ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/pi.html E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 7
N 1
2016

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Information about the jornal is available online at: http://novtex.ru/pi.html, e-mail:prin@novtex.ru

УДК 004.051; 004.912 DOI: 10.17587/prin.7.3-13

В. А. Васенин, д-р физ.-мат. наук, проф., e-mail: vasenin@msu.ru,
В. А. Роганов, ст. науч. сотр., e-mail: var@msu.ru, НИИ Механики МГУ имени М. В. Ломоносова,
М. Д. Дзабраев, инженер, e-mail: dzabraew@gmail.com, НИИСИ РАН, г. Москва

Экспресс-анализ потоковых текстовых данных на предмет вхождения в них ключевых слов и фраз

Представлены результаты начального этапа исследований, в том числе предложенный авторами подход к решению задачи быстрого обнаружения ключевых слов и фраз в потоковых текстовых данных. Важной характеристикой при решении этой задачи является скорость работы программы-анализатора, которая должна обеспечивать обработку интенсивных потоков в режиме реального времени. Представлены примеры эффективных алгоритмов, решающих поставленную задачу.

Ключевые слова: *Deep Packet Inspection, DPI, анализатор текста, детерминированный конечный автомат, недетерминированный конечный автомат, совершенная хэш-функция, технология CUDA*

Введение

В настоящее время, когда активно развивают и применяют на практике методы и средства сбора и систематизации, поиска и обработки данных большого и сверхбольшого объема (Big Data), механизмы быстрого автоматического анализа текстовой информации выходят на первый план. Интерес к экспресс-анализу текстового содержимого, отражающего состояние различных отношений в обществе, проявляют государственные организации и ведомства на самом высоком уровне как в России, так и за ее пределами. Механизмы непрерывного анализа текстовых данных являются определяющими в системах мониторинга как Интернет-пространства, так и трафика частных сетей. Они вносят все более заметный вклад в решение многих задач национального хозяйственного комплекса, в обеспечение информационной безопасности как общества в целом, так и отдельной личности.

Экспресс-анализ данных имеет следующую специфику, благодаря которой он выделяется среди других методов анализа текстовой информации.

- Программная система экспресс-анализа должна обрабатывать высокоскоростные потоки данных в масштабе реального времени. По этой причине глубина такого анализа обычно ограничивается поиском вхождения в текстовые данные ключевых слов. Поскольку поиск фраз относительно легко сводится к пост-анализу результата поиска слов, во многих случаях непосредственно для высокопроизводительного решающего ядра системы такой задачи не ставится.
- Сам по себе экспресс-анализ должен быть незаметен для поставщиков и потребителей информации. В случае распознавания ключевого слова система должна только инициировать те или иные действия, ассоциированные с обнаруженным образцом.
- Набор ключевых слов и фраз, ранжированных по категориям (степень важности, реакция на обнаружение), может изменяться непосредственно во время работы анализатора.

Такая специфика создает технические трудности, обусловленные прежде всего тем обстоятельством, что алгоритмы, их программная поддержка и аппаратура, способные обслуживать современные высокоскоростные каналы связи, должны иметь достаточно высокую производительность. Чтобы обеспечить столь высокий уровень производительности, необходимо не только использовать параллельные вычисления, но и динамически конструировать оптимальные вычислительные ядра анализирующих систем по заданному набору образцов (шаблонов) для поиска. Очевидно, чем больше будет таких образцов, тем большее число операций в среднем потребуется выполнить для их надежного обнаружения. Отмеченные трудности сказываются как на возможностях алгоритмов, так и на стоимости современных аппаратно-программных средств для экспресс-анализа потоковых данных. Кроме того, программное обеспечение подобных алгоритмов далеко не всегда имеет открытый исходный код, что затрудняет их сертификацию и использование в целом ряде критически важных приложений.

В рамках исследования, результаты которого представлены далее, авторы пытаются свести воедино несколько хорошо зарекомендовавших себя приемов экспресс-анализа потоковых данных в целях получения прототипа программного средства, способного

взять на себя решение отмеченных выше задач. Вначале представим описания некоторых классических алгоритмов для поиска образцов, которые известны и применяются на практике. Далее опишем предлагаемый авторами метод, агрегирующий некоторые классические подходы к анализу потоковых текстовых данных, но более перспективный с точки зрения производительности и более практичный в части его параллельной реализации на современных многоядерных платформах и спецвычислителях.

Алгоритм Aho—Corasick

Алгоритм Aho—Corasick [1—7] применяют для поиска множества фиксированных шаблонов в тексте. Для этого поиска строится конечный автомат. Графически этот автомат представляет собой специальное "суффиксное" дерево (в англоязычной литературе его называют *trie*). При поиске шаблонов в соответствии со структурой этого дерева проводится проход по тексту. Если на некотором шаге алгоритм "дошел" до специально помеченной вершины, то это означает, что найдено вхождение одного из шаблонов в текст. Опишем более подробно структуру и построение суффиксного дерева. Согласно алгоритму Aho—Corasick суффиксное дерево состоит из корневой и дочерних вершин. Дочерние вершины раскрашены в два цвета: серый и синий. Вершины соединены ребрами и ссылками. Каждому ребру приписан некоторый символ алфавита.

Изначально дерево, именуемое *Trie*, состоит из одной вершины — корня. Шаблоны последовательно пополняют дерево следующим образом. Пусть P — непустой шаблон, имеющий вид $s_1...s_n$, где s_i — символы используемого алфавита, $n > 0$ — число символов в шаблоне. Процедура добавления шаблона P в дерево *Trie* рекурсивна. Она начинается с корня дерева, первого символа шаблона s_1 и осуществляется следующим образом.

Рассматривается текущая вершина дерева и исходящие из нее ребра. Если среди ребер не оказалось такого, которому приписан текущий символ s_i, то нужно добавить новую вершину u серого цвета и ребро, которое идет из текущей вершины в u. Добавленному ребру приписывается символ s_i. В случае если уже существует ребро с приписанным символом s_i, идущее из текущей вершины в некоторую вершину u, то ничего делать не нужно.

Далее, если $i = n$, то был обработан последний символ шаблона. Вершина u в этом случае перекрашивается в синий цвет и процедура добавления шаблона завершается. Иначе переходим в вершину u, и, рассматривая ее как текущую, переходим к добавлению очередного символа s_{i+1}. На рис. 1 (см. вторую сторону обложки) представлено суффиксное дерево, полученное в результате добавления набора шаблонов {*a, ab, bab, bc, bca, c, caa*}, где *a, b, c* — символы используемого алфавита.

Далее определим понятие **суффиксной ссылки**. Для этого сначала введем две функции — w и I, которые отображают вершины рассматриваемого дерева в слова.

Пусть r — корень дерева. Положим $I(r) = \varepsilon$ — пустое слово. Если некоторая вершина u отлична от r, тогда в u входит ровно одно ребро, которому приписан некоторый символ a. Положим $I(u) = a$.

Функцию w определим следующим образом. Пусть в вершину v идет путь через вершины $v_0 = r$, v_1, ..., $v_s = v$. Тогда $w(v) = I(v_1) ... I(v_s)$.

Суффиксная ссылка для произвольной вершины дерева v определяется следующим образом. Рассмотрим всевозможные разложения слова $w(v)$ в конкатенацию двух строк:

$$w(v) = s_1 + w(u_1)$$

$$...$$

$$w(v) = s_n + w(u_n),$$

где u_i — некоторые вершины дерева. Выберем самое длинное слово из $\{w(u_1), ..., w(u_n)\}$. Предположим, что это будет $w(u_i)$. Тогда суффиксная ссылка для вершины v — это ребро (v, u_i).

На рис. 2 (см. вторую сторону обложки) добавлены суффиксные ссылки для суффиксного дерева, изображенного на рис. 1.

Кроме суффиксных ссылок вводят так называемые **словарные суффиксные ссылки**. Словарные суффиксные ссылки строятся по следующему правилу. Рассмотрим произвольную вершину дерева v. Будем двигаться из вершины v по суффиксным ссылкам, пока это возможно. Если мы на некотором шаге оказались в синей вершине v_i, то словарной суффиксной ссылкой для вершины v будет ребро (v, v_i).

На рис. 3 (см. вторую сторону обложки) добавлены словарные суффиксные ссылки для суффиксного дерева, изображенного на рис. 2.

Алгоритм поиска интересующих нас шаблонов сводится к переходам по вершинам дерева и выглядит следующим образом.

Шаг 1. Встать на начало текста T и в корень суффиксного дерева. Если из корня дерева идет ребро, подписанное символом $T[0]$, проследовать по нему, иначе остаться в корне. В тексте следует перейти к следующему символу $T[1]$.

Шаг 2. Пусть v — текущая вершина дерева и $T[j]$ — текущий символ. Если из вершины v выходит ребро, подписанное символом $T[j]$, перейти по этому ребру, сделать текущим символ $T[j + 1]$. В противном случае текущей вершиной станет вершина, на которую указывает суффиксная ссылка, а текущая позиция в тексте не изменится. Переходы по суффиксным ссылкам осуществлять далее до тех пор, пока не произойдет переход по символу $T[j]$, или в процессе перехода не окажемся в корне дерева.

Шаг 3. При переходе в очередную вершину v поступить следующим образом. Если вершина v — синяя, найден шаблон $w(v)$. Далее, если из вершины v идет словарная суффиксная ссылка в вершину v_1, найден также шаблон $w(v_1)$. Продолжая такие переходы по словарным суффиксным ссылкам, пока это

возможно, находим все встретившиеся на текущей позиции текста шаблоны $w(v_i)$.

Алгоритм Commentz — Walter

Алгоритм Commentz—Walter [8] является модификацией алгоритма Boyer—Moore [9, 10]. Далее будет описан оригинальный алгоритм Boyer—Moore и метод, которым из него можно получить алгоритм Commentz—Walter.

Алгоритм Boyer—Moore применяют для поиска единственной подстроки в тексте. Для этого левый край шаблона накладывается на начало текста и затем проводится сравнение символов справа налево. Если шаблон полностью совпал с подстрокой в тексте, то искомое совпадение найдено. В противном случае вычисляют сдвиг, на который искомый шаблон можно сместить вправо, и продолжают базовую процедуру посимвольного сравнения справа налево.

Значение сдвига при несовпадении берут как максимум из двух величин — $\max(\delta_1, \delta_2)$. Значения δ_1 и δ_2 получают применением описываемых далее эвристик — эвристики стоп-символа и эвристики совпавшего суффикса.

Эвристика стоп-символа представляет собой следующее. Пусть шаблон сопоставляется с текстом справа налево, и в некоторый момент времени в тексте встречается символ *c*, на котором произошло первое расхождение. Тогда δ_1 есть сдвиг, на который надо сместить шаблон, чтобы как можно более правый символ шаблона совпал с обнаруженным несовпавшим символом *c* в тексте. В случае если символ *c* в шаблоне не встречается, полагаем δ_1 равной полной длине шаблона:

```
Строка:            ******к******
Шаблон:            колокол
Следующий шаг:       колокол
```

Смысл такой тактики состоит в том, чтобы пропустить максимальное число позиций в тексте, для которых совпадение с шаблоном заведомо невозможно. Однако одной этой эвристики, к сожалению, не достаточно. Далее приведен пример, где δ_1 становится отрицательным числом.

```
Строка:            ****ккол*****
Шаблон:             колокол
Следующий шаг:     колокол
```

Для быстрого вычисления δ_1 строится таблица, именуемая *StopTable*, в которой каждому символу алфавита ставится в соответствие число. Если некоторый символ не присутствует в шаблоне, то соответствующим ему числом будет ноль. Если символ присутствует в шаблоне, то число полагается равным позиции самого правого вхождения этого символа в шаблон (с учетом того, что нумерация символов шаблона начинается с единицы). Исключение составляет последний символ, для которого записывается его предпоследнее вхождение. Если же символ на последней позиции шаблона входит один раз, то записывается ноль.

Теперь, если на *i*-й позиции шаблона обнаружено первое несоответствие с текстом, то $\delta_1 = i - StopTable[i]$.

Эвристика совпавшего суффикса. Пусть, как и ранее, поиск шаблона в тексте происходит справа налево. Пусть на $(i-1)$-м символе шаблона встретилось первое расхождение между текстом и шаблоном. Выберем такой максимальный суффикс из суффиксов $\{v_i...v_n, v_{i+1}...v_n, \ldots, v_{n-1}...v_n\}$, где v_i — символ шаблона, который содержится в шаблоне как подстрока и не совпадает с самим собой по местоположению. Если для выбранного суффикса таких подстрок несколько, то выбираем самую правую. Сдвинем шаблон так, чтобы эта подстрока встала на место суффикса. Если такой максимальный суффикс шаблона начинается со смещения *s*, а подстрока начинается со смещения *d*, то искомый сдвиг δ_2 есть $s-d$. Таким образом рассматривают все суффиксы шаблона, включая пустой суффикс, и для них заполняют таблицу, именуемую *SuffixTable*, которая состоит из пар {суффикс, смещение}. Для пустого суффикса смещение равно 1. Имея такую таблицу, положим $\delta_2 = SuffixTable[suffix]$. На этом описание алгоритма Boyer—Moore можно считать завершенным.

Теперь нетрудно описать, как работает интересующий нас алгоритм Commentz—Walter, который осуществляет поиск в тексте сразу нескольких шаблонов.

По каждому из шаблонов в этом случае строится описанная выше пара таблиц (*StopTable, SuffixTable*), затем по ним строятся так называемые результирующие таблицы по следующему правилу: для каждого символа алфавита *c* из соответствующих величин *val* выбирается наименьшая. В результирующую таблицу *StopTable* попадает пара $(c, val_{\min})$.

Для результирующей таблицы *SuffixTable* проводится аналогичная операция. При этом если в нескольких таблицах окажутся одинаковые суффиксы, то берется минимальный сдвиг.

Вычисление δ_1 и δ_2 проводится по тем же формулам, что и в оригинальном алгоритме Boyer—Moore, но с использованием результирующих таблиц.

Далее строится специальное дерево, в чем-то похожее на суффиксное дерево из алгоритма Aho—Corasick. Сначала шаблоны для поиска инвертируются. Затем для перевернутых шаблонов строится дерево по тому же правилу, что и в алгоритме Aho—Corasick, только без суффиксных и словарных ссылок.

Поисковая часть алгоритма Commentz—Walter работает следующим образом. Пусть $l_{\min}$ — длина наиболее короткого шаблона. Начиная с корня дерева и символа текста с номером $l_{\min}$, будем "прогонять" текст через дерево, двигаясь по тексту справа налево. Если алгоритм дошел до вершины дерева, которой соответствует один из искомых шаблонов, то очередное вхождение данного шаблона в текст найдено. Если же у некоторой вершины в процессе прогона

не оказалось потомка, помеченного очередным символом, то из имеющихся таблиц нужно вычислить сдвиг для следующей позиции в просматриваемом тексте. После этого вновь начинаем с корня дерева и продолжаем движение так, как было описано выше.

Алгоритм Rabin—Karp

Алгоритм Rabin—Karp [11, 12] был разработан в 1987 г. Этот алгоритм осуществляет поиск вхождений набора шаблонов в исследуемый текст с использованием механизма хэширования. Работает этот алгоритм следующим образом.

Пусть имеется m шаблонов $subsP = \{P_1, ..., P_m\}$ с длинами $l_1 < ... < l_s$, и пусть имеется текст T, в котором требуется искать эти шаблоны. На множестве всех слов определим некоторую хэш-функцию $h: \Sigma^* \to Z$ и для всех заданных шаблонов построим множество $hashes = \{h(P_1), ..., h(P_m)\}$.

При поиске нужно идти по тексту и вычислять s чисел (s — мощность множества длин шаблонов) при помощи функции h. Находясь в j-й позиции текста T, вычисляют значения $h_1 = h(T[j - l_1 + 1, ..., j])$, ..., $h_s = h(T[j - l_s + 1, ..., j])$. После того как были вычислены все значения h_i, проводится проверка того, что h_i принадлежит множеству *hashes*. Очевидно, что такую проверку можно выполнить достаточно быстро.

В случае если h_i принадлежит множеству *hashes*, на основе числа h_i выполняется проверка на принадлежность подстроки $T[j - l_i + 1, ..., j]$ к *subsP*.

Быстродействие алгоритма может сильно изменяться в зависимости от скорости вычисления хэш-функции h и от того, как реализованы проверки принадлежности h_i к множеству *hashes* и принадлежности подстроки-кандидата к множеству шаблонов.

Псевдокод для этого алгоритма представлен на рис. 4.

В общем случае значения h_i на j-м шаге вычисляют без использования результатов вычислений, полученных на предыдущих шагах. Однако существуют такие хэш-функции, которые могут быть быстро вычислены с использованием их значений на предыдущем шаге: $h_i := rh(h_i, T[j])$. Если хэш-функция может вычисляться таким образом, то она относится к семейству **rolling-hash**. Известна оптимизация данного подхода — Rabin—Karp rolling hash [13].

Пусть $c_1, ..., c_k$ — последовательно идущие символы. Введем константу a, которая будет параметром хэш-функции. Rabin—Karp rolling hash-функция вычисляется как

$$H_1 = c_1 a^{k-1} + ... + c_k a^0.$$

Пересчет значения rolling hash-функции для очередного символа c_2 осуществляется следующим образом:

$$H_2 = (H_1 - c_1 a^{k-1})a + c_{k+1}.$$

Эта хэш-функция использует потенциально медленную операцию — умножение. Наиболее быстрым является умножение на степени 2. Известна также

```
subsP={P_1..P_n}
hashes={h(P_1),...,h(P_n)}
RK_search(T[0..N], subsP, hashes) {
  pre_search(T[0..l_s-2],subsP)
  for(j=l_s-1;j<=N;j++){
    h_1 = h( T[j-l_1+1,..,j] )
      ...
    h_s = h( T[j-l_s+1,..,j] )

    if (h_1 ∈ hashes)
      if (T[j-l_1+1,..,j] ∈ subsP)
        print j, T[j-l_1+1,..,j]
          ...
    if (h_s ∈ hashes)
      if (T[j-l_s+1,..,j] ∈ subsP)
        print j, T[j-l_s+1,..,j]
  }
}
```

Рис.4. Структура алгоритма Rabin—Karp

циклическая полиномиальная хэш-функция, которую иногда называют **buzhash** [11]. Эта функция имеет следующий вид:

$$H = 2^{k-1}d(c_1) + 2^{k-2}d(c_2) + ... + 2^1 d(c_{k-1}) + d(c_k),$$

где d — некоторая быстро вычислимая функция, а операция + есть поразрядное сложение по модулю 2 (т. е. операция XOR). Умножение на 2 программно реализуется как битовый сдвиг. Пересчет по очередному символу c_{k+1} проводится следующим образом:

$$H = 2H + 2^k d(c_1) + d(c_{k+1}).$$

Выбрав наиболее эффективную хэш-функцию, нужно также уметь быстро определять, принадлежит ли вычисленное значение хэш-функции множеству вычисленных ранее значений *hashes*, а также по значению хэш-функции быстро находить соответствующий ему шаблон. Если область значений хэш-функции не слишком большая, то можно под ее область значений выделить массив, в ячейках которого на местах h_i хранить адреса шаблонов P_i. Если же область значений выбранной хэш-функции слишком обширна, то можно, например, применить технику First Fit Decreasing Method, описание которой представлено в следующем разделе.

First Fit Decreasing Method

Метод, именуемый First Fit Decreasing Method [14], используется для решения следующей задачи. Пусть имеется отображение $key_i \to val_i$, где i и key_i принад-

лежат множествам целых чисел [1...n] и [1...N] соответственно, $N > n$, а val_i — объекты произвольной природы из множества *Val*. Требуется программно реализовать эффективную (с точки зрения быстродействия) **идентифицирующую функцию *f***, которая принимает на вход любое число x из отрезка [1, ..., N] и возвращает либо NULL, либо элемент из множества *Val*, причем $x = key_i => f(x) = val_i$.

Приведем описание шагов построения идентифицирующей функции по этому методу.

Шаг 1. Выбрать некоторое число t, которое удовлетворяет условию $N < t^2$.

Шаг 2. Поместить каждый объект val_i в изначально пустую матрицу размером $t \times t$ (данная матрица будет разреженная, поэтому хранить ее в памяти компьютера рекомендуется с учетом данной специфики). Позиция (x, y) для размещения объекта val_i определяется по формулам $y = key_i / t$ и $x = key_i \bmod t$

Шаг 3. Поочередно сдвигать строки матрицы, начиная с верхней, вправо до тех пор, пока хотя бы один из элементов сдвигаемой строки находится в одном столбце с элементами из предыдущих строк.

Шаг 4. После необходимых сдвигов в каждом столбце расширенной матрицы останется не более одного элемента. Как следствие, теперь можно спроецировать ее столбцы на линейный массив *C*. Если после сдвигов элемент оказался в позиции (i, j), где j — номер столбца, то его индексом в линейном массиве *C* будет ячейка с номером j.

Искомая идентифицирующая функция f использует число t, массив смещений из шага 3, и линейный массив из шага 4. Ее программная реализация имеет следующий вид:

```
y = K/t
x = K mod t
index = r[x] + y
f(K) = C[index]
```

Продемонстрируем построение идентифицирующей функции на примере. Пусть даны следующие множества ключей K и значений V:

K = {0, 3, 4, 7, 10, 13, 15, 18, 19, 21, 22, 24, 26, 29, 30, 34};

V = {v0, v3, v4, v7, v10, v13, v15, v18, v19, v21, v22, v24, v26, v29, v30, v34}.

Поскольку каждый элемент V_i должен быть помещен в квадратную матрицу **A**, то ее размер t должен быть не менее шести. В общем случае, как было указано выше, следует выбрать число t из соображений $N < t^2$. При $t = 6$ имеем следующее содержимое матрицы **A**:

A	0	1	2	3	4	5
0	v0	.	.	v3	v4	.
1	.	v7	.	.	v10	
2	.	v13	.	v15	.	.
3	v18	v19	.	v21	v22	.
4	v24	.	v26	.	.	v29
5	v30	.	.	.	v34	.

Теперь начнем сдвигать строки матрицы вправо, пока не останется не более одного элемента на каждый столбец. Будем записывать сдвиг каждой строки в массив сдвигов *r*. Эта процедура, собственно, и называется First Fit Method (FFM). Результат применения FFM к матрице **A** будет следующим:

```
r[0]=0   v0   .    .    v3   v4   .
r[1]=1        .    v7   .    .    v10  .
r[2]=5                          .    v13  .    v15  .    .
r[3]=9                                         v18  v19  .    v21  v22  .
r[4]=14                                                                  v24  .    v26  .    .    v29
r[5]=7                                    v30  .    .    .    v34  .
```

Осталось спроецировать сдвинутые строки на линейный массив *C:*

```
vals:   v0   .    v7   v3   v4   v10  v13  v30  v15  v18  v19  v34  v21  v22  v24   .   v26   .    .   v29
index:  0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19
```

Формальная постановка задачи и предлагаемый метод решения

Формальная постановка задачи, которую необходимо эффективно решить, состоит в следующем. Пусть даны алфавит Σ, шаблоны P_1, ..., P_m — конечные, непустые последовательности символов из алфавита Σ, и текст T — конечная последовательность символов из алфавита Σ. Требуется находить все вхождения шаблонов P_1, ..., P_m в тексте T за минимальное время.

Предлагаемый авторами в рамках настоящей публикации комбинированный алгоритм **hash404** базируется на идее алгоритма Rabin—Karp, однако выбрана другая rolling hash-функция и принадлежность ее значений к заданному множеству чисел определяется с применением First Fit Decreasing Method. Далее предлагается искать в тексте шаблон не целиком, а только некоторый префикс шаблона. Затем, в случае обнаружения префикса, необходимо проверять совпадение с оставшейся частью (префиксы каких длин при этом следует использовать — это очень

важный вопрос, разрешение которого сильно влияет на скорость поиска предлагаемой реализации алгоритма). Ниже приведено описание алгоритма hash404.

Процедуру вычисления используемых хэш-функций определим следующим образом.

1. Введем локальные переменные тела хэш-функции — h_0^k, ..., h_m^k, которые проинициализируем нулями.

2. Для каждого очередного символа c из анализируемого текста пересчет значений локальных переменных будем осуществлять согласно формуле

$$h_n^{k+1} = \left(\left(h_{n-1}^k \ll m_n\right) \text{XOR } c\right) \,\&\, mask,$$

где n принадлежит $[1...m]$,

$$\text{при } n = 0\text{: } h_0^{k+1} = c \,\&\, mask.$$

Здесь c — очередной символ, операция $\ll$ представляет собой битовый сдвиг; XOR — eXclusive OR; m_n и $mask$ — параметры, которые заранее подобраны так, чтобы хэш-функция была инъективной на множестве шаблонов. Это, как нетрудно показать, осуществимо для любого набора шаблонов, так как ничто не ограничивает в выборе количества и значений параметров хэш-функции. Один из возможных способов подбора нужных параметров будет разобран в следующем разделе.

Процесс вычисления значений хэш-функции при подаче на вход начального фрагмента текста "*abcdef*" представлен графически на рис. 5.

Как видно на рис. 5, при анализе текста значение каждой локальной переменной h^s зависит не более чем от $s + 1$ последних обработанных символов.

Чтобы использовать алгоритм Rabin—Karp нужно выбрать хэш-функцию и выбрать алгоритм для реализации операций принадлежности для чисел и подстрок (см. рис. 4). В качестве хэш-функции предлагается взять изложенную выше процедуру. Для реализации операций принадлежности рекомендуется использовать алгоритм First Fit Decreasing Method (FFDM), который был описан ранее.

Для определения того, что вычисленное значение хэш-функции принадлежит множеству *hashes* (см. рис. 4), нужно положить в алгоритме FFDM в качестве множеств K и V множество *hashes*.

Для реализации операции проверки принадлежности подстроки множеству шаблонов нужно взять *hashes* в качестве K и указатели на шаблоны — в качестве V. Тогда по вычисленному значению хэш-функции реализация алгоритма FFDM в случае успеха будет возвращать указатель, и остается только сравнить подстроку текста с тем шаблоном, который лежит по этому указателю.

Способы поиска инъективных хэш-функций

В данном разделе рассмотрим способы перебора определенных выше параметров m_n и $mask$ для поиска инъективных на множестве шаблонов хэш-функций. В качестве параметров в предлагаемом подходе используются сдвиги $\{m_1, ..., m_k\}$, где m_i принадлежит $[0, 8]$, и множество масок $M_1 = 2^1 - 1, ..., M_{32} = 2^{32} - 1$. Множество всех сдвигов разбивается на слои таким образом, что $\Sigma m_i = L$. Зададим для величины L некоторый диапазон значений и будем последовательно перебирать слой за слоем.

Разбиение множества параметров на слои удобно по следующей причине. Если сумма сдвигов больше показателя маски $mask = 2^j - 1$, то, как легко заметить, будет происходить потеря информации. Как показала практика, если Σm_i на 2 больше, чем показатель маски, а искомых ключевых шаблонов около нескольких тысяч, то инъективной хэш-функции скорее всего обнаружить не удастся. Как следствие, такое разбиение помогает отбросить заведомо малоперспективные варианты.

Перебор параметров можно осуществить по каждому слою L, где L принадлежит некоторому отрезку $[a, b]$. Затем из множества всех подходящих параметров ($\{m_1, ..., m_k\}$, $mask$) можно выбрать наиболее оптимальные варианты.

Опишем, как проверяют конкретные параметры на наличие инъективности. Для конкретного набора сдвигов $m = \{m_1, ..., m_k\}$ нужно перебрать все маски и вычислить минимальную маску, на которой будет иметь место инъективность хэш-функции.

