ISSN 2220-3397

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

ПрИн
4
2018
Том 9

Clarivate Analytics
WEB OF SCIENCE
Russian Science Citation Index

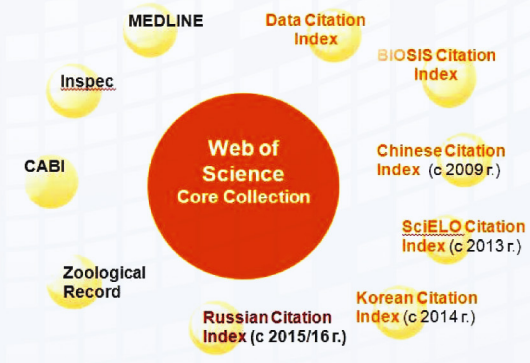
Уважаемые авторы и читатели журнала «Программная инженерия»!

С января 2018 г. наш журнал включен в список RSCI журналов, которые индексируются на платформе Web of Science. В первоначальный список журнал не попал по недосмотру: он оказался вне внимания комиссий и по инженерным специальностям, и по математике – информатике. Безусловно, что по этой причине журнал понес определенные репутационные издержки, однако правда все же восторжествовала. Вместе с тем новый статус накладывает на нас некоторую дополнительную меру ответственности за его содержание, включая научно-техническую, инновационную и образовательную компоненты (составляющие) каждой из публикаций.

Как мы и обещали в преддверии нынешнего года, в настоящее время мы изыскиваем возможности публикации на английском языке лучших (по оценкам экспертов) статей, прошедших апробацию в нашем журнале. Анализируются и вариант кооперации с каким-то авторитетным зарубежным издательством (изданием), и вариант ежеквартального собственного издания в России с публикацией в режиме открытого доступа к его материалам из сети Интернет. Целью такой инициативы является расширение внимания к российским результатам в инженерии программ со стороны международного научно-технического сообщества. В дополнение к отмеченным выше инициативам, предоставляем возможность авторам (в том числе и россиянам) уже сейчас подавать заявки на публикацию статьи в журнале «Программная инженерия» на английском языке. Для тех, кто выражает такое желание, предлагается кроме сопроводительных материалов, ранее рекомендуемых для представления статьи к публикации в журнале, дополнительно готовить Вашу версию статьи на английском языке. Эта версия будет в контакте с Вами дорабатываться экспертами, обладающими должными знаниями английского языка, для её окончательной публикации в журнале.

С пожеланием успехов, искренне Ваш
Гл. редактор журнала «Программная инженерия»,
д-р физ.-мат. наук, проф. *В.А. Васенин*

Платформа **Web of Science**



Новая база данных Russian science citation index (RSCI) на платформе Web of science

- Чтобы быть ещё ближе к российскому научному сообществу
- Чтобы получить возможность оценивать российскую науку по ещё более широкой выборке
- Чтобы дать российским учёным и журналам дополнительную возможность быть увиденными мировым научным сообществом

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

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

Редакционный совет
Садовничий В.А., акад. РАН (*председатель*)
Бетелин В.Б., акад. РАН
Васильев В.Н., чл.-корр. РАН
Жижченко А.Б., акад. РАН
Макаров В.Л., акад. РАН
Панченко В.Я., акад. РАН
Стемпковский А.Л., акад. РАН
Ухлинов Л.М., д.т.н.
Федоров И.Б., акад. РАН
Четверушкин Б.Н., акад. РАН

Главный редактор
Васенин В.А., д.ф.-м.н., проф.

