific spatio-temporal requirements et al. Therefore,

in practice there is a need to generalize the project scheduling problems and to consider their extended formulations. The paper provides a classical mathematical statement of RCPSP problems (Resource-Constrained Project Scheduling Problem) and a short description of the serial dispatching algorithm for their approximate solution. Shortcomings and limitations of the RCPSP statement are discussed and systemized in more details. The paper presents a mathematical formalization of project scheduling problems in extended definitioins taking into account numerous features of practical problems. The proposed statement of GCPSP problems (Generally Constrained Project Scheduling Problem) can be considered as an evolution of RCPSP problems. This statement implies a mathematically neutral specification of the optimization problem under algebraic constraints of the predefined sorts and priorities. Essentially, that the constraints are interpreted in terms of the local consistency that allows some violations in the case of overloaded algebraic systems. An effective algorithm for the approximate solution of GCPSP problems is also presented and explained by following to the classical serial algorithm. Moreover, the equivalence of the algorithms is proved for the cases when a solved GCPSP problem is reduced to the RCPSP. It is expected that the obtained results will allow developing a general-purpose software library for solving of diverse project scheduling problems.

Keywords: scheduling theory; project planning and scheduling.

DOI: 10.15514/ISPRAS-2017-29(2)-9

For citation: Anichkin A.S., Semenov V.A. Mathematical formalization of project scheduling problems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017. pp. 231-256 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-9

References

[1]. Kolisch R. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. // European Journal of Operational Research, vol. 90, 1996, pp. 320-333.
[2]. Kolisch R., Sprecher A. PSPLIB - A project scheduling library. // European Journal of Operational Research, vol. 96, 1996, pp. 205-216.
[3]. Lazarev A. A., Gafarov E. R. [Scheduling theory. Tasks and algorithms.] // Lomonosov Moscow State University, Moscow, 2011, 222 p. (in Russian).
[4]. Kelley James E. Jr., Walker Morgan R. Critical-Path Planning and Scheduling. // Proceedings of the eastern joint computer conference, 1959, pp. 160-173.
[5]. Land A. H., Doig A. G. An automatic method of solving discrete programming problems. Econometrica, vol. 28, issue 3, 1960, pp. 497-520.
[6]. Brucker P., Knust S. Complex scheduling. Springer, Berlin, 2006, 342 p.
[7]. Anichkin A. S., Semenov V. A. A survey of emerging models and methods of scheduling. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 3, 2014. pp. 5-50 (in Russian). DOI: 10.15514/ISPRAS-2014-26(3)-1.
[8]. Kolisch R. Project Scheduling under Resource Constraints: Efficient Heuristics for Several Problem Classes. Springer, Berlin, 1995, 212 p.