Введем множество H, которое будет хранить в себе вычисленные значения хэш-функции, диапазон масок и текущую минимальную маску. У реализации множества H в виде класса языка C++ будут методы `init`, `add` и `getMinMask`. Метод `add` получает на вход слово, и по нему вычисляет хэш-функцию без применения маски, сохраняя вычисленное значение. Затем метод `add` проверяет, приводит ли применение

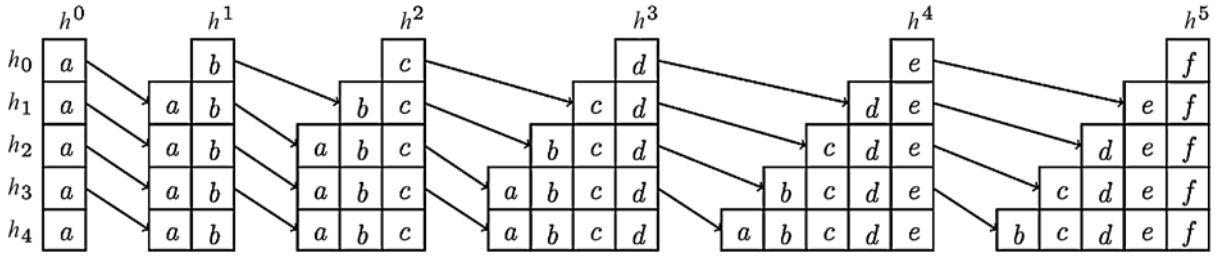

Рис. 5. Графическое представление вычисления значений хэш-функции

текущей маски к потере свойства инъективности. Если возникла коллизия, то значение маски изменяется с $2^k - 1$ на $2^{k+1} - 1$. Если ни одна маска из заданного диапазона не подошла, то хэш-функция с заданными параметрами $m = \{m_1, ..., m_k\}$ не является инъективной на множестве шаблонов.

Реализация вычислительного ядра для CPU

Авторами разработана программа-генератор, которая по заданному множеству шаблонов и оптимизированным параметрам инъективной хэш-функции генерирует код программы-анализатора на языке C++.

Вычислительное ядро программы-анализатора работает следующим образом.

Шаг 1. По каждому очередному символу вычисляется значение хэш-функции на основе значения, полученного на предыдущем шаге.

Шаг 2. Проверяется, содержится ли вычисленное значение в множестве *hashes*.

Шаг 3. В случае если ответ на последний вопрос положительный, извлекается смещение на префикс. Далее подстрока сравнивается с префиксом. Если выявилось совпадение, нужно сохранить позицию совпадения, чтобы в дальнейшем проверить суффиксы.

Наиболее значимые фрагменты кода программы-анализатора представлены далее.

```
 /* некоторые предварительные вычисления*/
 /* w — текст*/
 for(i = 3;i<textlen;i++){/*цикл по тексту*/
 hash404(h,w); /*вычисление хэш-функции на основе предыдущего значения хэш-функции*/
 /*проверка префикса длины три*/
 if(ffdm _ checkhash(h[2])){/*проверка на предмет
                        принадлежности значения хэш-функции к hashes*/
    unsigned offset3 = ffdm _ get _ offset(h[2]); /*смещение на префикс*/
    if(/*проверка совпадения символов подстроки и шаблона*/
       (w[0] == (words + offset3)[2]) &&
       (w[-1] == (words + offset3)[1]) &&
       (w[-2] == (words + offset3)[0]) &&
                !(words + offset3)[3]
    )

 {/*вхождение найдено, сохраняем позицию*/}
}

 /*префикс длины 7*/
 if(ffdm _ checkhash(h[6])) {
 unsigned offset7 = ffdm _ get _ offset(h[6]);
 if(
          (w[0]==(words + offset7)[6]) &&
          (w[-1]==(words + offset7)[5]) &&
          (w[-2]==(words + offset7)[4]) &&
          (w[-3]==(words + offset7)[3]) &&
          (w[-4]==(words + offset7)[2]) &&
          (w[-5]==(words + offset7)[1]) &&
          (w[-6]==(words + offset7)[0]) &&
                 !(words + offset7)[7]
    )
 {/* нашлось вхождение, сохраняем позицию*/}
 }
 w++;
}
```

Предлагаемая реализация хэш-функции hash404 представлена ниже.

```
  #define hash404(h,w) do{\
  h[6]=( (h[5] << hash_shifts[6] ) ^ (unsigned)( (unsigned char)(*w) ) )&mask;\
  h[5]=( (h[4] << hash_shifts[5] ) ^ (unsigned)( (unsigned char)(*w) ) );\
  h[4]=( (h[3] << hash_shifts[4] ) ^ (unsigned)( (unsigned char)(*w) ) );\
  h[3]=( (h[2] << hash_shifts[3] ) ^ (unsigned)( (unsigned char)(*w) ) );\
  h[2]=( (h[1] << hash_shifts[2] ) ^ (unsigned)( (unsigned char)(*w) ) );\
  h[1]=( (h[0] << hash_shifts[1] ) ^ (unsigned)( (unsigned char)(*w) ) );\
  h[0]=(                             (unsigned)( (unsigned char)(*w) ) );\
}while(0)
```

Измерения скорости работы анализаторов текста

В настоящем разделе представлены результаты измерения скорости обработки данных (далее — скорость) анализаторов, программные реализации которых построены на основе алгоритма Commentz—Walter, алгоритма поиска конечным автоматом (DFA), а также предлагаемого авторами алгоритма hash404. Измерения проводили на процессоре Intel Core i5.

Под скоростью здесь и далее понимаем отношение размера файла, Мбит, ко времени, за которое был проведен поиск шаблонов в этом файле. Скорость работы анализаторов измерялась на текстах четырех художественных произведений, написанных на английском языке: William Shakespeare "Romeo and Juliet", James Jones "From here to Eternity", Samuel Shem "Mount Mistery", Jerome Salinger "The Catcher in the Rye". Для каждого из анализаторов по результатам измерения скорости их работы на вышеперечисленных текстах были построены по четыре графика. Эти четыре графика на рисунках местами выглядят как одна жирная линия в силу близких значений скорости.

Алгоритмы Commentz—Walter и DFA реализованы в утилите GNU-grep [15].

На рис. 6 (см. третью сторону обложки) представлены результаты измерения скорости анализатора на базе алгоритма Commentz—Walter в ходе его работы на указанных выше художественных произведениях. На рис. 7 (см. третью сторону обложки) изображено содержание рис. 6, к которому добавили измерения реализации скорости работы алгоритма DFA на этих же произведениях. На рис. 8 (см. третью сторону обложки) изображено содержание рис. 7, к которому добавили результаты измерения скорости реализации анализатора на базе алгоритма, предлагаемого авторами.

Заметим, что у программной реализации алгоритма DFA наблюдается резкое уменьшение скорости в диапазоне 500...1100 шаблонов с 1,5...2 Гбит/с до 2...60 Мбит/с. Скорость предлагаемой авторами реализации при поиске 7000 шаблонов превосходит скорость реализации алгоритма Commentz—Walter примерно в 2 раза. Если увеличивать число шаблонов, то соотношение скоростей будет оставаться около 2. На 15454 шаблонах предлагаемый алгоритм демонстрирует скорости 1272, 1313, 1307, 1274 Мбит/с. На аналогичном числе шаблонов анализатора на основе алгоритма Commentz—Walter показал скорость 262, 502, 708 и 476 Мбит/с соответственно.

Как отмечалось ранее, целесообразным может стать поиск не шаблонов целиком, а некоторых префиксов от шаблонов. На рис. 8 (см. третью сторону обложки) изображена скорость предлагаемой реализации алгоритма, когда от шаблонов отделялись префиксы длин 3 и 5. Если длина шаблона больше или равна 5, тогда отделялся префикс длины 5. Если длина шаблона меньше 5, но больше 3, тогда отделялся префикс длины 3.

Ранее констатировалось, что выбор длин префиксов должен сильно влиять на скорость. На рис. 9 (см. четвертую сторону обложки) представлены результаты измерения скорости работы hash404, когда от шаблонов отделялись префиксы следующих длин: 3, 4; 3, 5; 3, 6, 9; 3, 6; 3, 7; 3, 8; 3, 9; 3. При числе шаблонов, меньшем 500, наивысшую скорость показал вариант с префиксами длины 3. На более чем 500 шаблонах наилучшие обнаруженные длины префиксов — это 3, 5. Отметим также, что до 500 шаблонов анализатор на базе алгоритма hash404 с префиксами 3 превосходит анализатор на базе DFA по скорости, что наглядно отображено на рис. 10, см. четвертую сторону обложки.

Следует, однако, отметить, что кроме программы GNU-grep существует также малоизвестная программа для поиска текстовых образцов CUDA-grep [16] для графических процессоров, поддерживающих технологию CUDA. Эта программа осуществляет поиск при помощи недетерминированного конечного автомата, который строится методом Томпсона.

К сожалению, реализация программы CUDA-grep оставляет желать лучшего. В частности, при использовании современной версии CUDA, оригинальный код этой программы оказался неработоспособным. Авторам удалось запустить и протестировать эту программу лишь после того, как в ней был исправлен ряд ошибок.

На рис. 11 отображено время поиска в секундах одного регулярного выражения при помощи средств GNU-grep и CUDA-grep.

Регулярное выражение	*Файл*							
	20000_leagues.txt x120		lua.lines.js.txt x40		passwd-perline-delboca.txt x400		romeojuliet.txt x400	
	CUDA-grep	GNU-grep	CUDA-grep	GNU-grep	CUDA-grep	GNU-grep	CUDA-grep	GNU-grep
"ROMEO"	**0,015**	**0,044**	**0,031**	**0,042**	**0,012**	**0,042**	**0,015**	**0,041**
"JULIET"	**0,015**	**0,036**	**0,031**	**0,046**	**0,012**	**0,047**	**0,015**	**0,036**
"ROMEO|JULIET"	**0,016**	**0,024**	0,033	0,025	**0,016**	**0,024**	**0,017**	**0,029**
"R+"	**0,015**	**0,049**	**0,031**	**0,073**	**0,013**	**0,075**	**0,014**	**0,052**
"R*"	**0,016**	**0,241**	**0,033**	**0,274**	**0,016**	**0,249**	**0,015**	**0,221**
"R"	**0,015**	**0,047**	**0,031**	**0,080**	**0,012**	**0,091**	**0,014**	**0,063**
"R+R*"	0,015	0,012	0,031	0,019	**0,013**	**0,015**	0,014	0,014
"R*R+"	0,016	0,015	0,033	0,022	0,017	0,013	0,016	0,008
"RR+"	**0,015**	**0,044**	**0,031**	**0,061**	**0,012**	**0,065**	**0,014**	**0,041**
"RR*"	**0,015**	**0,016**	0,031	0,023	**0,013**	**0,021**	**0,014**	**0,059**
"R+|J+"	**0,016**	**0,039**	**0,033**	**0,049**	**0,017**	**0,042**	**0,015**	**0,035**
"(R|J)ULIET"	**0,016**	**0,028**	0,034	0,031	**0,016**	**0,028**	**0,016**	**0,025**
"R..EO"	0,015	0,014	0,031	0,011	**0,013**	**0,017**	**0,015**	**0,035**
"R..EO|...IET"	0,017	0,013	0,039	0,008	0,022	0,012	0,018	0,010
"R..*"	**0,021**	**0,022**	0,032	0,023	**0,017**	**0,020**	**0,051**	**0,061**
"[a-b]"	**0,016**	**0,237**	**0,033**	**0,209**	**0,018**	**0,238**	**0,014**	**0,194**
"[q-s]"	**0,017**	**0,239**	**0,036**	**0,212**	**0,022**	**0,243**	**0,015**	**0,198**
"[0-9]"	**0,015**	**0,232**	**0,031**	**0,211**	**0,013**	**0,238**	**0,013**	**0,201**
"\?"	**0,015**	**0,044**	**0,031**	**0,081**	**0,012**	**0,095**	**0,013**	**0,061**
"|"	**0,015**	**0,247**	**0,031**	**0,274**	**0,012**	**0,250**	**0,013**	**0,209**
"\+"	**0,015**	**0,048**	**0,031**	**0,078**	**0,012**	**0,095**	**0,013**	**0,064**
"*"	**0,015**	**0,054**	**0,031**	**0,081**	**0,012**	**0,093**	**0,013**	**0,063**
"\."	**0,015**	**0,045**	**0,031**	**0,084**	**0,012**	**0,093**	**0,013**	**0,064**
"\.\?|\+*"	**0,015**	**0,242**	**0,031**	**0,274**	**0,012**	**0,257**	**0,013**	**0,214**
".*\?"	**0,113**	**0,277**	**0,250**	**0,314**	**0,070**	**0,378**	**0,092**	**0,247**
"t*hi.\?"	**0,016**	**0,208**	**0,033**	**0,044**	**0,021**	**0,097**	**0,014**	**0,132**
" *function .+\(\).*"	**0,015**	**0,063**	**0,031**	**0,049**	0,012	0,007	**0,013**	**0,041**
" *var +.+"	0,015	0,012	0,031	0,021	0,012	0,010	0,013	0,007
".*//.*"	0,016	0,010	0,035	0,021	0,016	0,009	**0,014**	**0,017**
".*[a-h]+\(.+\).*"	**0,004**	**0,013**	**0,002**	**0,051**	**0,002**	**0,011**	**0,003**	**0,007**

Рис. 11. Сравнение производительности GNU-grep и CUDA-grep

В левом столбце на рис. 11 отображены регулярные выражения. Поиск этих регулярных выражений проводился в четырех файлах (эти файлы входят в набор тестовых данных для утилиты CUDA-grep), которые указаны в заголовке таблицы. Измерялось время поиска каждого регулярного выражения в каждом файле для GNU-grep и CUDA-grep. У CUDA-grep измерялось время работы только вычислительного CUDA-ядра. Позиции, в которых утилита CUDA-grep работает быстрее, чем GNU-grep, выделены полужирным шрифтом.

Коллизии значений используемой хэш-функции

Для применения метода FFDM хэш-функция должна быть инъективной на множестве шаблонов. Как отмечалось ранее, вычислительное ядро предлагаемого алгоритма поиска реально работает не с шаблонами, а с префиксами шаблонов. Отсюда следует, что хэш-функция должна быть инъективной на множестве предварительно отобранных префиксов.

Если от всех шаблонов брать префикс длины 3, а битовые сдвиги, используемые при вычислении

хэш-функции, положить равными 8, то такая хэш-функция заведомо будет инъективной на выбранных префиксах, поскольку, как легко заметить, ее значение будет представлять собой число, представленное в двоичной записи первыми тремя байтами шаблона. Однако с такой хэш-функцией предлагаемый алгоритм будет работать медленно, что иллюстрирует рис. 12 (см. четвертую сторону обложки). Также на рис. 12 отображены измерения скорости программных реализаций для алгоритмов hash404 для префиксов 3,5 и алгоритма Commentz—Walter.

Для разрешения коллизий для значений хэш-функции на данном этапе предлагается брать от шаблонов префикс длины 3, и, возможно, рассматривать еще какие-либо варианты длины. Первые два сдвига m_1 и m_2 следует положить равными 8. Если же какой-либо префикс длины d шаблона P будет давать коллизию, тогда от шаблона P можно взять префикс длины 3, а не d. Таким образом нужно поступить со всеми шаблонами, которые дают коллизию, нарушающую свойство инъективности хэш-функции.

Заключение

В рамках первого этапа исследований, результаты которого представлены в настоящей публикации, была разработана модификация алгоритма Rabin—Karp. На нынешнем этапе исследований разработанная программная реализация состоит из двух программ. Первая программа — **оптимизатор** — для заданного множества шаблонов подбирает оптимальные параметры хэш-функции. Вторая программа — **генератор** — с использованием найденных оптимизатором параметров генерирует исходный код анализатора текста на языке C++.

К сожалению, авторам не удалось найти качественных реализаций оригинального алгоритма Rabin—Karp для измерения их производительности. В будущем планируется модифицировать предложенный генератор таким образом, чтобы он порождал также эффективный программный код и для оригинального алгоритма Rabin—Karp.

К перспективным направлениям для дальнейшего исследования следует отнести расширение функциональных возможностей генератора по генерации кода для многоядерных архитектур, включая платформу CUDA, а также разработку методов увеличения общего числа ключевых слов, для которых будут находиться оптимальные параметры хэш-функций. В результате экспериментов, проведенных в этом направлении, удалось убедиться, что результирующая скорость работы анализатора даже на одном современном многоядерном процессоре после распараллеливания и максимальной оптимизации при использовании алгоритмов, подобных предложенному в данной статье, может превышать значение 100 Гбит/с.

Список литературы

1. **Aho A. V., Corasick M. J.** Efficient string matching an aid to bibliographic search // Communications of the ACM. 1975. Vol. 18, N 6. P. 333—340, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.4671&rep=rep1&type = pdf.

2. **Dori S., Landau G. M.** Construction of Aho Corasick Automaton in Linear Time for Integer Alphabets // Lecture Notes in Computer Science. 2005. Vol. 3537. P. 168-177. URL: http://cs.haifa.ac.il/~landau/gadi/shiri.pdf.

3. **Визуализатор** алгоритма Aho—Corasick. URL: http://artlives.ru/ahocorasik.

4. **Описание** алгоритма Aho—Corasick. URL: http://habrahabr.ru/post/198682.

5. **Watson B. W**. Taxonomies and Toolkits of Regular Language Algorithms. URL: http://www.diku.dk/hjemmesider/ansatte/henglein/papers/watson1995.pdf.

6. **Смит Б.** Методы и алгоритмы вычислений на строках. Москва: Вильямс, 2006. 486 с. URL:http://scienceengineering.library.scilibgen.org/view.php?id=603002.

7. **Описание** алгоритма Aho—Corasick. URL: https://en.wikipedia.org/wiki/Aho%E2%80%93Corasick_algorithm.

8. **Commentz W.** A String Matching Algorithm Fast On Average // Proceedings of the 6th Colloquium, on Automata, Languages and Programming. 1979, P. 118—132.

9. **Boyer R. S., Moore J. S.** A Fast String Search Algorithm // Communications of the Association for Computing Machinery. 1977. Vol. 20, N 10. P. 762—772. URL: available at: https://www.cs.utexas.edu/~moore/publications/fstrpos.pdf.

10. **Описание** алгоритма Boyer—Moore. URL: https://ru.wikipedia.org/wiki/Алгоритм_Бойера_—_Мура.

11. **Karp R. M., Rabin M. O.** Efficient randomized pattern-matching algorithms // IBM Journal of Research and Development — Mathematics and computing archive. 1987. Vol. 31, N 2, P. 249—260. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.9502&rep=rep1&type=pdf.

12. **Описание** алгоритма Rabin—Karp. URL: https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm.

13. **Примеры** Rolling-hash функций. URL: https://en.wikipedia.org/wiki/Rolling_hash.

14. **Gettys T.** Generating Perfect Hash Functions; 2001. URL: http://www.drdobbs.com/architecture-and-design/generating-perfect-hash-functions/184404506.

15. **Домашняя** страница проекта GNU-grep. URL: http://www.gnu.org/software/grep.

16. **Описание** результатов проекта CUDA-grep. URL: http://www.cs.cmu.edu/afs/cs/academic/class/15418-s12/www/competition/bkase.github.com/CUDA-grep/finalreport.html.

Real-Time Analysis of Streaming Data for Presence of Keywords and Key Phrases

V. A. Vasenin, vasenin@msu.ru, **V. A. Roganov,** var@msu.ru, Moscow State University, 119234, Moscow, Russian Federation, **M. D. Dzabraev,** dzabraew@gmail.com, Scientific Research Institute for System, Analysis of the Russian Academy of Science, 117218, Moscow, Russian Federation

Corresponding author:

Vasenin Valery A., Professor, Moscow State University, 119234, Moscow, Russian Federation, e-mail: vasenin@msu.ru

Received on October 15, 2015
Accepted on November 5, 2015

In this article are presented the results of the first stage of research. Among them authors proposed the approach to the problem of detecting keywords and key phrases in streaming data. The key moment for solving this problem is the speed of computational kernel of analyzer, which must provide the real-time processing of intensive data streams. In the article are presented the examples of effective algorithms which solve the formulated problem. Most of them are used widely in many areas of data processing. Proposed approach is a kind of optimization of Rabin—Karp algorithm with use of special form of injective hash function and first fit decrease method (FFDM) combined. Additional special preprocessing of keyword list provides speed-up in two and more times comparing to widely used implementation of Commentz—Walter algorithm. The performance of existing CUDA-grep analyzer was evaluated too. The article contains benchmarks results for all considered analyzers. The parallel version of described algorithm is in a development stage. Simple experiment with such kind of algorithm showed that performance > 100 Gbit/s is reachable using single multi-core CPU. CUDA version of proposed analyzer is planned for implementation too.

Keywords: *deep packet inspection, DPI, text analyzer, deterministic finite state automata, nondeterministic finite state automata, perfect hash function, CUDA*

For citation:

Vasenin V. A., Roganov V. A., Dzabraev M. D. Real-Time Analysis of Streaming Data for Presence of Keywords and Key Phrases, *Programmnaya Ingeneria*, 2016, vol. 7, no. 1, pp. 3—13.

DOI: 10.17587/prin.7.3-13

References

1. **Aho A. V., Corasick M. J.** Efficient string matching an aid to bibliographic search, *Communications of the ACM*, 1975, vol. 18, no. 6, pp. 333—340, available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.4671&rep=rep1&type=pdf.

2. **Dori S., Landau G. M.** Construction of Aho Corasick Automaton in Linear Time for Integer Alphabets, *Lecture Notes in Computer Science*, 2005, vol. 3537, pp. 168—177, available at: http://cs.haifa.ac.il/~landau/gadi/shiri.pdf.

3. **Visualizer** of the Aho—Corasick algorithm, available at: http://artlives.ru/ahocorasik.

4. **Description** of the Aho—Corasick algorithm, available at: http://habrahabr.ru/post/198682 (in Russian).

5. **Watson B. W.** Taxonomies and Toolkits of Regular Language Algorithms, available at: http://www.diku.dk/hjemmesider/ansatte/henglein/papers/watson1995.pdf.

6. **Smyth B.** Computing Patterns in Strings, available at: http://scienceengineering.library.scilibgen.org/view.php?id=603002.

7. **The Aho—Corasick** algorithm, available at: https://en.wikipedia.org/wiki/Aho%E2%80%93Corasick_algorithm.

8. **Commentz W.** A String Matching Algorithm Fast On Average, *Proceedings of the 6th Colloquium, on Automata, Languages and Programming*, 1979, pp. 118—132.

9. **Boyer R. S., Moore J. S.** A Fast String Search Algorithm. *Communications of the Association for Computing Machinery*, 1977, vol. 20, no. 10, pp. 762—772, available at: https://www.cs.utexas.edu/~moore/publications/fstrpos.pdf.

10. **Description** of the Boyer—Moore Algorithm, available at: https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string_search_algorithm.

11. **Karp R. M., Rabin M. O.** Efficient randomized pattern-matching algorithms, *IBM Journal of Research and Development — Mathematics and computing archive*, 1987, vol. 31, no. 2, pp. 249—260, available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.9502&rep=rep1&type=pdf.

12. **Description** of the Rabin—Karp algorithm, available at: https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm.

13. **Examples** of Rolling-hash functions, available at: https://en.wikipedia.org/wiki/Rolling_hash.

14. **Gettys T.** Generating Perfect Hash Functions. First Fit Decreasing Method. 2001**,** available at: http://www.drdobbs.com/architecture-and-design/generating-perfect-hash-functions/184404506.

15. **Home** page of GNU grep project, available at: http://www.gnu.org/software/grep.

16. **CUDA-grep**, A Hardware Accelerated Regular Expression Matcher, available at: http://www.cs.cmu.edu/afs/cs/academic/class/15418-s12/www/competition/bkase.github.com/CUDA-grep/finalreport.html.

УДК 004.89 DOI: 10.17587/prin.7.14-20

В. В. Грибова, д-р техн. наук, зам. дир. по научной работе, e-mail: gribova@iacp.dvo.ru,
А. С. Клещев, д-р физ.-мат. наук, проф., гл. науч. сотр., e-mail: kleshev@iacp.dvo.ru,
Д. А. Крылов, канд. техн. наук, вед. инженер-программист, e-mail: dmalkr@gmail.com,
Ф. М. Москаленко, канд. техн. наук, науч. сотр., e-mail: philipmm@iacp.dvo.ru,
В. А. Тимченко, канд. техн. наук, науч. сотр., e-mail: vadim@iacp.dvo.ru,
Е. А. Шалфеева, канд. техн. наук, доц., ст. науч. сотр., e-mail: shalf@iacp.dvo.ru,
Институт автоматики и процессов управления Дальневосточного отделения Российской академии наук, г. Владивосток

Базовая технология разработки интеллектуальных сервисов на облачной платформе IACPaaS. Часть 2. Разработка агентов и шаблонов сообщений

Данная статья продолжает цикл работ, посвященных описанию базовой технологии разработки интеллектуальных мультиагентных сервисов и их компонентов с использованием инструментальных сервисов облачной платформы IACPaaS. Вторая часть посвящена описанию технологии разработки агентов и шаблонов сообщений решателя задач интеллектуального сервиса.

Ключевые слова: *интеллектуальные системы, мультиагентные системы, технология разработки интеллектуальных систем, иерархические орграфы, агентно-ориентированное программирование, облачные платформы, облачные сервисы*

Введение

В настоящей статье, входящей в цикл работ, посвященных описанию базовой технологии разработки интеллектуальных сервисов на платформе IACPaaS, и являющейся продолжением работы [1], дано описание технологии разработки агентов решателя задач интеллектуального сервиса и шаблонов сообщений, посредством которых агенты взаимодействуют между собой.

1. Разработка агентов решателя задач

1.1. Общие сведения

Под *агентом* облачной платформы IACPaaS будем понимать повторно используемый программный компонент, взаимодействующий с другими агентами посредством приема и передачи сообщений, а также способный обрабатывать (читать и модифицировать) единицы хранения Фонда платформы IACPaaS. Факт повторного использования означает, что агент может быть составной частью различных решателей задач без какой-либо модификации этого агента. Агент состоит из двух частей — декларативной и процедурной — и представлен кортежем вида $agent = \langle agent\_name, agent\_internal\_name, agent\_descr, [local\_structure], \{production\_block_1, ..., production\_block_n\}, [agent\_code]\rangle$, $n \geqslant 1$, где:

- *agent_name* — название агента на естественном языке, отображаемое всем авторизованным пользователям платформы IACPaaS при просмотре содержимого Фонда через Административную систему;
- *agent_internal_name* — внутреннее имя агента, которое используется в качестве имени класса-заготовки агента (в частности, при генерации заготовки его исходного кода) и в качестве префикса в имени класса-заготовки агента;
- *agent_descr* — содержательное описание назначения агента на естественном языке[1], включающее описание решаемой задачи или класса задач, шаблонов входных и выходных сообщений, посредством которых он взаимодействует с другими агентами, а также, возможно, единиц хранения Фонда, которые он обрабатывает;

[1] При генерации заготовки исходного кода агента данное описание помещается в исходный код его класса в качестве комментария к нему и ориентировано на разработчиков решателей задач и специалистов, сопровождающих агента. Это описание доступно всем авторизованным пользователям платформы IACPaaS при просмотре содержимого Фонда через Административную систему.

- *local_structure* — локальная структура данных, которая описывает структуру информации, доступной в каждом блоке продукций агента (при каждом обращении к экземпляру агента[2] во время работы сервиса) и используемой для хранения собственных данных, настроек агента, управляющих логикой его работы и т. п. (может отсутствовать, если все нужные данные доступны через сообщения и/или информационные ресурсы);
- $\{production\_block_1, ..., production\_block_n\}$ — непустое множество блоков продукций агента;
- *agent_code* — исполняемый код агента (байт-код), представляющий его процедурную часть (отсутствует до тех пор, пока код агента не разработан и не загружен в Фонд, в соответствующий этому агенту информационный ресурс).