Редколлегия
Антонов Б.И.
Афонин С.А., к.ф.-м.н.
Бурдонов И.Б., д.ф.-м.н., проф.
Борзовс Ю., проф. (Латвия)
Гаврилов А.В., к.т.н.
Галатенко А.В., к.ф.-м.н.
Корнеев В.В., д.т.н., проф.
Костюхин К.А., к.ф.-м.н.
Махортов С.Д., д.ф.-м.н., доц.
Манцивода А.В., д.ф.-м.н., доц.
Назиров Р.Р., д.т.н., проф.
Нечаев В.В., д.т.н., проф.
Новиков Б.А., д.ф.-м.н., проф.
Павлов В.Л. (США)
Пальчунов Д.Е., д.ф.-м.н., доц.
Петренко А.К., д.ф.-м.н., проф.
Позднеев Б.М., д.т.н., проф.
Позин Б.А., д.т.н., проф.
Серебряков В.А., д.ф.-м.н., проф.
Сорокин А.В., к.т.н., доц.
Терехов А.Н., д.ф.-м.н., проф.
Филимонов Н.Б., д.т.н., проф.
Шапченко К.А., к.ф.-м.н.
Шундеев А.С., к.ф.-м.н.
Щур Л.Н., д.ф.-м.н., проф.
Язов Ю.К., д.т.н., проф.
Якобсон И., проф. (Швейцария)

Редакция
Лысенко А.В., Чугунова А.В.

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/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования и базу данных RSCI на платформе Web of Science.
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.

© Издательство "Новые технологии", "Программная инженерия", 2018

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 9
N 4
2018

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

Editorial Council:

SADOVNICHY V. A., Dr. Sci. (Phys.-Math.), Acad. RAS (*Head*)
BETELIN V. B., Dr. Sci. (Phys.-Math.), Acad. RAS
VASIL'EV V. N., Dr. Sci. (Tech.), Cor.-Mem. RAS
ZHIZHCHENKO A. B., Dr. Sci. (Phys.-Math.), Acad. RAS
MAKAROV V. L., Dr. Sci. (Phys.-Math.), Acad. RAS
PANCHENKO V. YA., Dr. Sci. (Phys.-Math.), Acad. RAS
STEMPKOVSKY A. L., Dr. Sci. (Tech.), Acad. RAS
UKHLINOV L. M., Dr. Sci. (Tech.)
FEDOROV I. B., Dr. Sci. (Tech.), Acad. RAS
CHETVERTUSHKIN B. N., Dr. Sci. (Phys.-Math.), Acad. RAS

Editor-in-Chief:

VASENIN V. A., Dr. Sci. (Phys.-Math.)

Editorial Board:

ANTONOV B.I.
AFONIN S.A., Cand. Sci. (Phys.-Math)
BURDONOV I.B., Dr. Sci. (Phys.-Math)
BORZOVS JURIS, Dr. Sci. (Comp. Sci), Latvia
GALATENKO A.V., Cand. Sci. (Phys.-Math)
GAVRILOV A.V., Cand. Sci. (Tech)
JACOBSON IVAR, Dr. Sci. (Philos., Comp. Sci.), Switzerland
KORNEEV V.V., Dr. Sci. (Tech)
KOSTYUKHIN K.A., Cand. Sci. (Phys.-Math)
MAKHORTOV S.D., Dr. Sci. (Phys.-Math)
MANCIVODA A.V., Dr. Sci. (Phys.-Math)
NAZIROV R.R. , Dr. Sci. (Tech)
NECHAEV V.V., Cand. Sci. (Tech)
NOVIKOV B.A., Dr. Sci. (Phys.-Math)
PAVLOV V.L., USA
PAL'CHUNOV D.E., Dr. Sci. (Phys.-Math)
PETRENKO A.K., Dr. Sci. (Phys.-Math)
POZDNEEV B.M., Dr. Sci. (Tech)
POZIN B.A., Dr. Sci. (Tech)
SEREBRJAKOV V.A., Dr. Sci. (Phys.-Math)
SOROKIN A.V., Cand. Sci. (Tech)
TEREKHOV A.N., Dr. Sci. (Phys.-Math)
FILIMONOV N.B., Dr. Sci. (Tech)
SHAPCHENKO K.A., Cand. Sci. (Phys.-Math)
SHUNDEEV A.S., Cand. Sci. (Phys.-Math)
SHCHUR L.N., Dr. Sci. (Phys.-Math)
YAZOV Yu. K., Dr. Sci. (Tech)

Editors: LYSENKO A.V., CHUGUNOVA A.V.