Каждый блок продукций $production\_block_i$ ($i = 1, ..., n$) есть тройка вида $production\_block = \langle production\_block\_descr, inMsgTemplate, \{outMsgTemplate_1, ..., outMsgTemplate_m\}\rangle$, $m \geqslant 0$, где:

- *production_block_descr* — содержательное описание назначения блока продукций на естественном языке[3], включающее описание решаемой им задачи, а также, возможно, краткое описание метода ее решения;
- *inMsgTemplate* — шаблон входных сообщений, т. е. сообщений, инициирующих выполнение данного блока продукций агента (устройство шаблонов сообщений описано в разд. 2);
- $\{outMsgTemplate_1, ..., outMsgTemplate_m\}$ — возможно пустое множество шаблонов выходных сообщений, т. е. сообщений, создаваемых в процессе выполнения данного блока продукций агента и рассылаемых адресатам после завершения его выполнения (данный компонент может отсутствовать, если после выполнения блока продукций не требуется посылка каких-либо сообщений).

Декларативный компонент состоит, по существу, из двух частей — документации к агенту, которая представлена совокупностью содержательных описаний как самого агента, так и всех его блоков продукций на естественном языке, и формальной спецификации множества блоков продукций, из которых агент состоит. На основе декларативного компонента средствами платформы IACPaaS поддерживается автоматизация процесса разработки и сопровождения документированного исходного кода агента. Орграфовая связная двухуровневая модель информационных ресурсов и ее соответствующая поддержка процессором информационных ресурсов платформы IACPaaS позволяют хранить декларативную спецификацию и исполняемый код агента (получаемый в результате компиляции его исходного кода) в одной единице хранения Фонда, представляющей данный агент. Для создания и обработки таких единиц хранения платформа предоставляет инструментальный сервис.

Процедура генерации заготовки исходного кода разрабатываемого агента, в которую встраивается документация к последнему на основе декларативного описания этого агента, позволяет существенно упростить разработку и сопровождение процедурной части агента.

Для разработки агента необходимо, чтобы для каждого его блока продукций $production\_block_i$ ($i = 1, ..., n$) в Фонде присутствовали $inMsgTemplate_i$ — шаблон входных сообщений и, если $\{outMsgTemplate_1, ..., outMsgTemplate_m\} \neq \varnothing$, то все шаблоны выходных сообщений — $outMsgTemplate_{ij}$ ($j = 1, ..., m$).

Если некоторый шаблон сообщений $inMsgTemplate_i$ или $outMsgTemplate_{ij}$ отсутствует в Фонде, то его необходимо разработать по технологии, описанной в разд. 2.

Разработка агента состоит:

— в формировании в Фонде платформы IACPaaS информационного ресурса, представляющего декларативную спецификацию агента;

— в генерации заготовки исходного кода агента по его декларативному описанию;

— в написании исходного кода агента (в частности, кода блоков продукций агента);

— в получении байт-кода агента (в результате компиляции исходного кода) и загрузки его в Фонд — в соответствующий информационный ресурс.

1.2. Формирование информационного ресурса агента

Формирование в Фонде платформы информационного ресурса, представляющего декларативную спецификацию агента, выполняется по следующей схеме.

1. Создание с использованием Административной системы в Фонде платформы IACPaaS нового информационного ресурса (с названием, совпадающим с названием агента, и пустым содержимым), представляющего декларативную спецификацию разрабатываемого агента, по хранящейся в Фонде метаинформации *Структура агентов*, которая описывает онтологию декларативных представлений агентов платформы.

2. Формирование содержимого созданного информационного ресурса с использованием *Редактора агентов*, в котором процесс редактирования управляется метаинформацией Структура агентов. Специфицирование агента состоит в задании:

- описания агента (на естественном языке);
- внутреннего имени агента (оно должно удовлетворять ограничениям на допустимые имена классов в языке программирования Java);
- локальной структуры данных произвольного вида как орграфа, представляющего метаинформацию (если необходимо);

---

[2] Агент, у которого присутствует локальная структура данных, рассматривается процессором решателей задач как агент-экземпляр. Аналогами понятий "агент" и "агент-экземпляр" являются соответственно понятия "класс" и "объект" в объектно-ориентированной парадигме программирования.

[3] Данное описание помещается в исходный код класса агента в качестве комментария к методу, соответствующему данному блоку продукций. Оно ориентировано на разработчиков решателей задач и сопровождающих агента.

- для каждого блока продукций агента:
  - ◇ описания блока продукций (на естественном языке);
  - ◇ шаблона входных сообщений;
  - ◇ шаблонов выходных сообщений (если они есть).

Шаблоны входных и выходных сообщений задают путем создания ссылок на информационные ресурсы в Фонде платформы IACPaaS, представляющие декларативные спецификации соответствующих шаблонов сообщений.

Описывая блоки продукций, необходимо следовать перечисленным далее положениям:

- *корневой агент* должен содержать блок продукций, выполнение которого инициируется сообщением по шаблону *Инициализирующее сообщение*;
- если агент должен завершать работу решателя задач, то у него должен быть блок продукций, множество шаблонов выходных сообщений которого должно содержать шаблон *Финализирующее сообщение*;
- агент *Интерфейсный контроллер* должен иметь блок продукций, выполнение которого инициируется сообщением по шаблону *Запрос от агента Вид*;
- для ответа агенту платформы *Вид* (в целях формирования последним интерфейса для визуализации) агент Интерфейсный контроллер должен содержать блок продукций, множество шаблонов выходных сообщений которого должно содержать один из следующих шаблонов: *Отобразить окно*, *Вернуть инфоресурс в окно*, *Вернуть строку в окно*;
- все продукции, относящиеся к некоторому блоку продукций, обрабатывают сообщения по одному шаблону;
- разные блоки продукций одного и того же агента должны иметь разные шаблоны входных сообщений.

1.3. Генерация заготовки исходного кода агента

После того как декларативная часть разрабатываемого агента описана (или модифицирована), средствами Редактора агентов можно:

- на ее основе выполнить генерацию заготовки исходного кода разрабатываемого агента, включающей документацию, и загрузить ее на компьютер разработчика в виде упакованного в архив *<внутреннее имя агента>.java*-файла, содержащего набор необходимых классов на языке программирования Java;
- получить из Фонда исполняемый код используемых в блоках продукций данного агента (пользовательских и встроенных в платформу IACPaaS) шаблонов сообщений и загрузить его на компьютер разработчика в виде упакованных в архив файлов, содержащих байт-код этих шаблонов сообщений.

1.4. Написание исходного кода и подготовка байт-кода агента

Загрузив на компьютер архивы сгруппированных в пакеты файлов, содержащих набор необходимых для разрабатываемого агента классов на языке Java, а также байт-код используемых в нем шаблонов сообщений, разработчику необходимо создать (или модифицировать уже существующий у него) класс-наследник с именем *<внутреннее имя агента>Impl*. В этом классе необходимо переопределить все методы, соответствующие блокам продукций, описанным в декларативной спецификации (*void runProduction(...)*) агента, реализацией алгоритма решения задачи. В данном классе также можно описать и реализовать множество вспомогательных методов, внутренних классов и т. п. для использования последних внутри методов *void runProduction(...)*.

После того как исходный код агента разработан (или модифицирован), необходимо получить его байт-код и подготовить его к загрузке в Фонд. Для этого необходимо выполнить компиляцию исходного кода агента и поместить полученные в результате компиляции **.class*-файлы, содержащие байт-код агента, в *jar*- или *zip*-архив. Написание исходного кода агента и подготовка его байт-кода выполняется на компьютере разработчика.

1.5. Загрузка байт-кода агента в Фонд

Средствами Редактора агентов необходимо выполнить загрузку (или обновление) **.class*-файлов с байт-кодом агента, содержащихся в сформированном *jar*- или *zip*-архиве, в Фонд платформы IACPaaS — в информационный ресурс, представляющий декларативное описание агента. С этого момента виртуальная машина платформы IACPaaS может активировать данный агент и выполнять код его блоков продукций.

При генерации заготовок исходного кода и загрузке байт-кода агента проверяется полнота его декларативного описания. Если декларативное описание неполно, то разработчику отображается сообщение, локализующее соответствующее место в описании. При загрузке байт-кода агента и используемых в нем внутренних классов (если они есть) проверяется его корректность[4] и безопасность[5] с точки зрения его выполнения на виртуальной машине платформы. Если байт-код не корректен или не безопасен, то его загрузка или обновление не выполняется, а отображается сообщение о найденной ошибке.

1.6. Тестирование и отладка агента

Для функционального тестирования разработанного агента используется *Тестировщик агентов*, который обеспечивает многократный запуск и выполнение множества блоков продукций агента на заданном множестве тестов, а также поддерживает формирование и сохранение отчетов о результатах

---

[4] Среди классов в *jar*- или *zip*-архиве присутствует единственный класс-наследник от главного класса заготовки с именем *<внутреннее имя агента>*, в этом единственном классе-наследнике переопределены все методы *void runProduction(...)*, соответствующие блокам продукций агента, и т. п.

[5] Среди импортируемых классов отсутствуют те, используя которые можно получить доступ к внешним по отношению к платформе IACPaaS сущностям — операционной системе, файловой системе, СУБД, сокетам и т. п.

испытаний. Множество тестов для агента формируется с помощью *Редактора орграфов информации* по метаинформации *Структура тестов агента*. Тест для агента в общем случае есть четверка $test = \langle inMsg, \{testOutMsg_1, ..., testOutMsg_k\}, \{modifIr_1, ..., modifIr_n\}, \{testModifIr_1, ..., testModifIr_n\}\rangle$, $k \geqslant 0$, $n \geqslant 0$, где:

- *inMsg* — входное сообщение для некоторого блока продукций тестируемого агента;
- $\{testOutMsg_1, ..., testOutMsg_k\}$ — возможно пустое множество ожидаемых от тестируемого агента выходных сообщений в ответ на входное сообщение *inMsg* (отсутствуют, если от агента не ожидается никаких сообщений);
- $\{modifIr_1, ..., modifIr_n\}$ — возможно пустое упорядоченное множество информационных ресурсов, каждый из которых представляет начальное состояние одного из информационных ресурсов, который изменяется в процессе работы блока продукций тестируемого агента (множество является пустым, если агент не модифицирует никаких информационных ресурсов);
- $\{testModifIr_1, ..., testModifIr_n\}$ — возможно пустое упорядоченное множество информационных ресурсов, каждый из которых представляет ожидаемое конечное состояние одного из информационных ресурсов, указанных в предыдущем пункте ($testModifIr_i$ соответствует $modifIr_i$ ($i = 1, ..., n$)), после завершения работы блока продукций тестируемого агента (множество является пустым, если агент не модифицирует никаких информационных ресурсов).

Тест считается успешно пройденным:

- если число и содержимое ожидаемых сообщений $testOutMsg_i$ ($i = 1, ..., k$) совпало с числом и содержимым соответствующих сообщений, которые тестируемый агент послал в результате работы блока продукций — $outMsg_i$ ($i = 1, ..., m$), $k = m$;
- если содержимое изменяемых в процессе работы блока продукций информационных ресурсов $modifIr_i$ совпало с содержимым соответствующих информационных ресурсов $testModifIr_i$ ($i = 1, ..., n$).

Для просмотра отчетов о результатах испытаний и журналов используется Редактор орграфов информации (в режиме просмотра). Журнал работы агента на конкретном тесте присоединяется к отчету и также доступен для просмотра. Журналирование применяется для получения информации о том, какие события и в какой последовательности происходят во время работы блоков продукций агента, а также для того чтобы локализовать место возникновения ошибки. Сформированные наборы тестов могут использоваться для регрессионного тестирования в процессе сопровождения агента.

1.7. Ввод агента в эксплуатацию

После успешного завершения тестирования и отладки, с разрешения администратора предметной области, для которой разработан агент, последний средствами Административной системы должен быть переведен в режим публичного доступа. Находящийся в публичном доступе агент может быть включен в состав различных решателей задач на этапе их разработки и/или интеграции.

2. Разработка шаблонов сообщений

2.1. Общие сведения

Напомним, что под *сообщением* в мультиагентных системах понимается средство обеспечения согласованного взаимодействия агентов, подразумевающее обмен информацией между ними и передачу запросов на предоставление услуг [2]. С точки зрения представления сообщения как некоторой информационной единицы, формируемой в Фонде платформы IACPaaS, оно является временным информационным ресурсом. Жизненный цикл сообщения начинается с его создания некоторым агентом, за которым следует посылка этого сообщения другому агенту, который, в свою очередь, его получает и обрабатывает. После этого сообщение прекращает свое существование. Вышесказанное не относится к процессу тестирования и отладки агента, где сообщения, используемые в запускаемых тестах (в качестве входных и ожидаемых от агента выходных), представляют собой постоянные информационные ресурсы.

Сообщения должны быть представлены на некотором(-ых) языке(-ах), синтаксис и семантика которого(-ых) должны быть понятны взаимодействующим агентам. Поскольку сообщения — суть информационные ресурсы, представляющие информацию, то язык представления некоторого множества сообщений описывается информационным ресурсом, представляющим собой их метаинформацию. Под *шаблоном сообщений* облачной платформы IACPaaS будем понимать единицу хранения, содержащую метаинформацию определенного множества сообщений и множество методов обработки сообщений из этого множества (описание синтаксиса и семантики языка взаимодействия агентов).

По аналогии с агентом, шаблон сообщений состоит из двух частей — декларативной и процедурной. Шаблон представляет собой пятерку $msgTemplate = \langle template\_name, template\_internal\_name, template\_descr, [message\_structure], [template\_code]\rangle$, где:

- *template_name* — название шаблона сообщений на естественном языке, отображаемое всем авторизованным пользователям платформы IACPaaS при просмотре ими содержимого Фонда через Административную систему;
- *template_internal_name* — внутреннее имя шаблона сообщений, которое используется в качестве префикса в имени класса шаблона сообщений при генерации заготовки его исходного кода;
- *template_descr* — содержательное описание назначения шаблона сообщений на естественном языке[6], вклю-

[6] Данное описание помещается в исходный код класса шаблона сообщений в качестве комментария к нему и ориентировано на разработчиков агентов и специалистов, сопровождающих шаблоны сообщений. Это описание доступно также всем авторизованным пользователям платформы IACPaaS при просмотре содержимого Фонда через Административную систему.

чающее, возможно, толкование семантики содержательной информации, передаваемой в сообщениях по данному шаблону, а также, возможно, перечисление единиц хранения Фонда, ссылки на которые (как на фрагменты повторно используемой информации) могут в таких сообщениях содержаться;

- *message_structure* — описание структуры (синтаксиса языка представления) содержательной информации, которая передается в сообщениях, формируемых по данному шаблону (содержимого сообщений) — в случае, когда шаблон сообщений описывает не просто сообщения-команды, а сообщения, которые должны содержать некоторые данные;
- *template_code* — исполняемый код шаблона сообщений (байт-код)[7], представляющий его процедурную часть (отсутствует до тех пор пока код шаблона сообщений не разработан и не загружен в Фонд, в соответствующий информационный ресурс).

В существующих передовых стандартах построения мультиагентных систем, предлагаемых такими организациями, как FIPA, KAoS и OMG [3—5], языки коммуникации агентов (ACL, KQML) обладают плоской (горизонтальной) структурой. Они позволяют описывать сообщения, посредством которых агенты взаимодействуют между собой, в виде множества пар *имя параметра = значение параметра*. Значением параметра при этом может быть объект со сколь угодно сложной структурой (описываемый на языке представления этого объекта). Расширение этих языков состоит в наращивании числа используемых параметров, многие из которых объявляются необязательными к использованию при описании сообщений. Таким образом, разработчики стандартов пошли по пути создания и совершенствования одного универсального языка, в котором предпринимаются попытки учесть всевозможные ситуации взаимодействия агентов. Однако такой подход ведет к повышению избыточности языка и вместе с тем — к снижению его выразительности в силу того, что параметры, равно как и их значения, рассматривают независимо друг от друга.

Вместе с тем практика разработки и развития формальных языков показывает следующее:

- в большинстве своем они являются языками со сложной синтаксической структурой, которая может быть представлена сколь угодно сложными графами;
- как правило, проблемно-ориентированный (узкоспециализированный) язык позволяет более адекватно и выразительно формализовать целевое описание (сценарий, декларативную структуру, спецификацию и т. п.), соответствующее области его применения, чем универсальный язык общего назначения, использование которого для конкретных узких целей менее эффективно.

Альтернативным подходом к разработке средств взаимодействия агентов является многоязыковый подход. Именно он положен в основу взаимодействия агентов платформы IACPaaS. Каждый язык может иметь сколь угодно сложную синтаксическую структуру (представляемую орграфом метаинформации), в соответствии с которой формируется содержимое множества сообщений (представляемых орграфами информации). При этом как сам язык, так и все множество языков являются расширяемыми. Сочетание этих факторов в совокупности позволяет сводить к минимуму избыточность каждого конкретного языка, добиваясь при этом повышения его выразительности.

Процессы расширения как отдельно взятого языка, так и множества всех языков контролируются аппаратом управления платформы, в который входят администраторы предметных областей и администратор всей платформы IACPaaS. Основная цель, которая при этом преследуется аппаратом управления, состоит в контроле уровня повторного использования разрабатываемых шаблонов сообщений (содержащих описание синтаксиса и семантики языка взаимодействия агентов) и выработке рекомендаций по его повышению, с одной стороны, и контроле над тем, чтобы шаблон сообщений не претендовал на роль одного универсального языка, с другой стороны. Последняя ситуация при многоязыковом подходе не запрещается, но при этом противоречит его идеологии.

Агенты взаимодействуют между собой на разных языках, число которых расширяемо (за счет добавления в агенты новых блоков продукций) и ограничено лишь общим количеством шаблонов сообщений в Фонде платформы IACPaaS. Конкретные языки взаимодействия агентов (шаблоны сообщений) фиксируются на уровне отдельных блоков продукций. Таким образом, множество языков, на которых описаны сообщения, которые агент может принимать, определяется множеством шаблонов входных сообщений для всех его блоков продукций, а множество языков, на которых описаны сообщения, которые агент может отправлять, определяется множеством, в которое включаются разные шаблоны выходных сообщений для всех его блоков продукций. Интеграция языков выполняется динамически — средствами виртуальной машины платформы IACPaaS в процессе взаимодействия агентов друг с другом.

Выделение в шаблонах сообщений декларативного и процедурного компонентов позволяет получить такие же преимущества, что и в случае с агентами (см. разд. 1).

Разработка шаблона сообщений состоит в общем случае из тех же этапов, что и разработка агента. Различия имеются лишь на этапе написания исходного кода и обусловлены различиями в структуре декларативных спецификаций агентов и шаблонов сообщений.

2.2. Формирование информационного ресурса шаблона сообщений

Формирование в Фонде информационного ресурса, представляющего декларативную спецификацию шаблона сообщений, выполняется по следующей схеме.

[7] В случае, когда шаблон сообщений описывает сообщения, которые должны содержать некоторые данные, этот код содержит методы обработки содержимого сообщений, реализующие семантику языка его представления.

1. Создание с использованием Административной системы в Фонде платформы IACPaaS нового информационного ресурса (с названием, совпадающим с названием шаблона сообщений, и пустым содержимым), представляющего декларативную спецификацию разрабатываемого шаблона сообщений, по хранящейся в Фонде метаинформации *Структура шаблонов сообщений*, описывающей онтологию декларативных представлений шаблонов сообщений платформы (априори присутствующей в Фонде платформы IACPaaS).

2. Формирование содержимого созданного информационного ресурса с использованием сервиса *Редактор шаблонов сообщений*, в котором процесс редактирования управляется метаинформацией Структура шаблонов сообщений. Специфицирование шаблона сообщений состоит в задании:

- описания шаблона (на естественном языке);
- внутреннего имени шаблона (оно должно удовлетворять ограничениям на допустимые имена классов в языке программирования Java);
- структуры содержимого сообщений произвольного вида как орграфа, представляющего метаинформацию (если необходимо).

2.3. Генерация заготовки исходного кода шаблона сообщений

После того как декларативная часть разрабатываемого шаблона сообщений описана (или модифицирована), средствами Редактора шаблонов сообщений на ее основе можно выполнить генерацию заготовки исходного кода разрабатываемого шаблона сообщений, включающей документацию. Затем следует загрузить ее на компьютер разработчика в виде упакованного в архив файла, содержащего набор необходимых классов на языке программирования Java (размещенных в определенном пакете).

2.4. Написание исходного кода и подготовка байт-кода шаблона сообщений

Если в декларативной спецификации шаблона сообщений присутствует орграф, описывающий структуру содержимого сообщений, разработчику необходимо дополнить сгенерированный класс-заготовку кода шаблона сообщений методами обработки содержимого сообщений. К их числу относятся способы порождения, чтения и модификации орграфа информации, представляющего содержимое сообщения, в соответствии с орграфом метаинформации, описывающим структуру содержимого сообщений. В данном классе можно также описать и реализовать вспомогательные методы, внутренние классы и т. п. Таким образом, созданные по такому шаблону сообщения инкапсулируют в себе данные и методы их обработки. Такой подход существенно повышает степень повторного использования разрабатываемого шаблона.

Сгенерированный файл также может быть оставлен без изменений. Однако в такой ситуации весь код обработки содержимого сообщений (при наличии орграфа, описывающего структуру содержимого сообщений) разработчику необходимо реализовывать в агенте, который обрабатывает сообщения, созданные по соответствующему шаблону. Это ведет к дополнительному усложнению логики работы этого агента, а также может приводить к росту числа ошибок программирования, связанных с некорректной обработкой содержимого сообщений.

Подготовка байт-кода шаблона сообщений выполняется аналогично подготовке байт-кода агента. Отличие состоит лишь в том, что в случае с агентом необходима компиляция двух файлов, содержащих исходный код.

2.5. Загрузка байт-кода шаблона сообщений в Фонд

Средствами Редактора шаблонов сообщений необходимо выполнить загрузку (или обновление) **.class*-файлов с байт-кодом шаблона сообщений, содержащихся в сформированном *jar*- или *zip*-архиве, в Фонд платформы IACPaaS, а именно — в информационный ресурс, представляющий декларативное описание шаблона сообщений. С этого момента виртуальная машина платформы IACPaaS может создавать сообщения по данному шаблону, осуществлять их диспетчеризацию и своевременное удаление.

Как и в случае с агентами, при генерации заготовок исходного кода и загрузке байт-кода шаблона сообщений проверяется полнота его декларативного описания. В последнем случае проверяется также корректность и безопасность байт-кода шаблона сообщений и используемых в нем внутренних классов (если они есть).

2.6. Ввод шаблона сообщений в эксплуатацию

После загрузки байт-кода шаблона сообщений в Фонд с разрешения администратора предметной области, для которой разработан этот шаблон, последний средствами Административной системы должен быть переведен в режим публичного доступа. Находящийся в публичном доступе шаблон сообщений может использоваться в качестве шаблона входных и/или выходных сообщений в блоках продукций различных агентов на этапе разработки последних.

Заключение

Рассмотрены элементы базовой технологии разработки интеллектуальных сервисов платформы IACPaaS — технология разработки таких компонентов сервиса, как агент и шаблон сообщений. Разработка агента состоит: в формировании информационного ресурса, представляющего декларативную спецификацию агента; в генерации заготовки исходного кода агента по его декларативному описанию; в написании исходного кода агента (в частности, кода блоков продукций агента); в получении байт-кода агента (в результате компиляции исходного кода) и загрузки его в Фонд платформы (в соответствующий информационный ресурс). Разработка шаблона сообщений состоит в общем случае из тех же этапов, что и разработка агента.

Работа выполнена при частичной финансовой поддержке РФФИ (проект 13-07-00024, проект 14-07-00299 и проект 15-07-03193) и программы ФНИ (Дальний Восток).

Список литературы

1. **Грибова В. В., Клещев А. С., Крылов Д. А.** и др. Базовая технология разработки интеллектуальных сервисов на облачной платформе IACPaaS. Часть 1. Разработка базы знаний и решателя задач // Программная инженерия. 2015. № 12. С. 3—11.

2. **Тарасов В. Б**. От многоагентных систем к интеллектуальным организациям: философия, психология, информатика. М.: Эдиториал УРСС, 2002. 352 с.

3. **FIPA** Agent Management Specification [Электронный ресурс]. URL: http://www.fipa.org/specs/fipa00023/ (дата обращения 18.02.2015).

4. **Bradshaw J. M.** KAoS: An open agent architecture supporting reuse, interoperability, and extensibility/ Ed. B. R. Gaines, M. Musen// Proceedings of the Tenth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, 2. Banff, Alberta, Canada. 1996. P. 1—20.

5. **Milojicic D., Breugst M., Busse I.** et. al. MASIF. The OMG Mobile Agent System Interoperability Facility // Personal Technologies. 1998. Vol. 2, Issue 2. P. 117—129.

The Base Technology for Intelligent Services Development with the Use of IACPaaS Cloud Platform. Part 2. Development of Agents and Message Templates

V. V. Gribova, gribova@iacp.dvo.ru, **A. S. Kleschev,** kleschev@iacp.dvo.ru, **D. A. Krylov,** dmalkr@gmail.com, **Ph. M. Moskalenko,** philipmm@iacp.dvo.ru, **V. A. Timchenko,** vadim@iacp.dvo.ru, **E. A. Shalfeyeva,** shalf@iacp.dvo.ru, Federal State Budgetary Institution of Science "Institute for Automation and Control Processes" (Far Eastern Branch of the Russian Academy of Sciences), Vladivostok, 690041, Russian Federation

Corresponding author:

Moskalenko Philip M., Researcher, Federal State Budgetary Institution of Science "Institute for Automation and Control Processes" (Far Eastern Branch of the Russian Academy of Sciences), Vladivostok, 690041, Russian Federation, e-mail: philipmm@iacp.dvo.ru

Received on August 25, 2015
Accepted on October 23, 2015

The paper continues a series of works which present a base technology for development of intelligent multi-agent services and their components with the use of system tools of IACPaaS cloud platform. The technology is put to reduce the labour-intensiveness of development and first of all support for intelligent cloud services and proposes involvement of domain experts (without mediators or additional training) in these processes, whose task is to create and maintain information components of platforms' applied services in actual state during their lifecycle. This paper presents a technology for development of agents and message templates for problem solver of an intelligent service.

Keywords: *intelligent system, multi-agent system, intelligent software development technology, hierarchical semantic network, agent-oriented programming, cloud platform, cloud service*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project nos. 13-07-00024, 14-07-00299 and 15-07-03193 and by the Fundamental Scientific Research, Far East program.*

For citation:

Gribova V. V., Kleschev A. S., Krylov D. A., Moskalenko Ph. M., Timchenko V. A., Shalfeyeva E. A. A Base Technology for Intelligent Services Development with the Use of IACPaaS Cloud Platform. Part 2. A Development of Agents and Message Templates, *Programmnaya Ingeneria,* 2016, vol. 7, no 1, pp. 14—20.

DOI: 10.17587/prin.7.14-20

References

1. **Gribova V. V., Kleschev A. S., Krylov D. A., Moskalenko Ph. M., Timchenko V. A., Shalfeyeva E. A.** Bazovaja tekhnologija razrabotki intellektual'nykh servisov na oblachnoj platforme IACPaaS. Chast' 1. Razrabotka bazy znanij i reshatelya zadach (A Base Technology for Development of Intelligent Services with the Use of IACPaaS Cloud Platform. Part 1. A Development of Knowledge Base and Problem Solver) *Programmnaya Ingeneria*, 2015, no. 12, pp. 3—11 (in Russian).

2. **Tarasov V. B.** *Ot mnogoagentnykh sistem k intellektual'nym organizatsiyam: filosofiya, psikhologiya, informatika* (From multiagent systems to intellectual organizations: philosophy, psychology, computer science). Moscow, Editorial URSS, 2002. 352 p. (in Russian).

3. **FIPA** Agent Management Specification, available at: http://www.fipa.org/specs/fipa00023/ (date of access 18.02.2015).

4. **Bradshaw J. M.** KAoS: An open agent architecture supporting reuse, interoperability, and extensibility. B. R. Gaines & M. Musen (Ed.), *Proc. 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, 2. Banff, Alberta, Canada, 1996, pp. 1—20.

5. **Milojicic D., Breugst M., Busse I., Campbell J., Covaci S., Friedman B., Kosaka K., Lange D., Ono K., Oshima M., Tham C., Virdhagriswaran S., White J. less S.** MASIF: The OMG mobile agent system interoperability facility, *Personal Technologies,* 1998, vol. 2, issue 2, pp. 117—129.

УДК 512.742.72 DOI: 10.17587/prin.7.21-28

Д. С. Хлебородов, аспирант, e-mail: dkhleborodov@gmail.com,
Московский государственный технический университет имени Н. Э. Баумана

Эффективный алгоритм скалярного умножения точки эллиптической кривой на основе NAF-метода

Представлен алгоритм эффективного скалярного умножения точки эллиптической кривой, определенной над простым полем, на основе метода бинарного несовместного представления скаляра (Non-Adjacent Form, NAF). Для оценки эффективности полученных и известных на настоящее время алгоритмов введен критерий эффективности, основанный на средней вычислительной сложности. Для получения нового, более эффективного алгоритма использованы: эффективные операции в простом поле; операции с точкой, включая сложение, удвоение; операция "удвоить и сложить", масштабирование; свойства аффинной координатной системы; свойства координатной системы Якоби. Для предложенного алгоритма сформулировано и доказано утверждение относительно его вычислительной сложности.

Ключевые слова: *эллиптические кривые, скалярное умножение, операции с точкой, быстрые алгоритмы, вычислительная сложность, преобразования на эллиптических кривых, алгоритм*

Введение

Преобразования на основе эллиптических кривых широко используют в криптосистемах с открытым ключом (*Elliptic Curve Cryptography*, ECC), а также — в криптоанализе асимметричных шифров, основанных на решении задачи факторизации целых чисел. Главное преимущество подхода, используемого в криптографии на основе эллиптических кривых, по сравнению с другими подходами, используемыми в криптографии на основе простых полей (*Finite Field Cryptography*, FFC) и криптографии на основе факторизации целых чисел (*Integer Factorization Cryptography*, IFC) заключается в возможности обеспечить тот же уровень безопасности при существенно более коротких ключах (табл. 1) [16].

В первом столбце в табл. 1 указано число бит безопасности, которые обеспечивают соответствующие криптосистемы. Во втором столбце приведены названия алгоритмов симметричного шифрования. В остальных столбцах представлены соответствующие длины ключей. К группе FFC относят алгоритмы DSA (*Digital Signature Algorithm*) и DH (*Diffie-Hellman*), к группе IFC — алгоритм RSA (*Rivest, Shamir and Adleman*), а к ECC — алгоритм подписи на основе эллиптических кривых ECDS (*Elliptic Curve Digital Signature*). Как видно из данных табл. 1, криптосистема ECC обеспечивает уровень безопасности 256 бит при длине ключа в 30 раз меньше, чем криптосистемы FFC и IFC. Это возможно благодаря тому, что на настоящее время неизвестно о существовании субэкспоненциальных алгоритмов решения задачи дискретного логарифмирования в группе точек эллиптической кривой. Востребованность криптосистем, построенных на эллиптических кривых, обусловлена современными потребностями в снижении вычислительных ресурсов, энергопотребления и памяти при обеспечении заданного уровня безопасности. Преобразования на основе эллиптических кривых используют, в частности, при вычислении цифровых подписей в отечественном (ГОСТ Р 34.10—2012 [1]) и

Таблица 1

Соответствие уровней безопасности

Уровень безопасности, бит	Симметричные алгоритмы	Криптосистемы		
		FFC (DSA, DH), бит	IFC (RSA) k, бит	ECC (ECDS) f, бит
80	2TDEA	$L = 1024$ $N = 160$	1024	160...223
112	3TDEA	$L = 2048$ $N = 224$	2048	224...255
128	AES-128	$L = 3072$ $N = 256$	3072	256...383
192	AES-192	$L = 7680$ $N = 384$	7680	384...511
256	AES-256	$L = 15\,360$ $N = 512$	15 360	512+

ряде зарубежных (FIPS PUB 186 [8], ANSI X9.623 [2], SECG4 [18]) стандартах. Они находят также свое применение среди основных операций в алгоритмах тестирования натурального числа на "простоту". Упомянутые алгоритмы представляют как практический, так и теоретический интерес.

Существенным недостатком преобразований, основанных на эллиптических кривых, является их высокая вычислительная сложность. Этот недостаток может быть устранен за счет применения новых, более эффективных алгоритмов.

Целью исследования, результаты которого представлены в настоящей статье, является построение эффективного алгоритма вычисления скалярного умножения точки в соответствии с установленным критерием эффективности на основе метода бинарного несовместного представления скаляра (*Non-Adjacent Form*, NAF). Для этого было необходимо решить следующие задачи.

- Сформулировать критерий эффективности алгоритма скалярного умножения точки на основе NAF-метода.
- Получить новый, более эффективный, чем известный на настоящее время алгоритм.
- Доказать эффективность полученного алгоритма относительно известных на настоящее время алгоритмов.

В качестве методов исследования для решения перечисленных задач применяли теорию алгоритмов; теорию вычислительной сложности; аппарат высшей алгебры и теории чисел.

Для анализа научных результатов в исследуемой области в предварительном порядке были изучены материалы из двух наиболее широко известных и авторитетных источников — электронных каталогов статей Springer и ePrint, в которых представлены публикации, отражающие текущее состояние исследований в рассматриваемой проблемной области.

Свойства NAF-представления скаляра сформулированы В. Босма [7] в виде теоремы (полная формулировка которой приведена в разд. 3 настоящей статьи). Приблизительная оценка средней вычислительной сложности алгоритмов на основе NAF-метода сформулирована следующей леммой.

Лемма 1 [17]. *Для случайных скалярных величин $d \in \mathbb{Z}$, имеющих среднюю длину n, предполагаемое число удвоений (DBL) и сложений (ADD) при использовании алгоритма на основе метода несовместного представления (w = 2, w — ширина окна разложения) приблизительно равно (n − 1) и $\frac{n}{3}$ соответственно.*

Таким образом, средняя вычислительная сложность $\hat{L}(n)$ алгоритма на основе метода несовместного представления скаляра приблизительно равна

$$\hat{L}(n) = (n-1)\,DBL + \left(\frac{n}{3}\right) ADD. \qquad (1)$$

Лучшая, известная на настоящее время, средняя оценка $\hat{L}(n)$ вычислительной сложности алгоритма умножения точки на основе бинарного несовместного представления скаляра d, предложенного Д. Гордоном [10], получена Н. Суливаном [19]:

$$\hat{L}(n) = \mathbf{I} + \left(\frac{20}{3}n - 11\right)\mathbf{M} + \left(\frac{15}{3}n - 6\right)\mathbf{S}, \qquad (2)$$

где **I**, **M** и **S** — вычислительная сложность операций нахождения мультипликативного обратного элемента, умножения и возведения в квадрат в поле, над которым определена эллиптическая кривая, соответственно.

Сформулированные свойства NAF-представления скаляра, лемма о приблизительной оценке средней вычислительной сложности являются вспомогательными для получения нового алгоритма и оценки его эффективности. Представленная выше лучшая средняя оценка известного на настоящее время алгоритма на основе NAF-метода позволит доказать эффективность искомого алгоритма. Для полноты изложения полученных результатов в следующем разделе будет сформулирован ряд важных определений.

1. Эллиптические кривые над простыми полями

Определение 1 [12, 15]. *Пусть $\mathbb{F}_p$ является простым полем и для a, $\beta \in \mathbb{F}_p$ выполняется неравенство $4a^3 + 27\beta^2 \not\equiv 0 (\mathrm{mod}\, p)$. Тогда эллиптическая кривая $E(\mathbb{F}_p)$ над $\mathbb{F}_p$ (p — характеристика поля), определенная параметрами a, $\beta \in \mathbb{F}_p$, состоит из набора решений или точек $P = (x, y)$, $x, y \in \mathbb{F}_p$ уравнения*

$$E(\mathbb{F}_p): y^2 \equiv x^3 + ax^2 + \beta (\mathrm{mod}\, p), \qquad (3)$$

вместе с особой точкой $\mathcal{O}$, называемой точкой в бесконечности. Уравнение $y^2 \equiv x^3 + ax^2 + \beta (\mathrm{mod}\, p)$ называется определяющим уравнением $E(\mathbb{F}_p)$.

Для точки $P = (x_P, y_P)$, x_P является x-координатой P, а y_P является y-координатой P. Такого рода интерпретация точки называется аффинными координатами.

Число точек эллиптической кривой $E(\mathbb{F}_p)$ обозначается $\#E(\mathbb{F}_p)$ и определяет значение скаляра.

Теорема 1 (*Х. Хассе*). *Для $\#E(\mathbb{F}_p)$ выполняется следующее соотношение:*

$$p + 1 - 2\sqrt{p} \leqslant \#E(\mathbb{F}_p) \leqslant p + 1 + 2\sqrt{p}.$$

Определение 2. *Набор точек на $E(\mathbb{F}_p)$ (3) вместе с особой точкой $\mathcal{O}$, называемой точкой в бесконечности, представляет собой абелеву группу с операцией сложения $\left(E(\mathbb{F}_p), \boxplus, \mathcal{O}\right)$.*

В основе криптосистем с открытым ключом лежит проблема дискретного логарифмирования (*Elliptic Curve Discrete Logarithm Problem*, ECDLP), которая формулируется следующим образом.

Определение 3. *Дана эллиптическая кривая E, определенная над простым полем $\mathbb{F}_p$, точка $P \in E(\mathbb{F}_p)$*

порядка n и точка $Q \in \langle P \rangle$, *найти* $l \in [1, n-1]$ *такое, что* $Q = lP$. *Целое l является дискретным логарифмом Q по основанию P и обозначается* $l = \log_P Q$.

Как видно, в основе криптографических преобразований, основанных на эллиптических кривых, лежит скалярное умножение точек эллиптической кривой.

Дано целое число k и точка $P \in E(\mathbb{F}_p)$, скалярное умножение — это процесс сложения точки P с собой k раз. Результат скалярного умножения обозначается $kP = \underbrace{P \boxplus \ldots \boxplus P}_{k \text{ раз}}$.

Определим закон сложения точек на E. Закон сложения определяют следующим образом [13].

1. Правило сложения точки в бесконечности:

$$\mathcal{O} \boxplus \mathcal{O} = \mathcal{O}.$$

2. Сложение точки в бесконечности с любой другой точкой:

$$(x, y) \boxplus \mathcal{O} = \mathcal{O} \boxplus (x, y) = (x, y),\ \forall (x, y) \in E(\mathbb{F}_p).$$

3. Правило сложения двух точек с одинаковыми x-координатами, когда точки либо различны либо имеют y-координату, равную 0:

$$(x, y) \boxplus (x, -y) = \mathcal{O},\ \forall (x, y) \in E(\mathbb{F}_p),$$

где отрицательной точкой (x, y) является $-(x, y) = (x, -y)$.

4. Правило сложения двух точек с различными x-координатами: пусть $(x_1, y_1) \in E(\mathbb{F}_p)$ и $(x_2, y_2) \in E(\mathbb{F}_p)$ такие точки, что $x_1 \neq x_2$. Тогда $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, где:

$$x_3 \equiv \lambda^2 - x_1 - x_2 \pmod p,$$
$$y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod p \ \text{ и } \ \lambda \equiv \frac{y_2 - y_1}{x_2 - x_1} \pmod p. \quad (4)$$

5. Правило сложения точки с собой (удвоение точки): пусть $(x_1, y_1) \in E(\mathbb{F}_p)$ точка и $y_1 \neq 0$. Тогда $(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$, где:

$$x_3 \equiv \lambda^2 - 2x_1 \pmod p,$$
$$y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod p \ \text{ и } \ \lambda \equiv \frac{3x_1^2 + a}{2y_1} \pmod p. \quad (5)$$

Под *вычислительной* (*средней вычислительной*) *сложностью алгоритма a* понимается временная сложность, т. е. время (среднее время) работы алгоритма a как функция $L_a(n)$ $\left(\hat{L}_a(n)\right)$ от размера входа n. Емкостная сложность алгоритма определяется объемом памяти, необходимой для его выполнения. Для предлагаемого алгоритма емкостная сложность равна по порядку размеру входа n. Следует отметить, что при исследованиях в данной области внимание емкостным оценкам алгоритмов, как правило, не уделяют.

Базовые операции на кривой в ECC над простыми полями выполняют с использованием операций в поле [12]. Последние состоят из традиционных арифметических операций, выполняемых по модулю простого числа p.

Таблица 2

Обозначение вычислительной сложности операций в простом поле

Сложность	Операция
A	Сложение/вычитание (**a**ddition/subtraction) в поле $\mathbb{F}_p$
R	Приведение по модулю (**r**eduction) p
S	Возведение в квадрат (**s**quaring) в поле $\mathbb{F}_p$
M	Умножение в поле (**m**ultiplication) в поле $\mathbb{F}_p$
I	Нахождение мультипликативного обратного элемента (**i**nversion) в поле $\mathbb{F}_p$

Введем обозначения стоимости (времени) вычислений операций в поле. В табл. 2 дано обозначение их вычислительной сложности и представлено описание основных операций в поле.

В данном исследовании для сравнения эффективности различных алгоритмов будем считать (что широко принято), что константы в оценках вычислительной сложности соотносятся как $1\mathbf{S} = 0{,}6\mathbf{M}$ [3, 6, 9, 14], $1\mathbf{A} = 0{,}05\mathbf{M}$ [4, 5, 11].

После обозначения стоимостных оценок операций в поле (см. табл. 2) заметим, что вычислительная сложность сложения (4) двух точек P и Q с координатами $(x_1, y_1) \in E(\mathbb{F}_p)$ и $(x_2, y_2) \in E(\mathbb{F}_p)$, $x_1 \neq x_2$, равна $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$. Вычислительная сложность удвоения (5) точки P с координатой $(x, y) \in E(\mathbb{F}_p)$ $y \neq 0$, равна $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$. Отсюда следует, что выполнение сложения и удвоения точек в аффинных координатах требует нахождения мультипликативных обратных элементов. Нахождение мультипликативного обратного элемента над простыми полями является наиболее дорогостоящей операцией, которую нужно стараться избегать.

2. Критерий эффективности

Для построения эффективных алгоритмов на основе эллиптических кривых и оценки их вычислительной сложности воспользуемся иерархическим подходом. Рассмотрим математическую архитектуру алгоритмов на основе эллиптических кривых, состоящую из пяти уровней (рис. 1). Каждый из уровней представляет собой множество алгоритмов, являющихся основой для получения алгоритмов вышестоящего уровня методом композиции.

Уровень $\mathcal{L}_1$. К первому уровню архитектуры относятся алгоритмы операций с большими целыми числами, а именно:

— *сложение/вычитание* больших целых чисел (даны $g, b \in \mathbb{Z}$, $g \pm b \in \mathbb{Z}$);

— *умножение* больших целых чисел (даны $g, b \in \mathbb{Z}$, $gb \in \mathbb{Z}$).

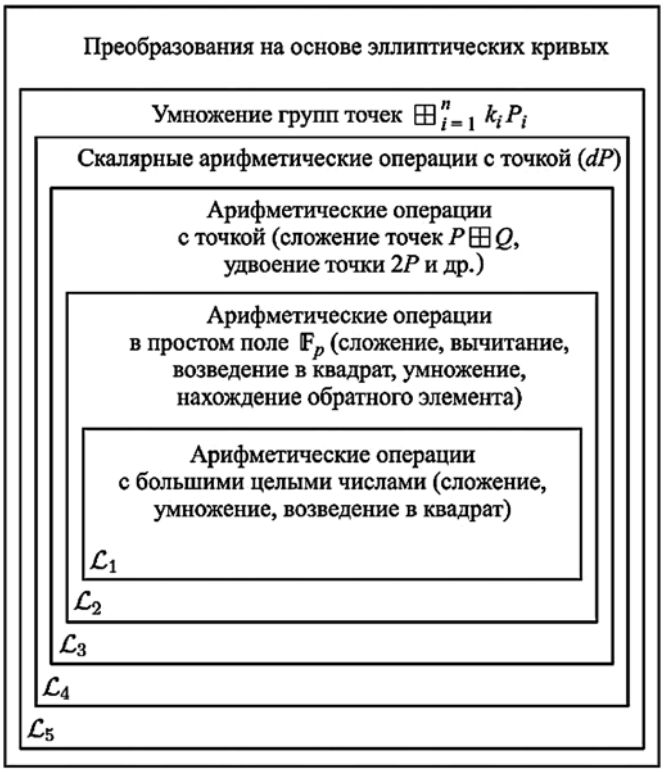

Рис. 1. Архитектура преобразований на основе эллиптических кривых

В качестве входных данных принимают n-разрядные целочисленные значения. Следует отметить, что целесообразность отдельного рассмотрения данного уровня имеет большое значение в определении вычислительной сложности алгоритмов вышестоящих уровней. Эффективность преобразований на основе эллиптических кривых может быть существенно снижена в силу неэффективных алгоритмов на данном уровне.

Уровень $\mathcal{L}_2$. Ко второму уровню архитектуры относят алгоритмы операций над элементами простого поля $\mathbb{F}_p$, которые можно также разделить на операции в мультипликативной и аддитивной коммутативных группах.

К операциям в аддитивной группе относят:

- *сложение* элементов поля (даны g, $b \in \mathbb{F}_p$, $(g + b) \bmod p \equiv r$, где r — остаток от деления $g + b$ на p, а $0 \leqslant r \leqslant p - 1$);
- *нахождение аддитивно обратного* элемента в поле (дан элемент $g \in \mathbb{F}_p$, $(-g) \bmod p \equiv r$ — это однозначное целое $r \in \mathbb{F}_p$, для которого $(g + (-g) \bmod p \equiv 0)$.

К операциям в мультипликативной группе относят:

— *умножение* элементов поля (даны g, $b \in \mathbb{F}_p$, $(gb) \bmod p \equiv r$, где r — остаток от деления gb на p, $0 \leqslant r \leqslant p - 1$);

— *возведение* в квадрат в поле (дан $g \in \mathbb{F}_p$, $(g)^2 \bmod p \equiv r$, где r — это остаток от деления g^2 на p, $0 \leqslant r \leqslant p - 1$);

— *нахождение мультипликативного обратного* элемента в поле (дан ненулевой элемент $g \in \mathbb{F}_p$, $(g^{-1}) \bmod p \equiv r$ — это однозначное целое $r \in \mathbb{F}_p$, для которого $(gr) \bmod p \equiv 1$).

Кроме обладающих спецификой для каждой из коммутативных групп алгоритмов существуют также алгоритмы редукции, которые позволяют обеспечить замкнутость этого множества.

В основе некоторых алгоритмов, таких как умножение и сложение элементов поля, лежат алгоритмы с соответствующими операциями над большими целыми числами. Таким образом, уровень $\mathcal{L}_2$ имеет связь с уровнем $\mathcal{L}_1$.

Уровень $\mathcal{L}_3$. К третьему уровню архитектуры относят алгоритмы арифметических операций с точкой эллиптической кривой. В их числе:

— *сложение* точек (дана пара точек P, $Q \in E(\mathbb{F}_p)$, $P \boxplus Q = R$, $R \in E(\mathbb{F}_p)$);

— *удвоение* точки (дана точка $P \in E(\mathbb{F}_p)$, $2P = R$, $R \in E(\mathbb{F}_p)$).

Выполнение сложений и удвоений точек в аффинных координатах требует нахождений мультипликативных обратных элементов. Такая операция над простыми полями требует наиболее высоких вычислительных затрат. Другие координатные системы эллиптических кривых позволяют выполнять сложение и удвоение без нахождений обратного элемента в поле. Вычислительная сложность алгоритмов на данном уровне может быть выражена в виде композиции алгоритмов нижестоящего уровня, что характеризует связь с уровнем $\mathcal{L}_2$. Входными данными являются элементы группы точек эллиптической кривой $E(\mathbb{F}_p)$, $n = \lceil \log p \rceil$.

Уровень $\mathcal{L}_4$. К четвертому уровню архитектуры относят алгоритмы операции *скалярного умножения точки*: дана точка $P \in E(\mathbb{F}_p)$ и скаляр $d \in \mathbb{Z}$, $dP = Q$, $Q \in E(\mathbb{F}_p)$. В качестве входных данных используют скаляр d и точку P. Алгоритмы данного уровня используют базовые операции сложения и удвоения точек уровня $\mathcal{L}_3$ для получения композитных операций и кратной точки.

Уровень $\mathcal{L}_5$. К этому уровню архитектуры относят алгоритмы мультискалярного умножения точек. Операцией *мультискалярного умножения* называется преобразование вида

$$\boxplus_{i=1}^{n} k_i Q_i, \text{ где } k_i \in \mathbb{Z}, \ Q_i \in E(\mathbb{F}_p).$$

В основе алгоритмов уровня $\mathcal{L}_5$ лежат алгоритмы уровней $\mathcal{L}_3$ и $\mathcal{L}_4$.

Введение обозначенной выше иерархии алгоритмов предоставляет следующие преимущества:

- позволяет разработчикам программного и аппаратного обеспечения оценить вычислительные затраты преобразований и определить компромиссную реализацию между вычислительными затратами, сложностью реализации и памяти, необходимой для программ и данных;
- позволяет получить независимую оценку сложности вычислений каждого из уровней архитектуры;
- позволяет эффективно применить инкапсуляцию, наследование, полиморфизм, что в значи-

тельной степени упрощает программную разработку больших проектов, поскольку данная архитектура хорошо "ложиться" на модель разработки с использованием объектно-ориентированного подхода в языках высокого уровня;

- позволяет скрыть детали реализации нижестоящих уровней для различных специалистов, участвующих в разработке;
- предоставляет широкие возможности для применения параллельных вычислений, для всех или отдельных уровней архитектуры;
- позволяет выполнить некоторые уровни архитектуры отдельными аппаратными модулями, устанавливая баланс между сложностью и производительностью.

В данной статье особое внимание будет уделено уровням $\mathcal{L}_3$ и $\mathcal{L}_4$.

Пусть $\mathbb{E}$ — множество рассматриваемых кривых $E(\mathbb{F}_p)$, $\mathcal{L}_i$ — множество алгоритмов вычисления преобразований на основе эллиптических кривых уровня i математической иерархии. При $i = 1$ множество алгоритмов $\mathcal{L}_1$ будем называть множеством базовых алгоритмов. Дадим определение алгоритма, получаемого композицией, и установим критерий эффективности алгоритмов в следующих определениях.

Определение 4. *Алгоритм $a \in \mathcal{L}_i$, $i > 1$ получается композицией из t алгоритмов множества $\mathcal{L}_{i-1}$:*

$$\mathcal{F}_a : \mathcal{L}_{i-1}^t \to \mathcal{L}_i, \quad i = \overline{2,5}.$$

Определение 5. *Архитектура алгоритмов представляет собой математическую иерархию. Множество алгоритмов всех уровней архитектуры обозначается $\mathcal{A}$:*

$$\mathcal{A} = \bigcup_{i=\overline{1,5}} \mathcal{L}_i.$$

Определение 6. *Вычислительной (средней вычислительной) сложностью алгоритма a, является функция $L_a(n)$ $\left(\hat{L}_a(n)\right)$, определяющая зависимость времени (среднего времени) работы алгоритма a от входного параметра n.*

Определение 7. *Эффективным алгоритмом $a \in \mathcal{L}_i$, полученным композицией $\mathcal{F}_a$, является такой алгоритм, что:*

$$L_a(n) = L^i_{\min}(n),$$

$$\text{при этом } L^i_{\min} = \min_{c \in \mathcal{L}_i} \{L_c(n)\}.$$

3. Эффективный алгоритм на основе метода бинарной несовместной формы представления скаляра

Знаковое бинарное представление является избыточной системой для представления целого числа. Для целого d стандартным представлением по основанию 2 является запись вида

$$d = \sum_{i=0}^{n-1} d_i 2^i.$$

Данное представление является единственным. Знаковое бинарное представление предполагает, что $d_i \in \{-1, 0, 1\}$. Пусть $(d_{n-1}, ..., d_0)$ обозначает последовательность чисел, представляющую целое d, где $d_{n-1} \neq 0$. Свойства NAF-представления сформулированы следующей теоремой.

Теорема 2 (В. Босма [7]). *Пусть $d \in \mathbb{Z}^+$, тогда:*

- *значение d имеет уникальное NAF-представление, обозначаемое как NAF(d);*
- *разложение NAF(d) имеет наименьшее число ненулевых значений любого знакового бинарного представления d;*
- *длина представления NAF(d) по меньшей мере на единицу больше бинарного представления d;*
- *если длина NAF(d) равна n, то $\frac{2^n}{3} < d < \frac{2^{n+1}}{3}$;*
- *средняя плотность ненулевых значений среди всех NAF-представлений длины n равна $\frac{n}{3}$.*

Поскольку $d_{n-1} = 1$ при NAF-представлении положительного целого скаляра d, то $d_{n-2} = 0$. Вычисление NAF-представления скаляра d осуществляется с использованием алгоритма, основанного на алгоритме, предложенном Д. Хенкерсоном [12] (далее — алгоритм 1).

Алгоритм 1. Вычисление бинарного NAF-представления скаляра $d \in \mathbb{Z}^+$ (рис. 2).

Вход:

— скаляр $d \in \mathbb{Z}^+$.

Выход:

— *NAF*(d) в виде знакового бинарного представления.

Другое знаковое представление было предложено К. Окия (Katsuyuki Okeya). Оно является взаимной противоположной формой представления скаляра (MOF, *mutual opposite form*). Такое представление имеет такие же свойства, что и бинарная несовместная форма представления. Главное отличие представления Окия заключается в том, что изменено направление прохода алгоритма.

```
    begin
1.      i ← 0
2.      while (d ≥ 1) {
3.          if (d mod 2 ≡ 0) {
4.              d_i ← 2 − (d mod 4)
5.              d ← d − d_i
6.          } else {
7.              d_i ← 0
8.          }
9.          d ← d/2
10.         i ← i + 1
11.     }
12.     return(d_{n−1}, ..., d_0)
    end
```

Рис. 2. Программная реализация алгоритма *NAF*(d), $d \in \mathbb{Z}^+$

```
    begin
1.      Q ← dbl(P)
2.      i ← n − 3
3.      while (i ≥ 0) {
4.          if (d_i = 1) {
5.              Q ← da(Q, P)
6.          } else {
7.              if (d_i = −1) {
8.                  Q ← da(Q, −P)
9.              } else {
10.                 Q ← dbl(Q)
11.             }
12.         }
13.         i ← i − 1
14.     }
15.     return(Q)
    end
```

Рис. 3. Программная реализация алгоритма эффективного вычисления скалярного умножения точки dP, $d \in \mathbb{Z}$, $P \in E(\mathbb{F}_p)$ на основе метода бинарного несовместного представления скаляра

При бинарной несовместной форме представления скаляра операция $2P \boxplus Q$ либо $2P$ выполняется для каждого значения в представлении. Если допустить использование значения -1, то выполняемые операции будут иметь вид $2P \boxplus Q$, $2P$ или $2P \boxminus Q$. Тот факт, что получение отрицательного значения $\boxminus Q$ является быстрой операцией означает, что $2P \boxminus Q$ вычисляется с затратами, сравнимыми с $2P \boxplus Q$, т. е. оценки вычислительной сложности операций $2P \boxplus Q$ и $2P \boxminus Q$ будут отличаться на константу. Алгоритм 2 позволяет вычислить скалярное умножение точки эллиптической кривой по NAF-представлению скаляра.