CONTENTS

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

УДК 004.03 DOI: 10.17587/prin.9.147-155

С. Д. Кузнецов, д-р техн. наук, проф., гл. науч. сотр., e-mail: kuzloc@ispras.ru,
Институт системного программирования им. В. П. Иванникова РАН, Москва,
Московский государственный университет им. М. В. Ломоносова,
Московский физико-технический институт,
Высшая школа экономики, Москва

Новые устройства хранения данных и их влияние на технологию баз данных

Продемонстрировано, что технология наиболее распространенных в настоящее время SQL-ориентированных систем управления базами данных (СУБД) неразрывно связана с технологией HDD (Hard Disk Drive). Особенности HDD влияют на структуры данных и алгоритмы выполнения операций, на методы управления буферным пулом СУБД, на управление транзакциями, оптимизацию запросов и т. д. Альтернативой дисковым СУБД являются in-memory-СУБД, хранящие базы данных целиком в основной памяти. Несмотря на наличие у in-memory-СУБД ряда преимуществ перед дисковыми СУБД, в настоящее время конкуренция между ними практически отсутствует. Это прежде всего связано с естественными ограничениями на размеры баз данных, свойственными in-memory-СУБД. В настоящее время появились новые виды аппаратуры хранения данных: SSD — блочные твердотельные накопители, SCM — энергонезависимая основная память. Характеристики SSD делали целесообразной разработку СУБД, которая была рассчитана на их исключительное использование, однако до сих пор такая СУБД не создана. Накопители SSD просто используются вместо HDD в СУБД, не учитывающих их особенности. Наличие SCM позволяет радикально упростить архитектуры СУБД и значительно повысить их производительность. Для этого нужно пересмотреть многие идеи, используемые в дисковых СУБД.

Ключевые слова: *SQL-ориентированные СУБД, магнитные диски с подвижными головками, оценочная оптимизация запросов, СУБД с хранением баз данных в основной памяти, твердотельные накопители на флэш-памяти, энергонезависимая основная память*

Введение

Технология наиболее распространенных SQL-ориентированных ("реляционных") СУБД неразрывно связана с технологией устройств хранения данных на магнитных дисках с подвижными головками (*Hard Disk Drive with Movable Heads*; в настоящее время такие устройства обычно называются просто *Hard Disk Drive*, HDD).

Первые HDD были выпущены компанией IBM в 1956 г. В технологии HDD преодолевались недостатки ранних устройств хранения данных — магнитных лент (*magnetic tape data storage*; основное ограничение — чисто последовательный доступ) и магнитных барабанов (*drum memory*; недостаток — ограниченная емкость). Технология обеспечивала меньшую, чем у магнитных лент, но значительно большую, чем у магнитных барабанов, емкость, а также меньшую, чем у магнитных барабанов, но значительно большую, чем у магнитных лент, скорость выполнения произвольных обменов данными между основной и внешней памятью. Если добавить к этому умеренную стоимость HDD, то эти устройства являлись вполне подходящими для хранения баз данных.

На технологию СУБД повлияли технологические особенности HDD. Во-первых, HDD обеспечивают внешнюю память, обмены с которой обычно осуществляются блоками байт одного и того же размера. Эта особенность приводит как минимум к двум архитектурным решениям.

1. Для хранения баз данных и убыстрения обработки запросов выбираются структуры данных и алгоритмы выполнения операций, для которых естественна блочная природа внешней памяти. В частности, для организации индексов наиболее часто применяют разновидности B-деревьев [1].

2. Для балансировки относительно небольшой скорости выполнения произвольных обменов с внешней памятью и относительно высокой скорости обработки данных в основной памяти СУБД осуществляет собственную буферизацию (кэширование) блоков внешней памяти базы данных в основной памяти [2, subsection 3.3, Buffer Management, 3, п. 10.1.1 Управление буферным пулом базы данных]. При организации такой буферизации блочное устройство внешней памяти является принципиальным.