Алгоритм 2. Эффективное вычисление скалярного умножения точки dP, $d \in \mathbb{Z}$, $P \in E(\mathbb{F}_p)$ на основе метода бинарной несовместной формы представления скаляра d (рис. 3).

Вход:

— эллиптическая кривая $E(\mathbb{F}_p)$;

— точка $P \in E(\mathbb{F}_p)$, $P = (x, y)$, $x, y \in \mathbb{F}_p$;

— представление $NAF(d) = (d_{n-1}, ..., d_0)$.

Выход:

— результат скалярного умножения $dP = (x, y)$, $x, y \in \mathbb{F}_p$.

Для получения оценки средней вычислительной сложности предложенного алгоритма, сформулируем и докажем следующую теорему.

Теорема 3. *Алгоритм a_{NAF} (алгоритм 2) вычисления скалярного умножения точки dP, $d \in \mathbb{Z}$, $P \in E(\mathbb{F}_p)$, $d = (d_{n-1}, ..., d_0)_2$, $d_i \in \{-1, 0, 1\}$ имеет среднюю вычислительную сложность*

$$\hat{L}_{a_{NAF}}(n) = \mathbf{I} + \left(\frac{17}{3}n - 10\right)\mathbf{M} + \left(\frac{21}{3}n - 8\right)\mathbf{S}.$$

Доказательство. Средняя вычислительная сложность предложенного алгоритма a_{NAF} на основе метода бинарной несовместной формы представления скаляра (NAF) (1):

$$\hat{L}_{a_{NAF}} = \left(\frac{n}{3} - 1\right) ADD + (n-1)\, DBL,$$

где $n \in \mathbb{Z}^+$ — это длина разложения скаляра $d \in \mathbb{Z}$.

В алгоритме a_{NAF} приблизительно в три раза больше удвоений (DBL), чем сложений (ADD). Система координат Якоби является наиболее эффективной для выполнения удвоений.

На шаге $Q \leftarrow 2P$ алгоритма выполняется $2\mathcal{A} \to \mathcal{J}$, на шаге $Q \leftarrow 2Q \boxplus P$ и $Q \leftarrow 2Q \boxminus P$ выполняется $2\mathcal{J} \boxplus \mathcal{A} \to \mathcal{J}$, на шаге $Q \leftarrow 2Q$ выполняется $2\mathcal{J} \to \mathcal{J}$. Результат работы алгоритма представляется в координатах Якоби, поэтому необходимо перевести его в аффинные координаты $\mathcal{J} \to \mathcal{A}$ ($\mathcal{A}$ и $\mathcal{J}$ обозначают представление точки в аффинных и Якоби-координатах соответственно). Операция $c \to \tilde{c}$ обозначает преобразование и сохранение точки из координат c в $\tilde{c}$, если c не совпадают с $\tilde{c}$, и только сохранение, если совпадают.

Тогда, для выполнения алгоритма a_{NAF} (алгоритм 1), необходимо в среднем следующее количество операций в поле:

$$\hat{L}_{a_{NAF}} = \left(\frac{n-3}{3}\right)(13\mathbf{M} + 5\mathbf{S}) + (2\mathbf{M} + 4\mathbf{S}) +$$
$$+\left(n - 2 - \frac{n-3}{3}\right)(2\mathbf{M} + 8\mathbf{S}) + (\mathbf{I} + 3\mathbf{M} + \mathbf{S}) =$$
$$= \mathbf{I} + \left(\frac{17}{3}n - 10\right)\mathbf{M} + \left(\frac{21}{3}n - 8\right)\mathbf{S}.$$

Другая известная лучшая средняя оценка $\hat{L}_{a^*_{NAF}}(n)$ вычислительной сложности алгоритма a^*_{NAF} умножения точки на основе метода бинарного несовместного представления скаляра d, предложенного Д. Гордоном [10], получена Н. Суливаном [19] и составляет (2)

$$\hat{L}_{a^*_{NAF}}(n) = \mathbf{I} + \left(\frac{20}{3}n - 11\right)\mathbf{M} + \left(\frac{15}{3}n - 6\right)\mathbf{S}.$$

Таблица 3

Средняя вычислительная сложность предлагаемого (a_{NAF}) и известного $\left(a^*_{NAF}\right)$ алгоритмов на множестве эллиптических кривых $\mathbb{E}$

$\mathbb{E}$	Известный алгоритм, $\hat{L}_{a^*_{NAF}}(n)$	Предлагаемый алгоритм, $\hat{L}_{a_{NAF}}(n)$
P_{192}	**I** + 1269,0**M** + 954,0**S**	**I** + 1078,0**M** + 1336,0**S**
P_{224}	**I** + 1482,3**M** + 1114,0**S**	**I** + 1259,3**M** + 1560,3**S**
P_{256}	**I** + 1695,7**M** + 1274,0**S**	**I** + 1440,7**M** + 1784,7**S**
P_{384}	**I** + 2569,0**M** + 1914,0**S**	**I** + 2166,0**M** + 2680,0**S**
P_{521}	**I** + 3462,3**M** + 2599,0**S**	**I** + 2942,3**M** + 3649,0**S**

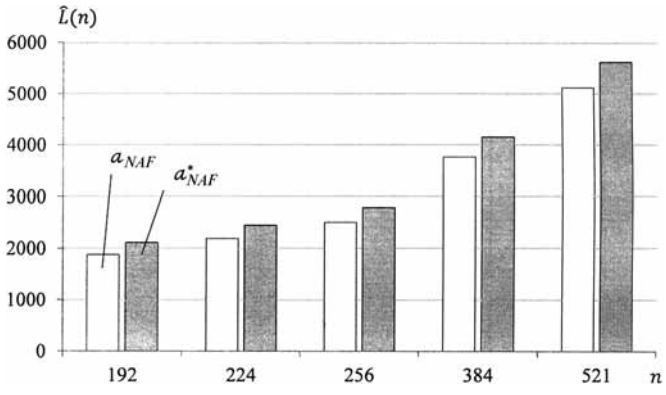

Рис. 4. Сравнение средней вычислительной сложности предлагаемого (a_{NAF}) и известного (a^*_{NAF}) алгоритмов скалярного умножения точки на множестве эллиптических кривых $\mathbb{E}$

Этот результат был достигнут за счет использования метода бинарной несовместной формы представления скаляра, путем применения эффективной композитной операции *DA* и операции *DBL* в аффинных координатах и координатах Якоби, а также с учетом свойств координатных систем.

В табл. 3 и на рис. 4 представлена сравнительная оценка средней вычислительной сложности нахождения кратной точки предлагаемым алгоритмом a_{NAF} и лучшим из известных на настоящее время алгоритмом a^*_{NAF} для рассматриваемых подмножеств $\mathbb{E}$. Для вычисления кратной точки на эллиптической кривой алгоритмом на основе NAF-метода не предъявляются требования к выделению дополнительной памяти. Благодаря этому качеству предлагаемый алгоритм является образующим для скалярного умножения без предварительных вычислений.

За счет замены умножений возведениями в квадрат при получении быстрого сложения удалось снизить вычислительную сложность алгоритма. При соотношении констант, связанных с возведением в квадрат ($c_{\mathbf{S}}$) и умножением ($c_{\mathbf{M}}$) в поле $\mathbb{F}_p$ как $c_{\mathbf{S}} \cong 0{,}6\ c_{\mathbf{M}}$, на рис. 4 демонстрируется эффективность предлагаемого алгоритма a_{NAF} на множестве эллиптических кривых $\mathbb{E}$.

Заключение

Представлен новый алгоритм вычисления скалярного умножения точки эллиптической кривой на основе NAF-метода. Приведен комплексный анализ этого алгоритма и представлена оценка его вычислительной сложности, доказана эффективность такого алгоритма относительно других, существующих на настоящее время алгоритмов на основе NAF-метода. Полученные результаты могут быть использованы разработчиками для выбора оптимальной конфигурации аппаратных средств в условиях их объективных ограничений по памяти и вычислительным ресурсам, а также для ускорения работы алгоритмов факторизации натуральных чисел и тестирования чисел на простоту.

Практическая значимость исследования состоит в том, что полученный результат можно использовать для создания программного и аппаратного обеспечения, позволяющего повысить эффективность обработки данных в ходе их преобразований, основанных на эллиптических кривых.

Список литературы

1. **ГОСТ Р 34.10—2012.** Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи. М.: Стандартинформ, 2012.
2. **ANSI X9.62:2005.** Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), Accredited Standards Committee on Financial Services, X9, 2005.
3. **Avanzi R.** Aspects of Hyperelliptic Curves over Large Prime Fields //Software Implementations in Workshop on Cryptographic Hardware and Embedded Systems (CHES'04). Springer-Verlag, 2004. Vol. 3156 of Lecture Notes in Computer Science. P. 148—162.
4. **Bernstein D.** Curve25519: New Diffie-Hellman Speed Records. URL: http://cr.yp.to/talks.html (дата обращения 12.10.2015).
5. **Bernstein D.** High-Speed Diffie-Hellman, Part 2. Presentation in INDOCRYPT'06, tutorial session, 2006. URL: http://cr.yp.to/talks/2006.12.10-2/slides.pdf.
6. **Brown M., Hankerson D., Lopez J., Menezes A.** Software Implementation of the NIST elliptic curves over prime fields // Progress in Cryptology CT-RSA, 2001. Springer-Verlag, 2001. Vol. 2020 of Lecture Notes in Computer Science. P. 250—265.
7. **Bosma W.** Signed bits and fast exponentiation, J-Theory // Nombres Bordeaux 2001. Vol. 13, N 1. P. 27—41.
8. **FIPS 186-2.** Digital Signature Standard (DSS). National Institute of Standards and Technology, 2000.
9. **Gebotys C. H., Gebotys R. J.** Secure Elliptic Curve Implementations: Analysis of Resistance to Power-Attacks in a DSP Processor // Workshop on Cryptographic Hardware and Embedded Systems (CHES'03). Springer-Verlag, 2003. Vol. 2523 Lecture Notes in Computer Science. P. 114—128.
10. **Gordon D.** A Survey of Fast Exponentiation Methods // Journal of Algorithms, 1998. Vol. 27. P. 129—146.
11. **Gura N., Patel A., Wander A.** et al. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs // Workshop on Cryptographic Hardware and Embedded Systems (CHES'04). Springer-Verlag, 2004. Vol. 3156 of Lecture Notes in Computer Science. P. 119—132.
12. **Hankerson D., Menezes A., Vanstone S.** Guide to Elliptic Curve Cryptography, Springer-Verlag New York, Inc., 2004. 311 p.
13. **Koblitz N**. Introduction to Elliptic Curves and Modular Forms. New York, Springer-Verlag, 1984. 250 p.
14. **Lim C. H., Hwang H. S.** Fast implementation of Elliptic Curve Arithmetic in GF // Proc. PKC'00, Lecture Notes in Computer Science. Springer-Verlag, 2000. Vol. 1751. P. 405—421.
15. **Meloni N.** Fast and Secure Elliptic Curve Scalar Multiplication over Prime Fields using Special Addition Chains. Cryptology ePrint Archive, Report 2006/216, 2006.
16. **NIST PUB 800-57.** Recommendation for Key Management — Part 1: General (Revision 3), National Institute of Standards and Technology. 2012.
17. **Okeya K., Schmidt-Samoa K., Spahn C., Takagi T.** Signed binary representations revisited // Advances in Cryptology — CRYPTO 2004, Lecture Notes in Computer Science. 2004. Vol. 3152. P. 123—139.
18. **SEC 1:** Elliptic Curve Cryptography. Standards for Efficient Cryptography. Certicom Corp., 2009.
19. **Sullivan N. T.** Fast Algorithms for Arithmetic on Elliptic Curves over Prime Fields. University of Calgary, Calgary, Alberta, Cryptology ePrint Archive. 2007.

Efficient Algorithm of Scalar Point Multiplication on Elliptic Curve Based on NAF-method

D. S. Khleborodov, dkhleborodov@gmail.com, Bauman Moscow State Technical University, Moscow, 105005, Russian Federation

Corresponding author:

Khleborodov Denis S., Postgraduate Student, Bauman Moscow State Technical University, Moscow, 105005, Russian Federation, e-mail: dkhleborodov@gmail.com

Received on October 13, 2015
Accepted on October 27, 2015

The issue of this paper — efficient algorithms for calculating transformations based on elliptic curves - is a relatively new area of research. Transformations based on elliptic curves are widely used in public-key cryptosystems, and are also used in cryptanalysis of asymmetric ciphers based on integer factorization problem. Efficiency of point scalar multiplication on an elliptic curve defined over the prime fields on the basis of the binary Non-Adjacent Form representation of the scalar was also investigated. To assess effectiveness of the received and previously proposed algorithms the criterion of effectiveness, based on the average computational complexity was introduced. For new algorithms the effective operation in prime field, operations with point (addition and doubling); composite operation "double and add"; scaling operation; affine coordinate system properties; properties of the coordinate system of Jacobi were used. Formulated and proved assertion regarding computational complexity for the proposed algorithm.

Keywords: *elliptic curves, ECC, fast algorithms, scalar multiplication, point multiplication, NAF, non-adjacent form*

For citation:

Khleborodov D. S. Efficient Algorithm of Scalar Point Multiplication on Elliptic Curve Based on NAF-method, *Programmnaya Ingeneria,* 2016, vol. 7, no 1, pp. 21—28.

DOI: 10.17587/prin.7.21-28

References

1. **GOST R 34.10—2012.** Informacionnaja tehnologija. Kriptograficheskaja zashhita informacii. Processy formirovanija i proverki jelektronnoj cifrovoj podpisi (Information technology. Cryptographic data security. Signature and verification processes of digital signature), Moscow, Standartinform, 2012 (in Russian).

2. **ANSI X9.62:2005.** Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), Accredited Standards Committee on Financial Services, X9, 2005.

3. **Avanzi R.** Aspects of Hyperelliptic Curves over Large Prime Fields, *Software Implementations in Workshop on Cryptographic Hardware and Embedded Systems (CHES'04)*. Springer-Verlag, 2004, vol. 3156 of Lecture Notes in Computer Science, pp. 148—162.

4. **Bernstein D.** Curve25519: New Diffie-Hellman Speed Records, available at: http://cr.yp.to/talks.html (date of access 12.10.2015).

5. **Bernstein D.** High-Speed Diffie-Hellman, Part 2, *Presentation in INDOCRYPT'06*, tutorial session, 2006, available at: http://cr.yp.to/talks/2006.12.10-2/slides.pdf.

6. **Brown M., Hankerson D., Lopez J., Menezes A.** Software Implementation of the NIST elliptic curves over prime fields, *Progress in Cryptology CT-RSA*, 2001, Springer-Verlag, 2001, vol. 2020 of Lecture Notes in Computer Science, pp. 250—265.

7. **Bosma W.** Signed bits and fast exponentiation, J-Theory, *Nombres Bordeaux* 2001, vol. 13, no. 1, pp. 27—41.

8. **FIPS 186-2.** Digital Signature Standard (DSS). National Institute of Standards and Technology, 2000.

9. **Gebotys C. H. and Gebotys R. J.** Secure Elliptic Curve Implementations: Analysis of Resistance to Power-Attacks in a DSP Processor. *Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, Springer-Verlag, 2003, vol. 2523, Lecture Notes in Computer Science, pp. 114—128.

10. **Gordon D.** A Survey of Fast Exponentiation Methods, *Journal of Algorithms*, 1998, vol. 27, pp. 129—146.

11. **Gura N., Patel A., Wander A., Eberle H., Shantz S. C.** Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs, *Workshop on Cryptographic Hardware and Embedded Systems (CHES'04)*, Springer-Verlag, 2004, vol. 3156 of Lecture Notes in Computer Science, pp. 119—132.

12. **Hankerson D., Menezes A., Vanstone S.** Guide to Elliptic Curve Cryptography, Springer-Verlag New York, Inc., 2004, 311 p.

13. **Koblitz N.** Introduction to Elliptic Curves and Modular Forms, Springer-Verlag, New York, 1984, 250 p.

14. **Lim C. H., Hwang H. S.** Fast implementation of Elliptic Curve Arithmetic in GF. *Proc. PKC'00,* 2000 Lecture Notes in Compuret Science, vol. 1751, pp. 405—421, Springer-Verlag.

15. **Meloni N.** Fast and Secure Elliptic Curve Scalar Multiplication over Prime Fields using Special Addition Chains. *Cryptology ePrint Archive*, Report 2006/216, 2006.

16. **NIST PUB 800-57.** Recommendation for Key Management — Part 1: General (Revision 3), National Institute of Standards and Technology, 2012.

17. **Okeya K., Schmidt-Samoa K., Spahn C., Takagi T.** Signed binary representations revisited, *Advances in Cryptology — CRYPTO 2004*, 2004, vol. 3152, Lecture Notes in Computer Science, pp. 123—139.

18. **SEC 1**: Elliptic Curve Cryptography. Standards for Efficient Cryptography, Certicom Corp., 2009.

19. **Sullivan N. T.** Fast Algorithms for Arithmetic on Elliptic Curves over Prime Fields, University of Calgary, Calgary, Alberta, Cryptology ePrint Archive, 2007.

УДК 519.72 DOI: 10.17587/prin.7.29-36

А. М. Кукарцев, ст. препод., e-mail: amkukarcev@yandex.ru,
А. А. Кузнецов, д-р физ.-мат. наук, проф., e-mail: kuznetsov@sibsau.ru,
Сибирский государственный аэрокосмический университет
имени академика М. Ф. Решетнева, г. Красноярск

О действиях группы Джевонса на множествах бинарных векторов и булевых функций для инженерно-технических решений обработки информации[1]

*Действия группы Джевонса на множестве бинарных векторов и булевых функций позволяют строить инженерно-технические решения для представления и обработки информации. Настоящая статья является продолжением работы [1] о конструктивном представлении группы Джевонса для построения инженерно-технических решений обработки информации. В настоящей статье приведены правила вычисления и коммутации действий на множествах булевых векторов и булевых функций. Введено понятие эквивалентных действий (эквиморфизмов) на множестве булевых функций. Эквиморфизмы позволяют существенно повысить производительность программных, аппаратных и программно-аппаратных реализаций. Рассматриваемый в статье материал является ядром разработанной библиотеки программных функций "domain operations processor" (или **dop**), в которой реализованы указанные правила вычисления действий. Библиотека распространяется под лицензией GNU Lesser General Public License.*

Ключевые слова: *представление информации, группа Джевонса, действие группы на множестве, бинарные векторы, булевы функции, вычислительные средства*

Введение

Настоящая работа является продолжением статьи "О конструктивном представлении группы Джевонса для построения инженерно-технических решений обработки информации" [1]. Цикл посвящен описанию и обоснованию методов обработки информации [2], представляемой в форме комбинации связанных (зависимых) символов того или иного алфавита и отождествляемой с некоторой математической функцией [3]. Причем символы являются значениями функции.

В работе [1] рассматривался один из основных объектов для работы с этой формой информации — группа Джевонса. В настоящей статье описаны непосредственно действия группы Джевонса на множестве бинарных векторов и булевых функций (подробнее о булевых функциях см. в работе [4]).

Суть представления информации заключается в следующем. Бинарный вектор (далее БВ), длина которого кратна n-й степени двойки, сопоставляется с булевой функцией (далее БФ) n аргументов. При этом БВ является столбцом значений БФ и задано соответствие между аргументами БФ и номерами бит БВ. Если зафиксировать порядок следования аргументов БФ, то сами они рассматриваются как БВ и как двоичные числа. Последнее позволяет связать БФ и БВ.

После такого связывания можно анализировать и преобразовывать БВ, применяя методы теории БФ. Нетрудно видеть, что появляется естественная эквивалентность на множестве БВ [4—6]. Если переставить и/или инвертировать аргументы БФ, то ее формула (как упорядоченная комбинация булевых операций) неизменна. Откуда из эквивалентности формул БФ следует эквивалентность БВ. Работа с информацией, представленной как БФ, сложна вследствие связей между объектами: бинарным вектором, булевой функцией, группой Джевонса и ее изоморфными вложениями в симметрическую группу степени 2^n [7].

Цель настоящей статьи — продолжить описание математического аппарата действий группы Дже-

[1] Работа выполнена при поддержке гранта Президента РФ (проект МД-3952.2015.9).

вонса на множествах БВ и БФ, необходимого для разработки соответствующих инженерно-технических решений (ИТР), для работы с представлением информации как с БФ. В основной части статьи определены правила действия группы Джевонса на множествах БВ и БФ, правила их коммутации; введено понятие эквивалентного действия на множестве БФ посредством действия изоморфного вложения группы Джевонса в подгруппу симметрической группы. Такие эквивалентные действия позволяют существенно повысить производительность реализации ИТР.

Предлагаемые правила были успешно применены для решения некоторых уравнений действия группы Джевонса на множестве БФ, рассматриваемых в работах [6, 8, 9]. Без использования данных правил решение таких уравнений тривиальными алгоритмами требует экспоненциального времени работы. Назначение статьи — представление решения технических проблем реализации [10] действий группы Джевонса на множестве БВ и БФ. Поэтому решения указанных уравнений не приводятся.

Перед изложением основного материала дадим в форме тезисов ядро предыдущей статьи (только формулировки лемм, теорем и ключевые формулы). Рассматривается два семейства действий: тип А и тип Б. Необходимость рассмотрения обоих семейств обусловлена их равнозначностью и сложностью перехода от одного семейства к другому в результирующих соотношениях. Разработчик ИТР может использовать тип семейства, исходя из своих предпочтений.

Пусть $E = \{0, 1\}$ — множество, состоящее из двух элементов. Декартовое произведение этого множества на себя определим как $E^n = \underbrace{E \times \cdots \times E}_{n}$. Под бинарным вектором длины n будем понимать элемент множества E^n. Для координат множителей будем использовать классическую числовую нотацию L2R (left to right), т. е. большие координаты находятся левее. Обозначим произвольный элемент такого множества как $x \in E^n : x = \{x_{n-1}, \ldots, x_i, \ldots, x_0\}$. Зададим операцию над элементами такого множества как сложение соответственных координат по модулю два и будем обозначать такую группу как E_n. Подстановкой [11] степени n будем называть объект вида $\pi = \begin{pmatrix} n-1 & \ldots & 0 \\ i_{n-1} & \ldots & i_0 \end{pmatrix}$. Причем в верхней и нижней строках находятся различные числа от 0 до $n - 1$. Нотация подстановок выбрана таким образом, чтобы соответствовать индексам БВ. Всю группу подстановок обозначим как S_n. Определим $0 \leqslant i < n$, $k = 2^n$, $0 \leqslant j < k$ — целые неотрицательные числа, являющиеся индексами-счетчиками.

Используемые в настоящей статье символика и нотация однозначно соответствуют формату входных и/или выходных данных библиотеки **dop**. Здесь и далее по тексту, а также в библиотеке **dop** фигурные скобки применяют для обозначения БВ, а круглые — для подстановок. Это необходимо для семантического разделения сущностей при эксплуатации **dop**. Далее приведем тезисы статьи [1].

$$x'_{\pi(i)} = x_i; \tag{1.А}$$

$$x'_i = x_{\pi(i)}. \tag{1.Б}$$

Формулы (1.А) и (1.Б) определяют правила действия симметрической группы на БВ. Лемма 1.А и лемма 1.Б показывают, как вычисляются композиции действий по формулам (1.А) и (1.Б) соответственно.

Лемма 1.А (О действии типа А). Если подстановки $\pi, \rho \in S_n$ действуют последовательно (π и затем ρ) на элемент $x \in E^n$, то результат действия эквивалентен действию подстановки-произведения $\pi\rho$ или в символьном виде $\left(x^{\pi}\right)^{\rho} = x^{(\pi\rho)}$.

Лемма 1.Б (О действии типа Б). Если подстановки $\pi, \rho \in S_n$ действуют последовательно (π и затем ρ) на элемент $x \in E^n$, то результат действия эквивалентен действию подстановки-произведения $\rho\pi$ или в символьном виде $\left(x^{\pi}\right)^{\rho} = x^{(\rho\pi)}$.

Для элементов n_0, n_1 группы N и элементов h_0, h_1 группы H покажем конструкцию полупрямого группового произведения в общем виде:

$$(n_0, h_0) \circ (n_1, h_1) = \left(n_0 \psi_{h_0}(n_1), h_0 h_1\right). \tag{2}$$

Для (2) верно, что подгруппа N есть нормальный делитель произведения $N \leftthreetimes H$, а ψ — гомоморфизм из H в группу автоморфизмов N. Элементы множества БВ биективно отображаются на целые неотрицательные числа:

$$\forall x \in E^n, \exists! x \in Z_+ : x = \sum_{i=n-1}^{0} x_i 2^i. \tag{3}$$

Условимся не различать по начертанию вектор и целое неотрицательное число, так как по сути двоичное число и есть БВ.

Описание группы Джевонса начнем с автоморфизма группы E_n посредством симметрической группы (теорема 1).

Теорема 1 (Об автоморфизме). Отображение $E_n^{\pi} \to E_n, \forall \pi \in S_n$ есть автоморфизм независимо от типа преобразования А или Б.

Действия групп E_n и S_n как над целым неотрицательным числом (3) зададим в следующем виде:

$$x^z = \sum_{i=n-1}^{0} (x_i \oplus z_i) 2^i; \tag{4}$$

$$x^{\pi} = \sum_{i=n-1}^{0} x_{\pi^{-1}(i)} 2^i = \sum_{i=n-1}^{0} x_i 2^{\pi(i)}; \tag{5.А}$$

$$x^{\pi} = \sum_{i=n-1}^{0} x_{\pi(i)} 2^i = \sum_{i=n-1}^{0} x_i 2^{\pi^{-1}(i)}. \tag{5.Б}$$

Это позволяет перейти к подстановкам симметрической группы степени k:

$$\varphi_z(j) = j^z, \forall z \in E_n; \quad (6)$$

$$\varphi_\pi(j) = j^\pi, \forall \pi \in S_n. \quad (7)$$

Множество таких подстановок образуют подгруппы в симметрической группе S_k, и имеют место изоморфные (антиизоморфные) вложения из E_n и S_n в S_k. Следующие две теоремы доказывают эти вложения.

Теорема 2 (Об изоморфизме B_n). Отображение вида (6) $E_n \to B_n$, $z \in E_n$, $\varphi_z \in B_n : \varphi_z(j) = j^z$ есть изоморфизм.

$$\varphi_z = \begin{pmatrix} 2^n - 1 & \dots & j & \dots & 0 \\ \sum_{i=n-1}^{0} \overline{z_i} 2^i & \dots & \sum_{i=n-1}^{0} (j_i \oplus z_i) 2^i & \dots & \sum_{i=n-1}^{0} z_i 2^i \end{pmatrix}. \quad (8)$$

Теорема 3 (Об изоморфизме T_n). Отображение вида (7) $S_n \to T_n$, $\pi \in S_n$, $\varphi_\pi \in T_n : \varphi_{\pi(j)} = j^\pi$ есть изоморфизм для типа преобразования А и антиизоморфизм для типа преобразования Б.

$$\varphi_\pi = \begin{pmatrix} 2^n - 1 & \dots & j & \dots & 0 \\ 2^n - 1 & \dots & \sum_{i=n-1}^{0} j_i 2^{\pi(i)} & \dots & 0 \end{pmatrix}; \quad (9.\text{А})$$

$$\varphi_\pi = \begin{pmatrix} 2^n - 1 & \dots & j & \dots & 0 \\ 2^n - 1 & \dots & \sum_{i=n-1}^{0} j_{\pi(i)} 2^i & \dots & 0 \end{pmatrix}. \quad (9.\text{Б})$$

При этом изоморфизмами (антиизоморфизмами) будут отображения (8), (9.А) и (9.Б) соответственно. Изоморфные вложения E_n и S_n в S_k образуют внутреннее полупрямое произведение, которое является изоморфным вложением группы Джевонса.