Во-вторых, при выполнении обменов с внешней памятью HDD дисковая аппаратура выполняет три

основных операции: подвод головок к требуемому цилиндру дискового пакета (*seek*); прокручивание дискового пакета на требуемое угловое расстояние (*latency*); чтение или запись данных с их передачей в основную память или из нее (*data transfer*). При выполнении произвольного обмена время выполнения первых двух операций исчисляется миллисекундами. Этот факт означает, что время чтения произвольного блока данных из внешней памяти или его записи на несколько десятичных порядков больше времени выполнения соответствующего цикла переписи в основной памяти. Поэтому при выполнении любой операции уровня SQL над базой данных определяющим накладным расходом является число требуемых обменов с внешней памятью. На этом наблюдении основана оценочная оптимизация запросов (*cost-based query optimization*), применяемая во всех развитых SQL-ориентированных СУБД и основанная на пионерской работе [4].

Приведенных замечаний достаточно, чтобы убедиться в глубокой зависимости наиболее распространенной технологии SQL-ориентированных СУБД от особенностей HDD. Ориентация на использование этих устройств хранения данных влияет как на общую архитектуру СУБД, так и на выбор основных структур данных и алгоритмов.

В конце 1970-х — 1980-х гг. предпринимались попытки создания специализированной аппаратуры для поддержки СУБД, включая аппаратуру хранения данных на дисках с фиксированными головками (*head-per-track disk*). Более того, существовали прототипы таких устройств, в которых в магнитные головки встраивались специальные микропроцессоры, фильтрующие данные "на лету" при их считывании с диска (*processor-per-track systems* и *processor-per-head systems*) [5]. Однако к началу 1990-х гг. стала ясна бесперспективность такого подхода [6], и на протяжении следующих двух десятилетий технология СУБД базировалась главным образом на устройствах хранения данных категории HDD.

В то же время появилась и развилась альтернативная технология СУБД с хранением баз данных в обычной энергозависимой основной памяти (*in-memory DBMS*) [7]. В таких СУБД структуры данных и алгоритмы выполнения операций отличаются от используемых в дисковых СУБД. В частности, при выборе структур данных нужно учитывать наличие кэш-памяти в процессорах [8]. Должны отличаться и принципы оптимизации запросов, хотя публикаций об оптимизаторах запросов в in-memory-СУБД настолько мало, что, похоже, соответствующие принципы просто не сформировались.

Вероятно, наиболее зрелыми представителями этой категории СУБД являются TimesTen [9], существующая с 1996 г. и приобретенная Oracle в 2005 г., а также solidDB [10], существующая с 1992 г. и приобретенная IBM в 2007 г. Эти системы поддерживают очень быстрое выполнение запросов к базам данных, поскольку база данных и все индексы целиком сохраняются в основной памяти. Однако при выполнении операций изменения баз данных требуются обращения к внешней памяти. Для обеспечения долговечности (*durability*) транзакций журнал изменений базы данных сохраняется в дисковой памяти. Как следствие, скорость выполнения таких операций не отличается от соответствующей скорости СУБД, которые хранят базы данных на дисках.

Особняком с отмеченных позиций стоит in-memory-СУБД VoltDB [11], являющаяся транзакционной массивно-параллельной системой без общих ресурсов между узлами (*shared nothing*). В этой системе свойство долговечности транзакций поддерживается на основе репликации данных в нескольких узлах, а внешняя память вообще не используется. Подробности организации VoltDB (и ее прототипа H-Store) рассмотрены в работе [12].

По слухам, в аналогичном направлении движется и TimesTen. Эти слухи частично подтверждаются наличием в семействе Oracle TimesTen In-Memory Database опции High Availability (высокого уровня доступности) [13], которая обеспечивается за счет репликации в кластерной среде. Интересно, что хотя аналогичные слухи по поводу solidDB пока отсутствуют, для этой системы также поддерживается опция высокой доступности [14].

Следует заметить, что несмотря на наличие у in-memory-СУБД ряда преимуществ перед дисковыми СУБД, в настоящее время конкуренция между ними практически отсутствует. Это прежде всего связано с естественными ограничениями на размеры баз данных, свойственными in-memory-СУБД.