Теорема 4 (Об изоморфизме β_n). $\beta_n = B_n T_n$, образованная как теоретико-множественное произведение групп B_n и T_n, есть группа, которая является внутренним полупрямым произведением $B_n \leftthreetimes T_n$.

$$(z_0\pi_0)(z_1\pi_1) = \left(z_0 z_1^{\pi_0^{-1}} \pi_0\pi_1\right); \quad (10.\text{А})$$

$$(z_0\pi_0)(z_1\pi_1) = \left(z_0 z_1^{\pi_0^{-1}} \pi_1\pi_0\right). \quad (10.\text{Б})$$

При этом операция в группе Джевонса (внешнем полупрямом произведении) будет задана как (10.А) и (10.Б).

$$(z\pi)^{-1} = \left(\psi_{\pi^{-1}}\left(z^{-1}\right)\pi^{-1}\right) = \left(\left(z^{-1}\right)^{(\pi^{-1})^{-1}} \pi^{-1}\right) = \left(z^\pi \pi^{-1}\right); \quad (11)$$

$$(z_0\pi_0)^{(z_1\pi_1)} = \left(\left(z_0 z_1 z_1^{\pi_0^{-1}}\right)^{\pi_1} \pi_0^{\pi_1}\right); \quad (12.\text{А})$$

$$(z_0\pi_0)^{(z_1\pi_1)} = \left(\left(z_0 z_1 z_1^{\pi_0^{-1}}\right)^{\pi_1} \pi_0^{\pi_1^{-1}}\right). \quad (12.\text{Б})$$

Для такого представления группы Джевонса будут верны правила вычисления обратных (11) и сопряженных элементов (12.А) и (12.Б) соотвественно.

Действие группы Джевонса на множестве бинарных векторов

Группы E_n, S_n и D_n заданы. Действие первых двух групп на БВ (в том числе как на целое неотрицательное число (3)) определено как (1.А), (1.Б), (4), (5.А), (5.Б). Тогда определим действие группы Джевонса D_n на БВ. Пусть $(z\pi) \in D_n$ действует на БВ $x \in E^n$. Для определенности примем, что сначала действует элемент z нормального делителя E_n подгруппы группы Джевонса, а затем подстановка

$$x^{(z\pi)} = \left(x^z\right)^\pi. \quad (13)$$

Большинство последующих выводов включают действие по формулам (1.А) и (1.Б) абстрактно. Под "абстрактно" понимается свойство виртуальной полиморфности [12] при наследовании классов в объектно-ориентированном программировании. Другими словами, независимо от типа преобразований приведенные формулы допускают полиморфное обобщение (виртуализацию [12]), но конкретные расчетные формулы должны определяться при задании типа преобразования. Выражение (13) абстрактно и примет различные формы для типов А и Б, оно является определением, а поэтому не может быть истинно или ложно.

Лемма 2 (О композиции действий над бинарным вектором). Последовательное действие двух элементов группы Джевонса $(z_0\pi_0), (z_1\pi_1) \in D_n$ на бинарный вектор $x \in E^n$ эквивалентно одному действию произведения этих элементов в исходном порядке по внутригрупповой операции, т. е. $\left(x^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = x^{(z_0\pi_0)(z_1\pi_1)}$.

Доказательство. На основе выражения (13) перейдем к действию на вектор множителями группы Джевонса $\left(x^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = \left(\left(\left(x^{z_0}\right)^{\pi_0}\right)^{z_1}\right)^{\pi_1} = \left(\left(x \oplus z_0\right)^{\pi_0} \oplus z_1\right)^{\pi_1}$. По теореме 1 $\left(x^{\pi_0} \oplus z_0^{\pi_0} \oplus \left(z_1^{\pi_0^{-1}}\right)^{\pi_0}\right)^{\pi_1} = \left(\left(x \oplus z_0 \oplus z_1^{\pi_0^{-1}}\right)^{\pi_0}\right)^{\pi_1}$.

Во внутренних скобках получен элемент E_n, а внешние скобки необходимо раскрыть по типу А или Б согласно лемме 1.А или лемме 1.Б соответственно. Для типа А: $\left(x^{z_0 z_1^{\pi_0^{-1}}}\right)^{\pi_0\pi_1} = x^{\left(z_0 z_1^{\pi_0^{-1}} \pi_0\pi_1\right)}$ и для типа Б: $\left(x^{z_0 z_1^{\pi_0^{-1}}}\right)^{\pi_1\pi_0} = x^{\left(z_0 z_1^{\pi_0^{-1}} \pi_1\pi_0\right)}$, что является определением операции в группе Джевонса согласно формулам (10.А) и (10.Б). Для обоих типов А и Б получим $\left(x^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = x^{(z_0\pi_0)(z_1\pi_1)}$. Что и требовалось доказать.

Утверждение леммы 2 абстрактно, т. е. конкретные расчетные формулы необходимо выводить согласно формулам (10.А) и (10.Б) и далее согласно (1.А) и (1.Б) соответственно. Обратите внимание, что действие элементов симметрической группы для типа Б переставляет операнды местами в произведении (см. лемму 1.Б). В свою очередь, действие элементов группы Джевонса не приводит к перестановке операндов в произведении.

В практических задачах важен вопрос о результирующем действии нескольких элементов различных множителей (БВ и подстановок) группы Джевонса на БВ в произвольном порядке. Например, если сначала на вектор подействовала подстановка, а затем БВ, и требуется найти результат и правила определения эквивалентного по действию на БВ одного элемента группы Джевонса, т. е. правила коммутации элементов множителей группы Джевонса.

Выражение (13) однозначно определяет последовательное действие элементов групп E_n и S_n. Тогда сформулируем правила коммутации для такого порядка. Согласно выражениям (10.А) и (10.Б) верно, что $(z\pi)=(ze_S)(e_E\pi)$, где e_S и e_E нейтральные элементы групп S_n и E_n соответственно. С помощью сопряжений выполним перестановку с сохранением в неизменном виде первого и второго множителей.

Для первого множителя: $(ze_S)(e_E\pi)=(e_E\pi)^{(ze_S)^{-1}}(ze_S)$, далее по формуле (11) раскрываем отрицание $(e_E\pi)^{\left(z^{e_S}e_S^{-1}\right)}(ze_S)=(e_E\pi)^{(ze_S)}(ze_S)$. Сопряжение раскрывается для типов А и Б неодинаково, даже абстрактно. Но, так как $e_S^{-1}=e_S$, то верно, что выражения (12.А) и (12.Б) абстрактно совпадают, откуда по формуле (12.А) получим

$$(e_E\pi)^{(ze_S)}(ze_S)=\left(\left(e_Ezz^{\pi^{-1}}\right)^{e_S}\pi^{e_S}\right)(ze_S)=\left(zz^{\pi^{-1}}\pi\right)(ze_S).$$

Для второго множителя: $(ze_S)(e_E\pi)=(e_E\pi)(ze_S)^{(e_E\pi)}$.

Так как $e_S^{\pi}=e_S=e_S^{\pi^{-1}}$, , то верно, что выражения (12.А) и (12.Б) снова абстрактно совпадают. Поэтому опять по формуле (12.А) получим

$$(e_E\pi)(ze_S)^{(e_E\pi)}=(e_E\pi)\left(\left(ze_Ee_E^{e_S^{-1}}\right)^{\pi}e_S^{\pi}\right)=(e_E\pi)\left(z^{\pi}e_S\right).$$

Запишем правила коммутации следующим образом:

$$(ze_S)(e_E\pi)=\left(zz^{\pi^{-1}}\pi\right)(ze_S)=(e_E\pi)\left(z^{\pi}e_S\right)=(z\pi). \qquad (14)$$

Далее сформулируем правила коммутации для $(e_E\pi)(ze_S)$. Относительно сохранения первого множителя имеем $(ze_S)^{(e_E\pi)^{-1}}(e_E\pi)$. Далее по формуле (11) раскрываем отрицание $(ze_S)^{\left(e_E^{\pi^{-1}}\pi^{-1}\right)}(e_E\pi)=(ze_S)^{\left(e_E\pi^{-1}\right)}(e_E\pi)$. Подстановка сопряжения по выражению (12.А) или (12.Б) будет равна e_S, так как нейтральный элемент сопряжен сам с собой. Тогда для множителя нормального делителя независимо от типа А и типа Б имеем $\left(e_Ezee_E^{e_S^{-1}}\right)^{\pi^{-1}}=z^{\pi^{-1}}$ или $(e_E\pi)(ze_S)=\left(z^{\pi^{-1}}e_S\right)(e_E\pi)$. По формуле (14) также будет справедливо $(e_E\pi)(ze_S)=\left(z^{\pi^{-1}}\pi\right)$, и по выражению (11) можно перейти к $\left(z^{\pi^{-1}}\left(\pi^{-1}\right)^{-1}\right)=\left(z\pi^{-1}\right)^{-1}$. Теперь выполним коммутацию с сохранением второго множителя $(ze_S)(e_E\pi)^{(ze_S)^{-1}}=(ze_S)(e_E\pi)^{(ze_S)}$. Так как $\pi^{e_S}=\pi^{e_S^{-1}}=\pi$, то выражения (12.А) и (12.Б) абстрактно совпадают. Тогда для множителя нормального делителя имеем $\left(ze_Ez^{\pi^{-1}}\right)^{e_S}=zz^{\pi^{-1}}$, откуда $(ze_S)\left(zz^{\pi^{-1}}\pi\right)$. Запишем правила коммутации следующим образом:

$$\begin{aligned}(e_E\pi)(ze_S)&=\left(z^{\pi^{-1}}e_S\right)(e_E\pi)=(ze_S)\left(zz^{\pi^{-1}}\pi\right)=\\&=\left(z\pi^{-1}\right)^{-1}=\left(z^{\pi^{-1}}\pi\right).\end{aligned} \qquad (15)$$

Соотношения (14) и (15) являются частными случаями определения (13) и предоставляют аппарат для решения задач, связанных с действием элементов на вектор. Важно, что соотношения (14) и (15) являются абстрактными, т. е. одинаковы с точностью до типа А и типа Б.

В случае действия на вектор нескольких элементов E_n подряд допустимо применение внутригрупповой операции для E_n. Более того, такие действия будут коммутативны.

В случае действий нескольких элементов из S_n подряд необходимо пользоваться леммой 1.А и леммой 1.Б. Абстрактные соотношения для этого случая задать не удается.

Действие группы Джевонса на множестве булевых функций

Перед определением действия над булевыми функциями [4, 5] рассмотрим действие над функциями вообще. Определим конечные множества X и Y. Тогда под функцией f будем понимать некоторое отображение $f: X\to Y$ или $y=f(x)$, $x\in X, y\in Y$, где X — область определения; Y — область значений функции соответственно.

Действие элементов внешнего множества (т. е. функция не является элементом этого множества) на функцию можно определить многими способами. Такими способами могут быть замена переменных или подфункций в формуле исходной функции и др. Один из способов действия на функцию — преобразование аргумента. Результат этого — исходная функция, взятая в точках значений, на которые подействовал внешний элемент. Для такого преобразования верно, что внешний элемент определяет отображение $X\to X$. Определим такое действие в следующем виде:

$$f, g : X \to Y;$$
$$\mu : X \to X; \quad (16)$$
$$g = f^{\mu} : g(x) = f\left(x^{\mu}\right), \forall x \in X.$$

Лемма 3 (О композиции действий над БФ). Если над функцией $f: X \to Y$ проводятся подряд действия через аргумент $\mu, \nu : X \to X$, то результат их эквивалентен функции, аргумент которой преобразуется этими же действиями в обратном порядке, или $\left(f^{\mu}\right)^{\nu} = f\left(\left(x^{\nu}\right)^{\mu}\right)$.

Доказательство. Пусть $f, g, h : X \to Y$ и $g = f^{\mu}$, и $h = g^{\nu} = \left(f^{\mu}\right)^{\nu}$. Доказать лемму — значит определить h относительно f как (16). Тогда по определению (16) имеем $g(x) = f\left(x^{\mu}\right)$ и $h(x) = g\left(x^{\nu}\right)$. Определим для удобства $x' = x^{\nu}$, тогда $h(x) = g(x')$. Для $g(x) = f\left(x^{\mu}\right)$ получим $g(x') = f\left((x')^{\mu}\right) = f\left(\left(x^{\nu}\right)^{\mu}\right) = h(x)$. Что и требовалось доказать.

Под булевой функцией n аргументов $f(x_{n-1}, \ldots, x_i, \ldots, x_0)$ будем понимать классическое отображение $E^n \to E$. Тогда определим действие элемента группы Джевонса $(z\pi) \in D_n$ согласно (16) в следующем виде:

$$f^{(z\pi)} = f\left(x^{(z\pi)}\right). \quad (17)$$

Действие группы Джевонса над БВ выполняется в два этапа согласно формуле (13). Сначала действует нормальная часть, а затем — подстановка. Но для БФ можно операции свести к одному этапу (под этапом понимать (16)), так как существует возможность сначала вычислить аргумент, а затем выполнить действие над БФ. Тогда, опираясь на определение (16), лемму 3, выражения (13) и (15), непосредственно получим поэтапное абстрактное правило вычисления действия:

$$f^{(z\pi)} = f\left(x^{(z\pi)}\right) = f\left(\left(x^{z}\right)^{\pi}\right) = \left(f^{(e_E\pi)}\right)^{(ze_S)}; \quad (18.1)$$

и эквивалент (13) для действия на БФ:

$$\left(f^{(ze_S)}\right)^{(e_E\pi)} = f\left(\left(x^{(e_E\pi)}\right)^{(ze_S)}\right) = f\left(x^{(e_E\pi)(ze_S)}\right) =$$
$$= f\left(x^{\left(z\pi^{-1}\right)^{-1}}\right) = f^{\left(z\pi^{-1}\right)^{-1}} = f^{\left(z^{\pi^{-1}}\pi\right)}. \quad (18.2)$$

На данном этапе изложения совместно существуют три рода операций: групповые операции, действия групп на БВ и действия на БФ. Для разделения операций в выражениях применяют скобки. В частности, крайняя правая часть (18.1) и крайняя левая часть (18.2) есть последовательные действия на БФ согласно лемме 3.

Далее, опираясь на лемму 3 и выражение (17), сформулируем правило последовательного действия для произвольных элементов группы Джевонса на БФ:

$$\left(f^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = f^{(z_1\pi_1)(z_0\pi_0)} =$$
$$= f\left(\left(\left(\left(x^{z_1}\right)^{\pi_1}\right)^{z_0}\right)^{\pi_0}\right) = f\left(\left(\left(x^{z_1 z_0^{\pi_1^{-1}}}\right)^{\pi_1}\right)^{\pi_0}\right). \quad (19)$$

Соотношение (19) фактически показывает, что правая часть есть абстрактное описание произведения в группе Джевонса $(z_1\pi_1)(z_0\pi_0)$, согласно выражению (10.А) или (10.Б), в зависимости от типа А или типа Б.

Вычисление действия элемента группы Джевонса требует наличия формулы исходной БФ. Но можно вести такие вычисления без формулы (т. е. БФ задана таблицей значений) непосредственно по определению (16).

Действие группы β_n на множестве булевых функций

Помимо действия группы Джевонса над БФ можно определить действие группы β_n. Удается также установить связь между группой Джевонса и группой β_n по действию над БФ. Использование действий группы β_n не требует наличия формулы исходной БФ [4] и в ряде случаев позволяет снизить вычислительную сложность при построении действий группы Джевонса на БФ.

Элементы группы β_n — это подстановки симметрической группы S_k, и явно задать действие над БФ не получается. Тогда можно пойти следующим путем: связать с каждой БФ бинарный вектор и определить действие группы β_n над БФ как действие над эквивалентным ей бинарным вектором. Обозначим $y \in E^k$ и определим вектор:

$$y = \{f(11\ldots11), f(11\ldots10), \ldots, f(00\ldots01), f(00\ldots00)\},$$
$$y_j = f(j), 0 \leqslant j < k. \quad (20)$$

Например, для $f(x_3, x_2, x_1, x_0) = x_3(x_2 \oplus x_1 x_0) \vee x_2 x_1 \overline{x_0}$ будет $y = \{0111100001000000\}$.

Классически под булевой функцией принято понимать отображение $E^n \to E$, которое может быть представлено формулой. В различных базисах может существовать бесконечное множество формул, реализующих одинаковое отображение $E^n \to E$. Поэтому обычно вводится понятие класса эквивалентных БФ [4—6]. Соотношение (20) показывает отображение класса эквивалентных БФ на бинарный вектор. Фактически (20) — столбец таблицы истинности БФ. В контексте настоящего изложения

вопрос способа вычисления БФ (выбор конкретной формулы) не рассматривается. Поэтому понятия БФ и класса эквивалентных БФ в настоящем изложении принимаются как равные.

Действие симметрической группы над вектором и их правила сформулированы ранее, в частности (1.А) и (1.Б), в зависимости от типа преобразования. Изоморфизмы группы Джевонса и группы β_n также определены ранее как (6), (7), (8), (9.А) и (9.Б). Поэтому далее рассмотрим вопрос об эквивалентности групп Джевонса и β_n при действии на БФ. Для этого введем новое понятие — "эквиморфизм групп".

Определение. Две группы G, G', действующие на некотором множестве M, будем называть **эквиморфными**, если существует биекция $\varphi : G \to G'$, такая, что для любых $a, b \in G$ и $m \in M$:

$$\left(m^a\right)^b = \left(m^{\varphi(a)}\right)^{\varphi(b)}. \qquad (21)$$

При этом отображение φ будем называть эквиморфизмом.

Теорема 5 (Об эквиморфизме). Группа Джевонса D_n эквиморфна группе β_n по действию над БФ по типу Б и эквиморфна в обратные (отрицательные) элементы β_n по действию над БФ по типу А, и эквиморфизмами будут композиции (8), (9.А) и (9.Б).

Доказательство. Сначала определим явное отображение элемента группы Джевонса на элемент группы β_n. Пусть $f' = f^{(z\pi)}$, $(z\pi) \in D_n$ или $f'(x) = f\left(x^{(z\pi)}\right)$, $x \in E^n$. Тогда по выражению (20) имеем $y'_j = f\left(j^{(z\pi)}\right) = y_{\left(j^z\right)^\pi}$. Применим формулы (6) и (7): $y'_j = y_{\varphi_\pi(\varphi_z(j))} = y_{(\varphi_z\varphi_\pi)(j)}$. Из доказательства теоремы 4 примем определение $\varphi_{(z\pi)} = \varphi_z\varphi_\pi$ и получим $y'_j = y_{\varphi_{(z\pi)}(j)}$. Данное выражение есть действие подстановки на бинарный вектор согласно определению (1.Б), т. е. $f^{(z\pi)} = f^{\varphi_{(z\pi)}}$ для типа Б. Тип А и тип Б по действию на вектор связаны как обратные элементы из выражений (1.А) и (1.Б). Поэтому для типа А имеем $y'_{\varphi^{-1}_{(z\pi)}(j)} = y_j$ или $f^{(z\pi)} = f^{\varphi^{-1}_{(z\pi)}}$. Пусть $(z_0\pi_0)$, $(z_1\pi_1) \in D_n$, тогда для обоих типов получим $\left(f^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = f^{(z_1\pi_1)(z_0\pi_0)}$. По теореме 4 изоморфный образ произведения будет отличаться для разных типов.

Для типа А: $f^{(z_1\pi_1)(z_0\pi_0)} = f^{\left(\varphi_{(z_1\pi_1)}\varphi_{(z_0\pi_0)}\right)^{-1}}$. Вычисляя обратный элемент, получим $\left(f^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = f^{\varphi^{-1}_{(z_0\pi_0)}\varphi^{-1}_{(z_1\pi_1)}}$. Правая часть этого выражения может быть преобразована по формуле (1.А), т. е. $\left(f^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = \left(f^{\varphi^{-1}_{(z_0\pi_0)}}\right)^{\varphi^{-1}_{(z_1\pi_1)}}$.

Для типа Б: $f^{(z_1\pi_1)(z_0\pi_0)} = f^{\varphi_{(z_1\pi_1)}\varphi_{(z_0\pi_0)}}$. Правая часть полученного выражения может быть преобразована по формуле (1.Б), т. е. $\left(f^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = \left(f^{\varphi_{(z_0\pi_0)}}\right)^{\varphi_{(z_1\pi_1)}}$. Что и требовалось доказать.

Эквиморфизмом типа А будет $(z\pi) \to \varphi^{-1}_{(z\pi)} = \varphi^{-1}_\pi\varphi^{-1}_z$, последние отображения должны вычисляться по формулам (8) и (9.А). Эквиморфизмом типа Б будет $(z\pi) \to \varphi_{(z\pi)} = \varphi_z\varphi_\pi$, последние отображения должны вычисляться по формулам (8) и (9.Б).

Теорему 5 можно формулировать как эквиморфное вложение для обоих типов. Это не породит ошибки. Но тогда для типа А эквиморфизм не будет произведением отображений (8) и (9.А), и появятся трудности при выводе более сложных соотношений.

Эквиморфизм группы Джевонса и группы β_n позволяет упростить вычисления при построении действий над БФ. Можно утверждать, что над всем множеством булевых функций задано два набора действий, которые эквиморфны друг другу. БФ ассоциированы с БВ длины k, следовательно, действия заданы и над ними. Но для задач обработки информации важно обратное этому утверждение: **для любого бинарного вектора длины k среди всех возможных ее преобразований посредством S_k можно выделить группу действий D_n, которая вкладывается изоморфно и эквиморфно в S_k и задает эквивалентность на множестве БВ длины k**. Группа действий D_n и задаваемая ею эквивалентность есть следствие природы бинарных векторов, а не наоборот. Другими словами, вышеописанный математический аппарат позволяет задать действия группы Джевонса степени n над бинарными векторами длины k.

Заключение

В работе [1] показана необходимость автоматизированной обработки элементов группы Джевонса. Настоящая статья преимущественно оперирует объектами экспоненциального размера. Это связано с тем, что сами по себе действия элементов группы Джевонса на множестве булевых векторов малоинтересны для обработки информации, но необходимы для описания действий над функциями, которые и являются биекцией информационных сообщений. Приведенные формулы также реализованы в библиотеке "domain operations processor" (или **dop**). Библиотека распространяется под лицензией GNU LGPL (GNU Lesser General Public License).

Макет библиотеки **dop** содержит основной макрос, реализующий выражение (17), и отмечен как "прямое

действие над БФ". Он является эталонным преобразованием, относительно которого можно проверять более сложные вычисления.

Отдельно стоит прокомментировать применение эквиморфизмов для повышения производительности вычислений. Прямое действие над БФ фактически сводится к преобразованию аргумента и процедурам извлечения элемента из бинарного вектора (таблицы истинности) с последующим его погружением в новый результат. На практике это приводит к следующему. Пусть $n = 23$, тогда бинарный вектор будет иметь размер 1 Мбайт ($2^{3+10+10} = 8\,388\,608$ бит). Прямое действие над таким объектом на среднестатистической (1 ядро, тактовая частота процессора 2 ГГц) ПЭВМ занимает несколько секунд. Для реальной работы (не эксперимент) это время огромно.

Эквиморфизмы, на первый взгляд, решают задачу временной оптимизации вычисления действия элементов группы Джевонса на БФ только частично. Фактически эквиморфный образ — это вычисленные значения аргументов БФ при прямом действии. Более того, можно показать недостаток. Эквиморфный образ нужно рассчитывать и хранить. Для примера, эквиморфный образ действия на БФ, эквивалентную 1 Мбайту (23 аргумента), занимает 32 Мбайта (из расчета 4 байта на значение точки подстановки).

Вместе с тем выражения (8), (9.А) и (9.Б) удобнореализуемы для транспозиций и БВ, содержащих только одну единицу. Нет необходимости их хранить и есть возможность одновременной массивной обработки. Покажем на примере БВ {00...001}. Действие приведет к перестановке четных и нечетных битов в каждой паре битов вектора (таблицы истинности). На языке C/C++ это будет выглядеть как $x = ((x \,\&\, 0x55555555) << 1) \mid ((x \,\&\, 0xaaaaaaaa) >> 1)$ для каждого двойного слова вектора, что приведет к обработке сразу 32 значений. Современные процессорные расширения позволяют обрабатывать 128 и 256 бит единовременно (обработка проводится элементарными арифметическими и логическими операциями). Начиная с определенных значений транспозиций и положения единицы в БВ, не требуется обработка битов. В силу регулярной структуры подстановок эквиморфных образов работа может проводиться в байтах, словах и т.д.

Для такого использования нужно раскладывать специфичным образом БВ и подстановки. Сложность таких разложений линейна. На малых значениях n прирост производительности невелик. Но при увеличении n в силу экспоненциальной связи между размером таблицы истинности БФ и количеством ее аргументов прирост производительности будет в сотни, а при корректной реализации — и в тысячи раз. В этом и состоит ключевое и основополагающее значение теоремы 5. Именно это и позволяет вести исследования информации практически. Такое расширение **dop** названо "эквиморфный вычислитель".

Материалы работы [1] и этой статьи, а также библиотека **dop** формируют вычислительное законченное ядро для обработки информации в виде БФ. На сегодняшний день ведется разработка эквиморфного вычислителя. Состав библиотеки, включая документацию разработчика (Software Development Kit или SDK), можно получить у авторов настоящей статьи.

Список литературы

1. **Кукарцев А. М., Кузнецов А. А.** О конструктивном представлении группы Джевонса для построения инженерно-технических решений обработки информации // Программная инженерия. 2015. № 11. С. 25—33.
2. **Шеннон К.** Работы по теории информации и кибернетике. Пер. с англ. / под ред. Р. Л. Добрушина и О. Б. Лупанова с пред. А. Н. Колмогорова. М.: Издательство иностранной литературы, 1963. 832 с.
3. **Сэломон Д.** Сжатие данных, изображений и звука. Пер. с англ. М.: Техносфера, 2004. 368 с.
4. **Логачёв О. А., Сальников А. А., Ященко В. В.** Булевы функции в теории кодирования и криптологии. М.: МЦМНО, 2004. 470 с.
5. **Марченко С. С.** Замкнутые классы булевых функций. М.: ФИЗМАТЛИТ, 2000. 128 с.
6. **Глухов М. М., Ремизов А. Б., Шапошников В. А.** Обзор по теории *k*-значных функций. Часть 1. Справ. пос. / Ред. Н. Р. Емельянов. Заказ № 163ф. М.: Типография в/ч 33965, 1988. 153 с.
7. **Каргаполов М. И., Мерзляков Ю. И.** Основы теории групп. 3-е изд., перераб. и доп. М.: Наука, 1982. 288 с.
8. **Яубайтис Э. А.** Субклассы и классы булевых функций // Автоматика и вычислительная техника. 1974. № 1. С. 1—8.
9. **Golomb S. W.** On classification of Boolean functions // IRE, Trans. circuit theory. 1959. N 6, Spec. Suppl. P. 176—186.
10. **Бентли Дж**. Жемчужины программирования. 2-е изд. Пер. с англ. / под ред. А. Пасечник. СПб.: Питер, 2002. 272 с.
11. **Супруненко Д. А**. Группы подстановок. Минск: Навука і тэхніка, 1996. 366 с.
12. **Страуструп Б.** Язык программирования C++. Специальное издание. Пер. с англ. / под ред. Н. Н. Мартынова. М.: Бином, 2011. 1136 с.