В первые десятилетия XXI века в технологии аппаратных средств хранения данных произошли (и продолжают происходить) существенные изменения. Появились так называемые блочные твердотельные накопители (*Solid-State Drive*, SSD), основанные на технологии флэш-памяти и сравнительно быстро догнавшие HDD по показателю максимальной емкости (до 32 Тбайт в 2016 г.), превосходя их по ряду других показателей (проигрывая в основном только в цене). В следующем разделе будут кратко обсуждены потенциальные возможности применения SSD в архитектуре СУБД, компоненты СУБД, на которые должен был бы максимально подействовать переход от HDD к SSD, а также состояние дел в технологии СУБД через десять лет после того, как SSD на флэш-памяти стали реально доступны.

В последние годы полностью реальной стала перспектива появления на рынке оперативной энергонезависимой памяти (*Non Volatile Random Access Memory*, NVRAM), которую, возможно, более выразительно, хотя и слишком длинно по-русски называют "основной памятью с возможностью долговременного хранения данных" (*Storage Class Memory*, SCM). Такая память допускает байтовую адресацию, прямо доступна для команд процессоров, однако при этом сохраняет содержимое после отключения электропитания.

Использование SCM открывает путь к построению СУБД, основанных на одноуровневой памяти.

Эти СУБД могут оказаться гораздо быстрее дисковых при более простой организации. Перспективам появления таких СУБД и имеющимся проблемам посвящен раздел 2 статьи.

1. SSD на флэш-памяти и технология СУБД

Как и HDD, SSD — это блочное внешнее запоминающее устройство, сохраняющее данные после выключения электропитания. Основными отличиями SSD от HDD являются следующие:

- в SSD отсутствуют механические компоненты, поэтому для любого блока скорость выполнения обмена с SSD одна и та же;
- если среднее время обмена с произвольным блоком HDD составляет около 10 мс как для чтения, так и для записи, то время чтения произвольного блока в современных SSD — около 20 мс (на три десятичных порядка меньше, чем у HDD), а время записи — около 200 мкс (на два десятичных порядка меньше, чем у HDD);
- пока SSD стоят дороже, чем HDD (на 2016 г. примерно в 10 раз), но стоимость HDD в пересчете на терабайт объема поддерживаемой памяти в последние годы стабилизировалась, а SSD дешевеют;
- в настоящее время SSD являются существенно менее надежными устройствами, чем HDD.

1.1. SSD-ориентированные СУБД

Только последняя в списке характеристика может в принципе препятствовать полномасштабному применению SSD в СУБД. Непонятно, удастся ли разработчикам аппаратуры SSD избавиться от этого недостатка. Однако первые две характеристики кажутся настолько привлекательными, что еще 10 лет назад автор пытался (хотя не слишком успешно) убедить своих студентов заняться исследованиями архитектуры СУБД, в которой для хранения баз данных используются SSD.

Понятно, что в наибольшей степени особенности SSD могли бы повлиять на управление внешней памятью, управление буферами основной памяти и оптимизатор запросов. В существующих дисковых СУБД, поскольку при выполнении запросов часто приходится проводить полный просмотр таблиц без использования индексов, стремятся располагать на диске блоки одной таблицы так, чтобы при переходе от текущего блока к следующему не требовалось сильно перемечать магнитные головки. В СУБД, основанной на использовании только SSD, блоки одной таблицы могут располагаться во внешней памяти произвольным образом.

Время записи блока во внешнюю память SSD на десятичный порядок больше времени чтения блока за счет потребности предварительной подготовки сектора внешней памяти, в который будет проводиться запись [15]. При управлении буферами основной памяти в СУБД, ориентированной на использование SSD, имеет смысл заранее подготавливать к записи секторы внешней памяти и при выталкивании из буфера во внешнюю память измененного образа ранее прочитанного блока внешней памяти писать его не в тот сектор, из которого он был прочитан, а в некоторый сектор, уже подготовленный к записи.

Однако следует заметить, что распределение внешней памяти и управление буферами основной памяти — это мелочи по сравнению с оптимизацией запросов. Как отмечалось во введении, современные оценочные оптимизаторы основываются на предположении, что произвольные обмены с внешней памятью выполняются так долго, что стоимость плана выполнения запроса можно оценивать числом требуемых для этого обменов, пренебрегая временем, которое потребуется для процессорной обработки данных. Чтение из внешней памяти SSD выполняется в 1000 раз быстрее, чем с использованием HDD. Поэтому при переходе от HDD к SSD это предположение нужно было бы подвергнуть строгой ревизии.