About Actions of the Jevons Group on Sets of Binary Vectors and Boolean Functions for Engineering Solutions of Information Processing

A. M. Kukartsev, amkukarcev@yandex.ru, **A. A. Kuznetsov,** kuznetsov@sibsau.ru, Siberian State Aerospace University named after academician M. F. Reshetnev, Krasnoyarsk, 660014, Russian Federation

Corresponding author:

Kukartsev Anatolii M., Senior Lecturer, Siberian State Aerospace University named after academician M. F. Reshetnev, Krasnoyarsk, 660014, Russian Federation, e-mail: amkukarcev@yandex.ru

Received on September 11, 2015
Accepted on September 25, 2015

Actions of the Jevons group on the set of binary vectors of the Boolean functions allow to build engineering solutions for the submission and processing of information. The materials are a continuation of the article "Constructive representation of the Jevons group for engineering solutions of information processing". In this case information will be presented as a binary vector of length 2^n, associated with the Boolean function. The binary vector is a column of values of the Boolean function. This part provides rules for computing and commutating on the set of Boolean vectors and Boolean functions. The notion of equivalent actions (equimorphisms) on the set of Boolean functions is presented. Equimorphisms are allowed to improve significantly the performance of software and hardware implementations. As a result (along with the previous article) we offer a complete mathematical apparatus for converting information by means of elements of the Jevons group. It concludes with recommendations on the use of equimorphisms in real problems' descriptions and solutions. We have developed the library of software functions "domain operations processor" (or dop) based on the rules. The library is licensed under the GNU Lesser General Public License.

Keywords: *information representation, the Jevons group, action on the set, binary vectors, Boolean functions, computational tools*

Acknowledgements: *This work was supported by the Grant of Russian Federation President, project no. MD-3952.2015.9*

For citation:

Kukartsev A. M., Kuznetsov A. A. About Actions of the Jevons Group on Sets of Binary Vectors and Boolean Functions for Engineering Solutions of Information Processing, *Programmnaya Ingeneria,* 2016, vol. 7, no 1, pp. 29—36.

DOI: 10.17587/prin.7.29-36

References

1. **Kukartsev A. M., Kuznetsov A. A.** O konstruktivnom predstavlenii gruppy Dzhevonsa dlja inzhenerno-tehnicheskih reshenij obrabotki informacii (Constructive representation of the Jevons group for engineering solutions of information processing), *Programmnaya Ingeneria*, 2015, no. 11, pp. 25—33 (in Russian).

2. **Shennon K.** *Raboty po teorii informacii i kibernetike* (Works on information theory and cybernetics). Per. s angl. / Ed. R. L. Dobrushin, O. B. Lupanov, Moscow, Izdatel'stvo inostrannoj literatury, 1963, 832 p. (in Russian).

3. **Salomon D.** *A guide to data compression methods,* New York, Springer-Verlag, 2006, 1092 p.

4. **Logachjov O. A., Sal'nikov A. A., Jashhenko V. V.** *Bulevy funkcii v teorii kodirovanija i kriptologii* (Boolean functions in coding theory and cryptology), Moscow, MCMNO, 2004, 470 p. (in Russian).

5. **Marchenko S. S.** *Zamknutye klassy bulevyh funkcij* (Post's lattice), Moscow, FIZMATLIT, 2000, 128 p. (in Russian).

6. **Gluhov M. M., Remizov A. B., Shaposhnikov V. A.** *Obzor po teorii k-znachnyh funkcij. Chast' 1. Spravochnoe posobie* (Review on the Theory of k-valued functions. Part 1. A Reference Guide) / Ed. N. R. Emel'janov. Zakaz no. 163f, Moscow, Tipografija v/ch 33965, 1988, 153 p. (in Russian).

7. **Kargapolov M. I., Merzljakov Ju.I.** *Osnovy teorii grupp* (Basics of group theory). 3rd ed., Moscow, Nauka, 1982, 288 p**.** (in Russian).

8. **Jaubajtis Je. A.** Subklassy i klassy bulevyh funkcij **(**Sub class and the class of Boolean functions), *Avtomatika i Vychislitel'naja Tehnika*, 1974, no. 1, pp. 1—8. (in Russian).

9. **Golomb S. W.** On classification of Boolean functions, *IRE, Trans. circuit theory,* 1959, no. 6, *Spec. Suppl.*, pp. 176—186.

10. **Bentley J.** *Programming pearls.* 2nd edition, New Jersey, Addison-Wesley, 1999. 256 p.

11. **Suprunenko D. A**. *Gruppy podstanovok* (Groups of permutations), Minsk, Navuka i tjehnika, 1996. 366 p. (in Russian).

12. **Stroustrup B.** *The C++ Programming Language.* Special edition, New Jersey: Addison-Wesley, 2000. 1040 p.

УДК 004.021 DOI: 10.17587/prin.7.37-45

О. В. Ефимова, инженер-исследователь, e-mail: office-nsk@wellink.ru,
С. А. Пирогов, генеральный директор, ООО Веллинк Технологии, г. Новосибирск,
С. А. Семенов, вед. системный инженер, ООО Веллинк, г. Москва

Алгоритм сравнения интернет-провайдеров, основанный на семействе методов ELECTRE

Представлен разработанный авторами многокритериальный способ сравнения интернет-провайдеров, базирующийся на семействе методов ELECTRE. Эти методы были разработаны для принятия решений при необходимости выбора среди нескольких альтернатив, при нескольких критериях выбора. Для сравнения провайдеров используются следующие критерии: скорость передачи данных, время суток проведения измерения скорости, разнообразие тарифных планов провайдеров. Данная методика разработана и реализована в компании ООО Веллинк и может использоваться пользователями сети Интернет.

Ключевые слова: *метод ELECTRE, рейтинг провайдеров, многокритериальные методы принятия решений, схема Simos, измерения скорости связи Интернет*

Введение

Для многих людей доступ к компьютеру и высокоскоростной сети Интернет с каждым днем становится все более и более важен. Компьютер и сеть Интернет используют для выполнения домашних заданий в школе, поиска работы, планирования отдыха и покупок билетов, просмотра фильмов, для связи с родными людьми и друзьями, что неоспоримо показывает высокую значимость цифровых технологий в жизни общества. С усовершенствованием программного обеспечения и оборудования, которые используются для передачи данных, растет также и число провайдеров (поставщиков услуг) связи. Развивается доступ к сети Интернет с помощью технологий радиодоступа, спутниковой и мобильной связи. В результате возникают вопросы как о качестве, так и о возможности сравнения и выбора наиболее оптимального из провайдеров, согласно тем или иным требованиям, которые к ним предъявляют. Как в повседневной жизни, так и в профессиональной деятельности люди сталкиваются с необходимостью принятия такого решения, имея при этом несколько конфликтующих друг с другом критериев. Например, одними из основных критериев при выборе провайдера и подключении к сети Интернет являются цена и скорость передачи информации. Эти два критерия могут рассматриваться как конфликтующие. В подобных ситуациях обычно выбор осуществляется интуитивно. Однако при большом числе критериев очень важно однозначно, с использованием математически точных формул, рассчитать их влияние. Данная работа посвящена подобному расчету.

Одним из широко используемых методов, помогающих в принятии решений, является метод анализа иерархий (МАИ), созданный американским математиком Томасом Саати [1]. Этот метод позволяет найти альтернативу, которая наилучшим образом согласуется с предъявляемыми требованиями. Следующие шаги кратко описывают алгоритм поиска решения:

- построение иерархической структуры, содержащей цель, критерии, альтернативы и другие факторы, влияющие на выбор;
- определение приоритетов всех элементов иерархии с использованием метода парных сравнений;
- построение глобальных приоритетов альтернатив с использованием линейной свертки приоритетов элементов;
- принятие решения на основе полученных результатов.

Этот метод широко используется как на межгосударственном уровне, так и при решении частных задач в бизнесе, промышленности, здравоохранении и образовании.

В основе подхода, который рассматривается в настоящей работе, лежит метод многокритериальной оценки ELECTRE (ELimination Et Choix Traduisant la Eealite). Его суть — исключение и выбор, отражающие реальность. Этот метод был разработан в Европе в 1960-х гг. группой французских ученых во главе с профессором Б. Руа [2, 3]. Позднее было развито целое семейство методов принятия решений, таких как выбор, сравнение и сортировка, базирующихся на методе ELECTRE. Метод работает для сравнения объектов по 3...12 критериям. Хотя бы по одному критерию должна собираться статистика. Для упо-

рядочивания многокритериальных альтернатив они сравниваются с использованием специальных индексов согласия (конкорданса) и несогласия (дискорданса). Речь идет о согласии или несогласии с гипотезой, что одна альтернатива превосходит другую.

Семейство методов PROMETHEE также является семейством методов принятия решений при наличии нескольких критериев. Одним из существенных отличий этих методов, по сравнению с методами ELECTRE, является меньший набор дополнительных данных (помимо матрицы критерий — альтернатива и весов критериев). Сильной стороной PROMETHEE является наличие мощных графических механизмов для представления ранжирования альтернатив [4].

В настоящей работе объектом исследования являются провайдеры сети Интернет. К основным характеристикам провайдеров относятся такие факторы, как скорость передачи данных, стабильность связи, стоимость, зона покрытия, разнообразие тарифных планов, качество предоставления помощи пользователям и т. д. К сожалению, не все из перечисленных характеристик являются доступными и поэтому не могут использоваться для расчета оценки провайдера.

Целью данной работы является разработка метода сравнения интернет-провайдеров с использованием варьируемого набора критериев. К числу таких критериев относятся: скорость передачи данных; время суток проведения измерения скорости; разнообразие тарифных планов провайдеров.

Основы формальной модели

Методы ELECTRE направлены на решение задач с уже заданными многокритериальными альтернативами. Оценка альтернатив является относительной, т. е. устанавливается условие превосходства одной альтернативы над другой. Попарное сравнение альтернатив с учетом численного значения веса каждого из критериев позволяет построить матрицы согласия/несогласия. Элементы этих матриц используют для расчета численного значения рейтингов каждой из альтернатив, т. е. каждого из провайдеров. В итоге расчет рейтингов провайдеров, более подробно описанный далее, проводится в три этапа: определение весов критериев; расчет матриц согласия/несогласия; расчет самих рейтингов.

Используемые данные. Для получения рабочего набора данных проводится многократное измерение скорости передачи информации для каждого из провайдеров из точки А в точку Б (далее, по маршруту АБ) в разное время суток с разных устройств доступа в сеть Интернет. Таким образом, получаем набор пар значений, а именно — скорость передачи данных и время измерения скорости.

Определение критериев. Как было упомянуто выше, для сравнения провайдеров в данной работе были выбраны три критерия: скорость передачи данных (*cs*, критерий скорости); разнообразие тарифных планов провайдеров (*cp*, критерий тарифных планов); время суток проведения измерения скорости (*ct*, критерий времени). Чем выше критерий скорости, тем более высокую оценку получает провайдер. Аналогично, чем выше вариация представленных провайдером скоростей, тем разнообразнее тарифные планы и, следовательно, более высокий рейтинг провайдера. В случае критерия времени рассматривают три периода: ночь (00.00—08.00); день (08.00—20.00); вечер (20.00—24.00). Более высокая оценка дается провайдеру, у которого преобладают вечерние тарифы с высокими скоростями передачи информации. Представляемый в данной работе алгоритм легко расширить путем добавления новых критериев, например, критерия типа связи (*cc*), либо критерия цены услуги предоставления связи (в текущей версии алгоритма отсутствует). В качестве типа связи могут выступать любые типы, например, мобильная или проводная, но этот выбор не влияет на результаты, предоставляемые в данной работе. В текущей реализации может использоваться один из таких типов. Ниже представлен пример расчета рейтингов провайдеров, где этот критерий был добавлен, но без реализации, так как все входящие данные для тестирования алгоритма имеют мобильный тип связи. Это сделано для того чтобы продемонстрировать гибкость использования алгоритма.

Одним из основных предположений, используемых при расчетах в данной работе, является тот факт, что измерения скорости по маршрутам предполагаются одинаково распределенными, даже если измерения были проведены с разных устройств доступа в сеть Интернет. Выборки данных также предполагаются однородными. Значение критерия скорости рассчитывается как выборочное среднее всех значений скоростей, полученных по одному маршруту, согласно формуле $cs = \frac{1}{N}\sum_{i=1}^{N} x_i$, где N — общее число измерений; x_i — значение скорости, полученное в измерении i.

Значение критерия тарифных планов рассчитывается как выборочная дисперсия всех измерений скорости с устройств с уникальным идентификатором. Такой подход был выбран для того, чтобы не учитывать повторно тарифный план с устройства, с которого было получено несколько измерений скорости. В случае нескольких измерений для расчета берется максимальное значение. Формула для расчета: $cp^2 = \frac{1}{K}\sum_{i=1}^{K}(x_i - \bar{x})^2$, где K — число измерений, полученных с устройств с уникальным идентификатором; $\bar{x}$ — выборочное среднее значение скорости; i — номер измерения.

Для расчета значения критерия времени вводится вектор [0,5 1,0 1,5], элементы которого соответствуют трем упомянутым выше периодам времени (ночь, день и вечер). Так как в вечернее нерабочее время число пользователей сети Интернет резко возрастает, этому периоду приписывается максимальн-

ный коэффициент, т. е. 1,5. В случае ночного периода этот коэффициент минимальный, т. е. 0,5. Сам параметр критерия рассчитывается по формуле $ct = \frac{1}{N}\sum_{i=1}^{N} t_i$, где t_i — одно из трех значений вектора [0,5 1,0 1,5] в зависимости от того, в какой период времени было проведено измерение; i — номер измерения; N — общее число измерений.

Шкалы критериев. Максимальные значения критериев выбраны следующим образом: 100 000 кбит/с для критерия скорости; 2 500 000 000 (что является максимальным значением дисперсии при максимальном значении скорости 100 000 кбит/с и минимальном значении 0 кбит/с) для критерия тарифных планов; 1,5 для критерия времени (максимальное значение элемента вектора критерия времени). Важно заметить, что эти значения могут варьироваться и не влияют на построение алгоритма и выводы, сделанные в данной работе.

Определение весов критериев. Веса критериев могут определяться несколькими способами. Первый способ основан на мнениях пользователей, которые назначают каждому критерию числовое значение в пределах заданного интервала. Чем ближе значение к максимально возможному, тем больший вклад будет иметь этот критерий при расчете матриц согласия/несогласия и самих рейтингов.

Второй способ, также базирующийся на личном мнении пользователей, заключается в постановке в соответствие каждому критерию игровой карты. Этот метод был предложен во Франции автором J. Simos в 1990 г. [5, 6] и позже усовершенствован J. Figueira и B. Roy [7]. При наличии M критериев пользователю дается M игральных карт, на каждой из которых отображено название критерия. Пользователю также дается набор белых карт, не имеющих каких-либо надписей. Число белых карт выбирается самим пользователем согласно алгоритму, описанному ниже. Далее пользователю предлагается отсортировать эти карты в порядке возрастания важности критерия согласно его личному выбору: первая карта соответствует наименее значимому критерию, последняя карта соответствует наиболее значимому критерию. Если нескольким критериям приписывается одинаковая значимость, то эти карты соединяются вместе в один набор. При этом оговаривается, что разница важности каждого из двух следующих друг за другом критериев может варьироваться. Для этого пользователю предлагается вставить некоторое число белых карт между всеми сортированными картами. Чем больше разница в важности критериев двух соседних карт (или наборов карт, в случае одинаковой важности), тем больше число добавляемых белых карт. Отсутствие белых карт между двумя соседними критериями означает, что разница важности между ними принимается минимальной, обозначим ее h. Если между двумя соседними критериями была добавлена одна белая карта, то разница важности этих двух критериев принимается вдвое большей и равной $2h$. Аналогичным образом поступают с большим числом белых карт. Подробный пример расчета весов критериев с использованием набора игральных карт описан в работе [7]. Так как процесс расчета является очень громоздким и используется строго по рекомендациям автора, в данной работе он не приводится.

Следует также принять во внимание, что число белых карт можно рассчитать, используя имеющуюся выборку измерений скорости. Для каждого из провайдеров скорость измеряется в разное время суток, и, как следствие, выборка является неоднородной. В данной работе для оценки провайдеров мы вынуждены пренебречь этой неоднородностью и считать выборку измерений распределенной нормально при достаточно большом количестве измерений. Число n белых карт для критерия скорости определяется по следующей формуле: $n = 3\left(1 - \frac{am}{N}\right)$, где am — амплитуда, соответствующая наиболее часто встречающейся скорости в гистограмме нормального распределения, нормированная на общее число измерений; N — общее число измерений. Коэффициент 3 соответствует максимально возможному числу белых карт и устанавливается независимо от измерений. Формула была выведена согласно следующим соображениям: чем больше разброс по скоростям и, соответственно, ниже максимальная амплитуда гистограммы, тем большую значимость получает критерий скорости. В этом случае число белых карт стремится к максимальному. Наоборот, чем меньше разброс по скоростям, тем меньшее значение приобретает этот критерий с числом белых карт, стремящимся к минимальному значению. Приведенную выше формулу можно также записать в следующем виде: $n = 3\left(1 - f\left(\bar{x}\right)\right)$, где n — число белых карт; $f\left(\bar{x}\right)$ — значение функции плотности вероятности в точке выборочного среднего. Нормальное распределение задается функцией Гаусса: $f\left(x\right) = \frac{1}{\left(\sqrt{s^2 2\pi}\right)} e^{\frac{-(x-\bar{x})}{2s^2}}$, где s^2 — выборочная дисперсия; $\bar{x}$ — выборочное среднее значений измерений x, представленных выборкой $x = \left(x_1, x_2, ..., x_N\right)$. Принимая $x = \bar{x}$, получаем следующее выражение для функции плотности: $f\left(\bar{x}\right) = \frac{1}{\sqrt{s^2 2\pi}}$.

Аналогично рассчитывается число белых карт для критерия тарифных планов. Отличием является то обстоятельство, что выборочная дисперсия рассчитывается по измерениям скорости с устройств с уникальным идентификатором. Если с одного устройства получено несколько измерений, то берется максимальное значение. Это сделано для того чтобы не учитывать повторно тарифный план одного и того же устройства (пользователя).

Для оценки критерия времени были выбраны описанные выше три интервала: ночь, день, вечер. Число измерений — N_{t1}, N_{t2}, N_{t3}, соответственно. Число белых карт рассчитывается по формуле, схожей по

смыслу с формулами для предыдущих критериев: $n = 3\left(1 - \frac{\max\{N_{t1}, N_{t2}, N_{t3}\}}{N}\right)$. Числитель дроби равен максимальному значению среди значений в скобках: N_{t1}, N_{t2}, N_{t3}; N — общее число измерений.

Закончив расчет числа белых карт, необходимо вернуться к схеме Simos для расчета веса каждого из критериев [7].

Расчет матриц согласия/несогласия. Для любой пары альтернатив (провайдеров) A_u и A_v существуют векторы оценок для каждого из имеющихся M критериев: $z_u = (z_{u1}, ..., z_{uM})$. В нашем случае A_u и A_v — это два сравниваемых провайдера, а z_u — это набор значений критериев определенного провайдера связи. Ранее было показано, как рассчитывают значения каждого из критериев k. Коэффициент согласия $\alpha_{u,v}$ для сравнения операторов A_u и A_v вычисляется по формуле $\alpha_{u,v} = \frac{1}{\sum_{i=1}^{M} k_i} \sum_{j \in X(u,v)} k_j$, где i — номер критерия, M — число критериев. Сумма всех весов критериев равна 100, $\sum_{i=1}^{M} k_i = 100,$, где i — номер критерия. В числителе суммирование проводится только по тем значениям весов критериев, для которых значения критериев провайдера A_u лучше, либо равны значениям критериев провайдера A_v. Таким образом, $X(u,v) = \{j \mid z_{uj} \geq z_{vj}\}$ — это набор критериев, для которых A_u лучше, чем A_v. Для этих критериев и проводится суммирование весов.

Коэффициент несогласия $\beta_{u,v}$ для сравнения операторов A_u и A_v вычисляют по формуле $\beta_{u,v} = \begin{cases} 0, & if \mid Y(u,v) \mid = 0 \\ \text{else} & \max\limits_{j \in Y(u,v)} \frac{k_i \mid z_{uj} - z_{vj} \mid}{s_j \sum_j k_j} \end{cases}$, где s_j — размер шкалы соответствующего критерия. Сумма всех весов критериев равна 100. Максимальное значение выбирается среди тех значений критериев, где A_u хуже, чем A_v. Таким образом, $Y(u,v) = \{j \mid z_{uj} < z_{vj}\}$ — это набор критериев, для которых A_u хуже, чем A_v. Если таких критериев нет, т. е. $\mid Y(u,v) \mid = 0,$ то значение коэффициента несогласия принимается равным нулю.

В случае двух сравниваемых провайдеров результирующие матрицы согласия и несогласия выглядят следующим образом:

$$\begin{pmatrix} - & \alpha_{u,v} & \alpha_{u,abs} \\ \alpha_{v,u} & - & \alpha_{v,abs} \\ \alpha_{abs,u} & \alpha_{abs,v} & - \end{pmatrix} \text{ и } \begin{pmatrix} - & \beta_{u,v} & \beta_{u,abs} \\ \beta_{v,u} & - & \beta_{v,abs} \\ \beta_{abs,u} & \beta_{abs,v} & - \end{pmatrix}.$$

В матрицах также присутствуют коэффициенты согласия и несогласия для сравнения провайдеров A_u и A_v с абсолютным провайдером A_{abs}, задающим максимально возможные значения для каждого из критериев. Параметры для абсолютного провайдера выбирают равными значениям шкал критериев, данных выше.

Расчет рейтингов провайдеров. Для получения числового значения рейтинга каждого из провайдеров используют следующую формулу:

$$r(j) = \frac{1}{m-1}\left(\sum_{i=1, i \neq j}^{m} \alpha_{j,i} - \sum_{i=1, i \neq j}^{m} \beta_{j,i}\right),$$

где j — номер провайдера; $(m - 1)$ — это число провайдеров без учета абсолютного провайдера. Формула показывает нормированную разницу между суммами коэффициентов согласия и несогласия для каждого из операторов. Для того чтобы рейтинги не были отрицательными, в случае, когда коэффициент согласия меньше коэффициента несогласия, шкала рейтингов сдвигается и нормируется. Итоговая формула имеет следующий вид: $R(j) = \frac{r(j) + 1}{2} 100.$

Обсуждение

Во всех рассуждениях, представленных ранее, речь шла об измерениях, полученных по одному маршруту, а именно, если передача данных осуществляется из города А в город Б с различных устройств, имеющих доступ в сеть Интернет. В случае если проводят измерения по различным маршрутам и необходимо посчитать суммарный рейтинг, каждый из рейтингов входит в суммарный с весом, пропорциональным числу абонентов в городе, из которого проводится измерение. В результате представленного подхода к расчетам рейтинги маршрутов с небольшим числом пользователей войдут в суммарный рейтинг с меньшим весом, повышая приоритет маршрутов с большим числом пользователей.

Наиболее долгим по времени шагом является расчет значений критериев, т. е. выборочных средних и выборочных дисперсий. Со временем число измерений может вырасти до сотен миллионов, поэтому важно, чтобы в алгоритм можно было добавлять новые выборки данных без пересчета предыдущих. Для того чтобы предусмотреть возможности добавления новых данных, рассмотрим шаги, схематично описывающие работу всего алгоритма.

Шаг 1. Получение набора данных.

Шаг 2. Определение весов критериев.

Шаг 3. Расчет значений каждого из критериев (включая расчет сумм элементов выборки, сумм квадратов элементов и дисперсий всех критериев для каждого из провайдеров).

Шаг 4. Расчет матриц согласия и несогласия.

Шаг 5. Расчет рейтингов по каждому из маршрутов и рейтинга в целом.

Даже для нескольких сотен провайдеров шаги 4 и 5 занимают время на несколько порядков меньше, чем шаг 3, включающий расчет сумм элементов. По этой причине очень важно не пересчитывать данные, полученные на этом шаге. При добавлении следующей выборки данных, например, за следующий период времени (неделя, месяц), необходимо снова вернуться на шаг 3 и рассчитать суммы элементов и суммы квадратов элементов новой выборки. Очень важно также сохранять значения сумм, полученные для предыдущей выборки, так как они пригодятся для расчета значений критериев объединенной выборки. Итак, при объединении двух выборок новые значения выборочного среднего и выборочной дисперсии рассчитывают по следующим формулам: $\bar{x} = \frac{\sum_{i=1}^{P_1} x_{1i} + \sum_{i=1}^{P_2} x_{2i}}{P_1 + P_2}$ и $s^2 = \frac{\sum_{i=1}^{P_1} x_{1i}^2 + \sum_{i=1}^{P_2} x_{2i}^2}{P_1 + P_2} - \bar{x}^2$, x_{1i}, x_{2i} — значения измеряемой величины в первом и во втором наборах измерений; i — номер измерения. Объемы выборок предполагаются разными, P_1 и P_2 для текущей и новой выборок, соответственно. Как видно из формул, при известных сумме и сумме квадратов расчет значения критериев для объединенной выборки не занимает много времени. Выборочная дисперсия является смещенной оценкой теоретической дисперсии, но разницей между смещенной и несмещенной оценками можно пренебречь при больших размерах выборки. Далее, как и прежде следует расчет новых матриц согласия/несогласия и новых рейтингов, занимающий относительно небольшое время. Этот цикл повторяется каждый раз при добавлении новой выборки результатов измерений без необходимости пересчета результатов, полученных на предыдущем этапе.

Пример расчета

Данные для тестирования алгоритма. В табл. 1 представлены гипотетические данные двух провайдеров, которые использовались для ручного тестирования алгоритма. Номера измерений представлены

Таблица 1

Данные для тестирования алгоритма

Провайдер 1					Провайдер 2				
Номер измерения	Id устройства	Скорость передачи информации, кбит/с	Время	Города (расстояние, км)	Номер измерения	Id устройства	Скорость передачи информации, кбит/с	Время	Города (расстояние, км)
Первый набор данных									
1.1	1	25 000	07.00	Москва—Казань (800)	1.2	2	50 000	15.00	Ульяновск—Томск (2800)
2.1	3	30 000	06.00	Новосибирск—Санкт-Петербург (3800)	2.2	2	20 000	22.00	Ульяновск—Самара (300)
3.1	4	15 000	10.00	Москва—Ярославль (270)	3.2	5	20 000	15.00	Москва—Санкт-Петербург (700)
4.1	6	15 000	02.00	Ульяновск—Ярославль (840)	4.2	7	25 000	16.00	Москва—Ульяновск (900)
5.1	8	20 000	14.00	Ульяновск—Санкт-Петербург (1600)	5.2	9	10 000	21.00	Ульяновск—Санкт-Петербург (1600)
6.1	10	24 000	02.00	Ульяновск—Санкт-Петербург (1600)	6.2	11	14 000	22.00	Ульяновск—Санкт-Петербург (1600)
7.1	12	20 000	17.00	Ульяновск—Пермь (800)	7.2	13	16 000	21.00	Ульяновск—Пермь (800)
Второй набор данных									
8.1	14	16 000	19.00	Ульяновск—Санкт-Петербург (1600)	8.2	15	18 000	07.00	Ульяновск—Санкт-Петербург (1600)
9.1	16	20 000	02.00	Ульяновск—Омск (1940)	9.2	17	14 000	22.00	Ульяновск—Омск (1940)
10.1	18	22 000	10.00	Ульяновск—Казань (740)	10.2	19	15 000	22.00	Москва—Пенза (640)

в виде *i*, *j*, где *i* — номер измерения, а *j* — номер провайдера. Для каждого из измерений в табл. 1 представлены следующие данные:

- идентификационный номер устройства;
- скорость передачи информации, кбит/с;
- время проведения измерения, представленное в 24-часовом формате;
- два города, между которыми происходит передача информации, в скобках указано расстояние между ними, км.