Имеется в виду, что прямой перенос оценок планов выполнения запросов из среды HDD в среду SSD может привести к отрицательным результатам. Неправильный учет временных затрат на обмены с внешней памятью и процессорную обработку данных в основной памяти может привести к выбору оптимизатором запросов заведомо не оптимальных планов выполнения запросов, что приведет к недоиспользованию потенциала SSD. Конечно, запросы не станут выполняться медленнее, чем при применении HDD, но ради этого не следует менять аппаратуру управления внешней памятью. Другими словами, для эффективного использования SSD оптимизаторы запросов нужно значительно переделывать.

Несмотря на привлекательность идеи замены HDD на SSD в аппаратной поддержке СУБД, практически отсутствуют проекты (как коммерческие, так и исследовательские) по разработке SSD-ориентированных СУБД. Автору удалось обнаружить только проект FlashyDB, выполняемый в немецком университете Ройтлингена [16]. Объявлены следующие цели этого проекта:

- исследовать влияние SSD на основе флэш-памяти на архитектуры и производительность существующих систем баз данных, реляционных хранилищ данных (*data warehouse*) и систем с поколоночным хранением таблиц (*column store*);
- разработать алгоритмы и структуры данных, обеспечивающие оптимальное использование характеристик SSD на основе флэш-памяти в сценариях OLTP и OLAP;
- реализовать прототип системы.

Список исследовательских тем, затрагиваемых в проекте, включает архитектуры систем баз данных, обработку транзакций, управление мультидоступом, восстановление после сбоев, управление буферами, индексацию, оптимизацию запросов, размещение данных. Как видно, направленность проекта вполне соответствует отмеченным выше соображениям. По-видимому, одной из первых статей, посвященных проекту FlashyDB, была работа [17]. Полный список

опубликованных статей доступен на сайте проекта [16]. Как показывает этот список, далеко не во всех намеченных направлениях исследований получены существенные результаты.

Возможно, недостаточная активность исследователей по построению истинных SSD-ориентированных СУБД связана с тем обстоятельством, что до недавнего прошлого максимальная емкость устройств хранения данных во флэш-памяти ограничивалась 1 Тбайтом. Однако технология быстро развивается, и уже в 2016 г. компания Samsung представила SSD емкостью 32 Тбайта и обещает довести емкость своих SSD до 100 Тбайт. Seagate показала SSD емкостью 60 Тбайт. Возможно, это "подстегнет" сообщество баз данных.

1.2. Двухуровневый кэш на основе SSD

До недавнего времени емкость SSD была сравнительно невелика. В связи с этим достаточно популярной была идея использования SSD в составе иерархического двухуровневого буфера в традиционных СУБД, ориентированных на использование HDD [18]. Суть идеи достаточна проста. Если по каким-то причинам необходимо продолжать использовать в СУБД для хранения баз данных HDD, но при этом получать достаточную пользу от применения SSD, то почему бы временно не хранить во флэш-памяти часть блоков базы данных, которая, вероятно, требуется в данный момент времени.

Для реализации этой идеи достаточно изменить лишь один компонент традиционной дисковой СУБД — менеджер буферов в основной памяти. Буфер становится двухуровневым: кэш первого уровня размещается в основной памяти, а кэш второго уровня — во флэш-памяти SSD. Блоки базы данных, необходимые для выполнения операций над базой данных, считываются из дисковой внешней памяти в буферные страницы кэша первого уровня. При нехватке памяти в кэше первого уровня происходит замещение какой-либо буферной страницы. Если ее содержимое изменялось после чтения из внешней памяти, то страница перемещается в кэш второго уровня (с учетом замечаний об управлении буферами, отмеченных в пп. 1.1). Если не хватает памяти в кэше второго уровня, то замещаемый блок перемещается во внешнюю память HDD.

В работе [18] приведен обзор алгоритмов управления подобным двухуровневым буферным пулом. Все разработанные алгоритмы являются сложными и ресурсоемкими. Автору неизвестна какая-либо СУБД, в которой эти алгоритмы реально бы применялись. Тем не менее, по-видимому, внедрение в состав дисковой СУБД двухуровневого кэша с использованием SSD — это наиболее дешевый способ модификации СУБД в целях повышения ее производительности за счет применения технологии SSD.

В этом случае в кэш второго уровня постепенно попадают наиболее часто используемые блоки базы данных, доступ к которым затем происходит со скоростью, свойственной SSD. Кроме того, поскольку флэш-память является энергонезависимой, не появляется необходимость в выталкивании из памяти SSD в память HHD ни в каких случаях, кроме нехватки места.

Однако необходимо заметить, что представленный выше подход не отменяет потребности в разработке чистых SSD-ориентированных СУБД, в которых особенности характеристик системы хранения данных учитываются во всех компонентах.

1.3. Гибридные устройства

Самый простой способ получить какой-то выигрыш в производительности СУБД от применения технологии SSD состоит в том, чтобы просто заменить аппаратуру HDD на аппаратуру SSD без каких-либо изменений СУБД. Как отмечалось в пп. 1.1, операции над базами данных после этого гарантированно не будут выполняться медленнее, а скорее всего будут выполняться в среднем быстрее. При наличии баз данных большого объема смена аппаратных средств хранения данных обойдется явно недешево, и смутные обещания "лучшей жизни" (на качественном уровне) вряд ли могут сподвигнуть менеджеров компаний на такие расходы.

В гибридных устройствах хранения данных на жестких дисках (*solid-state hybrid drive,* SSHD) совместно используются технологии SSD и HDD. В SSHD SSD используется для кэширования содержимого блоков HDD, к которым наиболее часто происходят обращения. В результате SSHD часто работает со скоростью SSD при стоимости, близкой к стоимости HDD. Попробовать повысить производительность СУБД за счет перехода от использования HDD к использованию SSHD представляется не столь уж затратным мероприятием. Хотя, конечно, это решение пока не опирается на какие-либо технологические доводы и остается рискованным.

2. Оперативная энергонезависимая память: перспективы для СУБД

В настоящее время реальные решения SCM могут обеспечить следующие три технологии: память на основе фазового перехода (*Phase-Change Memory*, PCRAM) [19]; резистивная память с произвольным доступом (*Resistive Random-Access Memory*, RRAM) [20]; магниторезистивная оперативная память (*Magnetoresistive Random-Access Memory*, MRAM) [21].

Память PCRAM основывается на поведении халькогенида[1], который при нагреве может "переключаться" между двумя состояниями: кристаллическим и аморфным. Кристаллическое и аморфное состояния халькогенида кардинально различаются электрическим сопротивлением. Аморфное состояние,

[1] Бинарные химические соединения халькогенов (элементов 16-й группы Периодической системы, к которым относятся кислород, сера, селен, теллур, полоний и ливерморий) с металлами.

обладающее высоким сопротивлением, используется для представления двоичного 0, а кристаллическое состояние, обладающее низким уровнем сопротивления, представляет 1.

Основная идея RRAM состоит в том, что диэлектрики, которые в нормальном состоянии имеют очень высокое сопротивление, после приложения достаточно высокого напряжения могут сформировать внутри себя проводящие нити низкого сопротивления и по сути превратиться из диэлектрика в проводник. Эти проводящие нити могут образовываться с помощью разных механизмов. С помощью приложения соответствующих уровней напряжения проводящие нити могут быть как разрушены (и материал снова станет диэлектриком), так и сформированы снова (и материал опять станет проводником).

Данные в MRAM хранятся в магнитных элементах памяти. Магнитные элементы сформированы из двух ферромагнитных слоев, разделенных тонким слоем диэлектрика. Один из слоев представляет собой постоянный магнит, намагниченный в определенном направлении, а намагниченность другого слоя изменяется под действием внешнего поля. Устройство памяти организовано по принципу сетки, состоящей из отдельных "ячеек", содержащих элемент памяти и транзистор.

Не будем останавливаться на том, какие компьютерные компании предпочитают ту или иную технологию SCM. Уже пару лет разные крупные компании обещают в ближайшем будущем начать производство соответствующих чипов. Пока на рынке появились SSD, основанные не на флэш-памяти, а на SCM с блочными обменами. Вероятно, соответствующая оперативная память появится не позже 2018 г.