Данные подобраны таким образом, чтобы отобразить все особенности алгоритма: наличие измерений с общими маршрутами для обоих провайдеров (например, измерения 5.1 и 5.2); различные расстояния между городами; добавление нового набора данных и возможность добавления дополнительных критериев (критерий типа связи).

Определение весов критериев. На данном этапе предполагается, что пользователь задал обозначенные в табл. 2 белые карты для каждого из критериев, согласно личному выбору.

Как было отмечено ранее, для расчета весов критериев используют формулы, приведенные в работе [5]. Число белых карт между рангами r и $r + 1$ обозначим как e'_r, $\overline{n}$ — число рангов, также введем следующие обозначения:

$$e_r = e'_r + 1, \forall r = 1, \ ..., \ \overline{n} - 1;$$

$$e = \sum_{r=1}^{\overline{n}-1} e_r;$$

$$u = \frac{z - 1}{e}.$$

Величина z показывает, во сколько раз последний критерий важнее первого. В случае если одному рангу соответствует группа критериев, вводят значения p и q. Значение p — это число критериев в группе с наивысшим рангом, т. е. самые важные критерии. Значение q — это число критериев в группе с наименьшим рангом. Так как в данной работе каждому рангу соответствует один критерий ($p = 1$, $q = 1$), то отношение z становится равным общему числу всех карт — T, которое, в свою очередь, равно сумме числа критериев и числа белых карт. Рассчитаем параметры, необходимые для вычисления ненормализованных весов рангов:

Таблица 2

Число белых карт

Критерий	Число белых карт
Тарифные планы *cp*	1
Время измерения *ct*	1
Тип связи *cc*	0
Скорость передачи данных *cs*	3

$$e = e_1 + e_2 + e_3 = 5;$$

$$T = 4 + 5 = 9;$$

$$z = \frac{\left(\sum_{i=0}^{q-1}(T - i)\right)}{\left(\sum_{i=0}^{p-1}(1 + i)\right)} = 9;$$

$$u = \frac{9 - 1}{5} = 1{,}6.$$

Используя общую формулу для расчета значений ненормализованных весов — $k(r) = 1 + u(e_0 + \cdots + e_{r-1})$, где $e_0 = 0$, получаем значения, приведенные в табл. 3.

Чем выше ранг, тем более значимым является критерий. Обозначим сумму ненормализованных весов как $K' = \sum_{i=1}^{n} k(r)$, она необходима для вычисления нормализованных весов по формуле $k_i^* = \frac{100}{K'} k(r)$. Для увеличения точности значащие цифры после запятой выбраны следующим образом: $w = 0$ (значащие цифры отсутствуют), $w = 1$ (одна значащая цифра), $w = 2$ (две значащие цифры). Уточненное выражение для суммы значений критериев имеет следующий вид: $K'' = \sum_{i=1}^{n} k_i''$, где k_i'' равно k_i^* с меньшим числом значащих цифр. Так как часть значащих цифр отбрасывается, значение суммы может быть меньше 100. Разницу обозначим следующим образом: $\varepsilon = 100 - K'' \leqslant 10^{-w} \times M$, где M — общее число критериев. Заметим, что значение $\nu = \varepsilon 10^w = (100 - K'')10^w \leqslant M$ и в случае данных расчетов для $w = 1$ равно $v = (100 - 99{,}8)10^1 = 2$. Что говорит о том, что для двух критериев необходимо использовать формулу $k_i = k_i'' + 10^{-1}$, и для оставшихся двух — формулу $k_i = k_i''$. Выбор двух критериев, для которых к значению добавляется 0,1, осуществляется согласно алгоритму, приведенному далее.

Таблица 3

Значения параметров для критериев

Критерий	Ранг r	Число белых карт	e_r	Ненормализованные веса $k(r)$
cp	1	1	2	1,00
ct	2	1	2	4,20
cc	3	0	1	7,40
cs	4	3	4	9,00
Сумма по всем критериям	—	5	9	21,60

Таблица 4

Значения весов для критериев

Критерий	Ранг r	Нормализованные веса k_i^*	Нормализованные веса k_i''	d_i	$\overline{d_i}$	Нормализованные веса k_i
cp	1	4,629629	4,6	0,015200	0,006399	4,6
ct	2	19,444444	19,4	0,002857	0,002285	19,4
cc	3	34,259259	34,2	0,001189	0,001729	34,3
cs	4	41,666666	41,6	0,000800	0,001599	41,7
Сумма по всем критериям	—	—	99,8	—	—	100,0

Шаг 1. Расчет отношений $d_i = \dfrac{10^{-w} - \left(k_i^* - k_i''\right)}{k_i^*}$ и $\overline{d_i} = \dfrac{\left(k_i^* - k_i''\right)}{k_i^*}$.

Шаг 2. Поиск множества значений $F = \{i\}$, такого, что $d_i \succ \overline{d_i}$. В случае данных расчетов $F = \{1, 2\}$.

Шаг 3. Поиск мощности множества F, $|F| = f$. В данном случае $f = 2$.

Шаг 4. Составление множества L из пар (i, d), располагающихся по увеличению d_i: $(4, d_4)$, $(3, d_3)$, $(2, d_2)$, $(1, d_1)$.

Шаг 5. Составление множества $\overline{L}$ из пар $\left(i, \overline{d}_i\right)$, располагающихся по уменьшению значений $\overline{d}_i$: $\left(1, \overline{d}_1\right), \left(2, \overline{d}_2\right), \left(3, \overline{d}_3\right), \left(4, \overline{d}_4\right)$.

Шаг 6. Для случая $f + v \leqslant M$: для двух критериев (так как 2 + 2 = 4) значение k_i'' остается прежним. Добавление 0,1 к значениям первых двух критериев (так как $v = 2$) множества $\overline{L}$, не входящих в множество F (это критерии типа связи ($r = 3$) и скорости ($r = 4$)). В итоге, сумма всех критериев получается равной 100, что отображено в табл. 4.

Расчет значений каждого из критериев. В качестве примера ниже показаны значения критериев только для одного маршрута (Ульяновск — Санкт-Петербург, измерения 5.1, 5.2, 6.1, 6.2, 8.1, 8.2) с использованием формул, описанных выше, получаем значения, представленные в табл. 5.

Например, критерий тарифного плана для провайдера 2 был рассчитан следующим образом:

$$\frac{\sum_{i=1}^{P_1} x_{1i}^2 + \sum_{i=1}^{P_2} x_{2i}^2}{P_1 + P_2} - \overline{x}^2 =$$

$$= \frac{(10\,000 + 14\,000)^2 + 18\,000^2}{2+1} - 14\,000^2 = 10\,666\,667.$$

Таблица 5

Числовые значения критериев

Провайдер	Критерий (вес)			
	cs (41,7)	*ct* (19,4)	*cp* (4,6)	*cc* (34,3)
1	20 000	0,83	10 666 667	5
2	14 000	1,17	10 666 667	5
Абсолютный провайдер	100 000	1,50	2 500 000 000	5

Расчет матриц согласия и несогласия. В качестве примера приведем расчет матриц согласия и несогласия также для одного маршрута (измерения 5.1, 5.2, 6.1, 6.2, 8.1, 8.2). В данном случае веса критериев обозначены следующим образом: k_{cs} — для критерия скорости передачи информации; k_{cp} — для критерия тарифных планов; k_{cc} — для критерия типа связи; k_{ct} — для критерия времени.

Для коэффициентов согласия имеем:

$$\alpha_{12} = \frac{1}{100}\left(k_{cs} + k_{cp} + k_{cc}\right) =$$

$$= \frac{1}{100}(41{,}7 + 4{,}6 + 34{,}3) = 0{,}806;$$

$$\alpha_{21} = \frac{1}{100}\left(k_{ct} + k_{cp} + k_{cc}\right) =$$

$$= \frac{1}{100}(19{,}4 + 4{,}6 + 34{,}3) = 0{,}583;$$

$$\alpha_{31} = \frac{1}{100}\left(k_{cs} + k_{ct} + k_{cp} + k_{cc}\right) =$$

$$= \frac{1}{100}(41{,}7 + 19{,}4 + 4{,}6 + 34{,}3) = 1{,}000;$$

$$\alpha_{13} = \frac{1}{100}k_{cc} = \frac{1}{100}34{,}3 = 0{,}343;$$

$$\alpha_{23} = \frac{1}{100}k_{cc} = \frac{1}{100}34{,}3 = 0{,}343;$$

$$\alpha_{32} = \frac{1}{100}\left(k_{cs} + k_{ct} + k_{cp} + k_{cc}\right) =$$

$$= \frac{1}{100}(41{,}7 + 19{,}4 + 4{,}6 + 34{,}3) = 1{,}000.$$

Итоговая матрица согласия имеет следующий вид:
$\begin{pmatrix} - & 0{,}806 & 0{,}343 \\ 0{,}583 & - & 0{,}343 \\ 1{,}000 & 1{,}000 & - \end{pmatrix}$.

Для коэффициентов несогласия имеем:

$$\beta_{12} = \frac{19{,}4\left|0{,}83 - 1{,}17\right|}{1{,}5 \cdot 100} = 0{,}04;$$

$$\beta_{21} = \frac{41{,}7\left|14\ 000 - 20\ 000\right|}{100\ 000 \cdot 100} = 0{,}03;$$

$$\beta_{31} = 0{,}0;$$

$$\beta_{32} = 0{,}0;$$

$$\beta_{23} = \max_{cs,ct,cp}\left\{\frac{41{,}7\left|14\,000 - 100\,000\right|}{100\,000 \cdot 100}; \frac{19{,}4\left|1{,}17 - 1{,}50\right|}{1{,}5 \cdot 100}; \frac{4{,}6\left|10\,666\,667 - 2\,500\,000\,000\right|}{2\,500\,000\,000 \cdot 100}\right\} = 0{,}36;$$

$$\beta_{13} = \max_{cs,ct,cp}\left\{\frac{41{,}7\left|20\ 000 - 100\ 000\right|}{100\ 000 \cdot 100}; \frac{19{,}4\left|0{,}83 - 1{,}50\right|}{1{,}5 \cdot 100}; \frac{4{,}6\left|10\ 666\ 667 - 2\ 500\ 000\ 000\right|}{2\ 500\ 000\ 000 \cdot 100}\right\} = 0{,}33.$$

Итоговая матрица несогласия имеет следующий вид: $\begin{pmatrix} - & 0{,}04 & 0{,}33 \\ 0{,}03 & - & 0{,}36 \\ 0{,}0 & 0{,}0 & - \end{pmatrix}$.

Расчет рейтингов по маршруту. Значения рейтингов для обоих провайдеров рассчитываются согласно представленной выше формуле:

$$R_1 = \frac{1}{2}\left[\frac{1}{2}\left(0{,}806 + 0{,}343 - 0{,}04 - 0{,}33\right) + 1\right]100 = 69;$$

$$R_2 = \frac{1}{2}\left[\frac{1}{2}\left(0{,}583 + 0{,}343 - 0{,}03 - 0{,}36\right) + 1\right]100 = 63.$$

Тестирование алгоритма. Для реализации алгоритма было создано веб-приложение. Тестирование проводили с использованием набора данных, состоящего из 3,2 млн измерений. Статистику собирали в режиме реального времени с помощью сервиса wiTest, включающего в себя сайт и набор мобильных приложений для измерения скорости. Для проверки аддитивности алгоритма было проведено сравнение выборок за четыре недели, полученных как отдельно, так и одним целым периодом (месяц). Расчеты показали одинаковый результат. Один из результатов расчета рейтингов представлен на рисунке, см. вторую сторону обложки.

Заключение

Разработан алгоритм многокритериальной оценки провайдеров связи сети Интернет. Данная методика разработана, реализована и успешно применяется компанией ООО Веллинк и в ближайшем будущем будет доступна пользователям сети Интернет для сравнения провайдеров в одном либо нескольких городах. По итогам работы программы рассчитываются рейтинги провайдеров, результаты которых представляются графически в виде гистограмм.

Список литературы

1. **Saaty T. L.** Decision making with the analytic hierarchy process // Int. J. Services Sciences. 2008. Vol. 1, N 1. P. 83—98.
2. **Roy B.** Multicriteria Methodology for Decision Aiding, Dordrecht: Kluwer Academic Publishers, 1996.
3. **Руа Б.** Классификация и выбор при наличии многих критериев (метод ЭЛЕКТРА) / Вопросы анализа и процедуры принятия решений. М.: Мир, 1976. С. 80—107.
4. **Кравченко Т. К., Авдеев Ю. В.** Аналитическое обоснование инвестиционной привлекательности банков // Аудит и финансовый анализ. 2012. № 3. С. 140—144.
5. **Simos J.** L'évaluation environnementale: Un processus cognitif négocié. Thèse de doctorat, Lausanne: DGF-EPFL, 1990.
6. **Simos J.** Evaluer l'impact sur l'environnement: Une approche originale par l'analyse multicritère et la négociation. Lausanne: Presses Polytechniques Universitaires Romandes, 1990.
7. **Figueira J., Roy B.** Determining the weights of criteria in the ELECTRE type methods with a revised Simos' procedure // European Journal of Operational Research. 2002. Vol. 139. P. 317—326.

Internet Provider Comparison Algorithm, Based on the Family of ELECTRE Methods

O. V. Efimova, office-nsk@wellink.ru, **S. A. Pirogov**, spirogov@wellink.ru, Wellink Technologies Ltd, 630090, Novosibirsk, Russian Federation, **S. A. Semenov,** office-nsk@wellink.ru, Wellink LLC, 141401, Moscow, Russian Federation

Corresponding author:

Pirogov Sergey A., CEO, Wellink Technologies Ltd, 630090, Novosibirsk, Russian Federation, e-mail: office-nsk@wellink.ru

Received on May 29, 2015
Accepted on October 19, 2015

The number of Internet providers grows rapidly with the improvement of software and equipment used for Internet access. As a result, questions appear about the quality of the services, as well as the possibility of comparing and selecting the most appropriate provider according to users' particular needs and requirements. A multicriteria method for comparing Internet providers is presented in this work. This method is based on a family of ELECTRE methods which were designed to facilitate decision making process having several alternatives, according to several criteria. The following criteria are used to compare providers: 1) speed of data transfer; 2) time, when the measurement of speed is performed; 3) variety of tariff plans. Method allows users to compare any number of providers, taking into account their personal opinion, as well as a set of available measurements. For the latter, criteria weights are calculated according to Simos' procedure, which uses an imaginary set of playing cards. The method of calculating provider ratings is developed and implemented by Wellink LLC and can be used by Internet users to compare different providers in one or several cities. As a result, this program calculates provider ratings and plots them in the form of histograms.

Keywords: *method ELECTRE, provider rating, multi-criteria decision making, Simos' procedure, measurement of Internet connection speed*

For citation:

Efimova O. V., Pirogov S. A., **Semenov S. A.** Internet Provider Comparison Algorithm, Based on the Family of ELECTRE Methods, *Programmnaya Ingeneria*, 2016, vol. 7, no 1, pp. 37-45

DOI: 10.17587/prin.7.37-45

References

1. **Saaty T. L.** Decision making with the analytic hierarchy process, *Int. J. Services Sciences*, 2008, vol. 1, no. 1, pp. 83—98.

2. **Roy B.** Multicriteria Methodology for Decision Aiding, Dordrecht: Kluwer Academic Publishers, 1996.

3. **Rua B.** Klassifikaciya i vybor pri nalichii mnogih kriteriev (metod ELECTRE) (Classification and selection in the presence of many criteria (metod ELECTRE)). *Voprosy Analiza i Procedury Prinyatiya Reshenij.* Moscow, Mir, 1976, pp. 80—107 (in Russian).

4. **Kravchenko T. K., Avdeev Yu. V.** Analiticheskoe obosnovanie investicionnoj privlekatel'nosti bankov (Analytical study of investment attractiveness of banks), *Audit i Finansovyj Analiz*, 2012, no. 3, pp. 140—144 (in Russian).

5. **Simos J.** Levaluation environnementale: Un processus cognitif negocie. These de doctorat, Lausanne, DGF-EPFL, 1990.

6. **Simos J.** Evaluer limpact sur lenvironnement: Une approche originale par l'analyse multicritère et la negociation. Lausanne, Presses Polytechniques Universitaires Romandes, 1990.

7. **Figueira J., Roy B.** Determining the weights of criteria in the ELECTRE type methods with a revised Simos' procedure. *European Journal of Operational Research,* 2002, vol. 139, pp. 317—326.

Итоги работы V Всероссийской конференции с международным участием "Знания—Онтологии—Теории" (ЗОНТ-15)

С 6 по 8 октября 2015 г. в Новосибирске прошла V Всероссийская конференция с международным участием "Знания—Онтологии—Теории". Конференцию проводил Институт математики им. С. Л. Соболева СО РАН при поддержке Российского Фонда Фундаментальных Исследований (проект 15-01-20804). На ней было продолжено обсуждение тематики и проблем, которые рассматривались на Всесоюзных и Всероссийских симпозиумах "Методы обнаружения закономерностей" в период с 1976 по 2002 гг., а также на предыдущих Всероссийских конференциях "Знания—Онтологии—Теории", прошедших в 2007—2013 гг.

В конференции участвовали 202 человека. Большинство участников было из Новосибирска. В работе конференции приняли участие 9 зарубежных ученых (США, Индия, Болгария, Казахстан) и 65 участников из 12 городов России (Москва, Санкт-Петербург, Краснодар, Воронеж, Самара, Пермь, Омск, Томск, Иркутск и Владивосток).

Было сделано 48 докладов (из них 14 пленарных) на самые актуальные темы — математические методы анализа и представления данных, анализ формальных понятий, методы обработки естественного языка, извлечение знаний, разработка онтологий и построение теорий предметных областей.

По итогам конференции был издан сборник трудов в двух томах (также издан в электронном виде), содержащий тексты всех прошедших рецензирование докладов участников конференции. Опубликованные труды участников конференции предполагается проиндексировать в базе данных РИНЦ. Избранные работы будут опубликованы в журнале "Программная инженерия".

Участниками конференции был отмечен высокий научный уровень как пленарных докладов (среди докладчиков — академик и член-корреспондент РАН, 10 докторов наук и 9 кандидатов наук), так и секционных сообщений (из авторов — 14 докторов наук и 20 кандидатов наук, много студентов, аспирантов и молодых исследователей). Отличный уровень подготовки продемонстрировали молодые ученые (доклады представили 16 аспирантов и 3 студента), что показывает хорошие перспективы развития направления, заинтересованность молодого поколения в данных областях науки.

Проблематика исследований, затронутая при проведении конференции, охватывает самые актуальные и современные вопросы в теории онтологий предметных областей, инженерии знаний, анализе данных, методах извлечения и обработки знаний, построении эмпирических теорий, исследовании процессов познания, анализе формальных понятий. Основными задачами конференции являлись обсуждение актуальных проблем, ознакомление с новейшими достижениями ведущих научных школ России и мира в этих областях, обмен научным опытом, информирование специалистов о перспективных научно-технических разработках, инновационных проектах вузов, промышленных предприятий, установление прочных научных связей между исследователями из различных научных школ нашей страны. Помимо этого, участие в конференции поспособствовало более продуктивной работе молодых специалистов.

Значительное число докладов на конференции было посвящено онтологиям и их практическим применениям. Был представлен подход к разработке программных систем на основе многоуровневых онтологий, для спецификации которых предложен многосортный логический язык. Данный подход использовался при разработке модели предметной области "Химия", при создании нескольких программных систем, решающих задачи в этой предметной области. Были показаны возможности применения данного подхода к разработке программных систем и в других предметных областях.

Большой интерес участников конференции вызвали доклады, посвященные практическому использованию формальных моделей и онтологий, их программных реализаций в составе интеллектуальной системы тематического анализа наукометрических данных "ИСТИНА", которая активно развивается в МГУ им. М. В. Ломоносова.

Были рассмотрены теоретико-модельные методы разработки онтологий и онтологических моделей в медицине. Эти методы основаны на представлении знаний при помощи фрагментов атомарных диаграмм алгебраических систем, а также на представлении знаний о пациентах в виде булевозначной прецедентной модели. В рамках представленного исследования была предложена четырехуровневая модель представления знаний, разработаны онтология и онтологическая модель предметной области "Деформации позвоночника и дегенеративные заболевания позвоночника".

На конференции были представлены новая расширенная онтология профессиональных виртуаль-

ных сред, а также новая онтология WIMP-интерфейса виртуальных сред. Приведенные онтологии предназначены для формирования декларативных моделей профессиональных виртуальных сред, на основе которых строят и интерпретируют полноценные программные интеллектуальные приложения с использованием виртуальной реальности с компьютерной 3D-графикой.

Также была предложена онтологическая модель пространства знаний для ситуационного управления в энергетике. При этом онтологический инжиниринг рассматривается как средство структуризации основных понятий данной области. Знания, необходимые для решения проблем ситуационного управления, группируются в виде пространства компонентов, объединенных общими задачами и целями разработки, формально описанного некоторой системой онтологий.

Были рассмотрены достоинства и недостатки ряда популярных онтологий. Так, в качестве главного недостатка онтологии FOAF (*Friend Of A Friend ontology*) отмечалось отсутствие возможности атрибутирования отношений, что значительно сужает изобразительные возможности этой онтологии. Было отмечено также, что в онтологии dbpedia_3.8, которая используется для формализации представления информации из популярного ресурса "Википедия", вводится определение участника события в виде обычной строки, а не ссылки, что не позволяет его полноценно представлять в виде объекта со свойствами, каковым обычно описывается, например, персона. В этой онтологии также наблюдается наличие очень большого числа обладающих спецификой классов и отношений, ориентированных на США и слабоформализуемых на фактографическом уровне, что сильно ограничивает область ее применения. Были приведены метаонтология SKOS (*Simple Knowledge Organization System*) в качестве примера того, как "правильно" структурировать области знаний, а также предложенная в ИСИ СО РАН базовая онтология неспецифических сущностей BONE, которая, по мнению ее авторов, лишена отмеченных недостатков.

Участниками конференции были рассмотрены проблемные вопросы и задачи автоматического построения таксономий (графов) сущностей достаточно узких предметных областей и их использование для повышения релевантности поиска по заданной тематике. Было отмечено, что конструирование таксономий выполняется на основе машинного обучения деревьев синтаксического разбора, когда начальная таксономия, состоящая из небольшого числа исходных сущностей, достраивается путем извлечения из страниц, найденных при поиске в Интернете, новых сущностей, ассоциированных с базовыми (входящих с ними в одно дерево разбора). Высокая релевантность результатов поиска достигается за счет того, что при сравнении запроса и контента веб-страницы используются не ключевые слова, а поддеревья деревьев их синтаксического разбора.

В рамках работы конференции были приведены результаты исследований возможности применения онтологий к разработке научных интернет-ресурсов. Было показано, что онтологии могут использоваться не только для формализации, систематизации и сбора различных видов знаний и данных моделируемой предметной области и средств их обработки и анализа, но и для организации удобного содержательного доступа к ним.

Также были рассмотрены проблемы построения и использования тезаурусов в научно-образовательных информационных системах, приведен обзор стандартов для представления тезаурусов, выполнен анализ и сравнение различных подходов к описанию схемы данных тезаурусов на основе объектной модели. При этом особое внимание было уделено работе со словарями ключевых терминов, которые используются для систематизации и классификации информационных ресурсов.

Участники конференции представили результаты по разработке методов поиска новых знаний в области молекулярно-биологических исследований на основе автоматизации процесса реконструкции сетей ассоциативных взаимосвязей между молекулярно-генетическими объектами из научных текстов и фактографических баз данных. В частности, методы создания и пополнения словарей названий молекулярно-генетических объектов на основе анализа найденных текстов, методы создания тематических словарей по ключевым словам, а также методы извлечения информации о взаимосвязях молекулярно-генетических объектов с использованием шаблонов.

Была предложена модель описания процесса извлечения информации из текста с помощью системы покрытий текста и онтологии. Также была описана мультиагентная система, обеспечивающая семантический этап обработки текста, и проведено описание результатов исследования ее свойств как информационной системы Скотта.

В рамках конференции был организован вечер памяти Николая Григорьевича Загоруйко. Николай Григорьевич был одним из идейных вдохновителей нашей конференции, одним из прародителей направления распознавания образов и машинного обучения в Советском Союзе, замечательным ученым, учителем и человеком.

Идеи, которые Николай Григорьевич Загоруйко продвигал в последние годы работы, заключались в том, что модель для решения различных задач анализа данных должна быть универсальной, отражать человеческую способность обрабатывать информацию и благодаря этому успешно заменять человека при необходимости искать закономерности в больших массивах данных. В своих исследо-

ваниях он опирался на гипотезу, что когнитивные способности человека основаны на двух ключевых принципах. Первый из них заключается в использовании специфической меры для оценки сходства между объектами, а второй — в стремлении к максимальной компактности и простоте описания мира в терминах этой меры сходства.

В качестве модели человеческого способа оценки сходства между объектами Н. Г. Загоруйко предложил использовать функцию конкурентного сходства (FRiS-функцию, *Function of Rival Similarity*), на основе которой вычислялась количественная оценка компактности классов. Эти два элемента — FRiS-сходство и FRiS-компактность — лежат в основе алгоритмов для решения различных типов задач анализа данных, таких как построение решающего правила (алгоритм FRiS-Stolp), таксономия (алгоритм FRiS-Tax), частичное обучение (алгоритм FRiS-TDR), выбор информативных признаков (алгоритм FRiS-GRAD), цензурирование (алгоритм FRiS-Censor), заполнение пробелов (алгоритм 3D-ZET). Примеры успешного использования вышеперечисленных алгоритмов для решения прикладных задач из области медицины, криминалистики, консалтинга и некоторых других подтверждают универсальность и эффективность модели конкурентного сходства.

На заключительном заседании конференции — круглом столе были подведены итоги и намечены планы на будущее. Работа конференции была признана успешной и плодотворной. Как опытные ученые, так и молодые участники конференции отметили большую полезность данного научного мероприятия. Были высказаны пожелания по расширению тематики конференции ее географии, о большем привлечении специалистов из реального производства.

Был отмечен высокий уровень докладов, представленных на конференции. Данный уровень свидетельствует о высокой квалификации российских специалистов по затронутой тематике, в том числе и молодых ученых.

Было принято решение провести следующую, VI Всероссийскую конференцию с международным участием "Знания—Онтологии—Теории" в 2017 г., при этом увеличив сроки проведения конференции.

Д-р физ.-мат. наук,
вед. науч. сотр. Д. Е. Пальчунов[1],
канд. техн. наук, зав. лаб. Ю. А. Загорулько[2],
канд. техн. наук, ст. науч. сотр. И. А. Борисова[1],
аспирант Ч. А. Найданов[3]
[1] *Институт математики*
им. С. Л. Соболева СО РАН
[2] *Институт систем информатики*
им. А. П. Ершова СО РАН
[3] *Новосибирский государственный университет*

ИНФОРМАЦИЯ

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *Е. В. Комиссарова*

Сдано в набор 06.11.2015 г. Подписано в печать 18.12.2015 г. Формат 60×88 1/8. Заказ PI116
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**