Интересно, что еще в 2011 г. государственная корпорация Роснано заключила с французской компанией Crocus соглашение о налаживании в России "производства памяти MRAM средней и высокой плотности с проектными нормами 90 и 65 нм" [22]. Для справедливости следует отметить, что Samsung планирует начать массовое производство такой памяти на основе 28-нанометровой технологии [23].

Тем не менее выбор для производства в России именно памяти MRAM, по-видимому, оправдан. Причина в том, что у MRAM ожидается время чтения и записи около 20 нс (меньше, чем у сегодняшней DRAM) при долговечности, соизмеримой с долговечностью DRAM и HDD, в то время как у PCRAM и RRAM время чтения в несколько раз больше (а запись медленнее чтения) при значительно меньшей долговечности [24].

Конечно, до появления разных видов SCM на рынке невозможно достоверно сравнивать их характеристики, но есть надежда, что MRAM с обещанными характеристиками действительно появится.

Следует также обратить внимание на то, что энергонезависимая оперативная память будет использоваться в компьютерах, процессоры которых оснащены энергозависимыми кэшами. Чтобы обеспечить возможность фиксации транзакций в SCM, в систему команд процессоров Intel были добавлены две команды — CLWB и CLFLUSH [25]. Обе команды предназначены для выталкивания данных из кэшей всех уровней в SCM, однако первая команда сохраняет выталкиваемые данные в кэше, а вторая вынуждает при следующем обращении считывать данные из SCM.

2.1. SQL-ориентированные СУБД на основе SCM

На первый взгляд, в качестве основы для разработки СУБД, в которой для хранения данных используется только SCM (и совсем не используется внешняя память), было бы разумно применять какую-либо имеющуюся in-memory-СУБД. Действительно, in-memory-СУБД, как и СУБД на основе SCM, сохраняют базы данных целиком в основной памяти. В расчете на это выбираются основные структуры данных и алгоритмы выполнения операций, в расчете на это строится оптимизатор запросов. Вернее, такой оптимизатор должен был бы строиться таким образом, поскольку достоверной информации об оптимизаторах запросов в существующих in-memory-СУБД автору получить не удалось.

Однако между in-memory-СУБД и СУБД на основе SCM существует принципиальное различие, которое не позволяет так просто использовать имеющиеся решения: in-memory-СУБД рассчитаны на использование традиционной энергозависимой основной памяти, а СУБД на основе SCM — на использование энергонезависимой основной памяти. Для поддержки свойства долговечности (*durability*) транзакций в in-memory-СУБД используется внешняя память (HDD или SSD — здесь не принципиально), т. е. как и в дисковых СУБД используется двухуровневая иерархия памяти, на первом уровне которой находится энергозависимая основная память, а на втором — энергонезависимая внешняя память. В отличие от дисковых СУБД, в этом случае основная память хранит всю базу данных (а не служит кэшем), а внешняя память служит для поддержки долговечности хранения баз данных[2].

При разработке СУБД на основе SCM работа осуществляется с принципиально одноуровневой средой хранения баз данных, обладающей возможностью байтовой адресации. В этом случае, вообще говоря, можно полностью отказаться от блочной структуры памяти и начать распределять ее (для всех целей, связанных с поддержкой баз данных) порциями произвольного размера. Следует задуматься над тем, может ли это оказаться полезным, и, если да, поразмышлять о распределении основной энергонезависимой памяти фрагментами произвольного размера:

- как бороться с внешней фрагментацией?
- допустимы ли сдвижки памяти?
- не стоит ли воспользоваться какой-либо разновидностью метода близнецов (например, методом фибоначчиевых близнецов [26, 12.5 — Методы близнецов]) и т. д.

[2] Как отмечалось во введении, особый случай представляет СУБД VoltDB, но она принципиально работает в режиме *shared nothing* в массивно-параллельной среде.

По мнению автора, если отсутствует блочная структура памяти, нет причин использовать для организации индексов B-деревья[3]. Как следствие, возникает необходимость поиска ответов на перечисленные далее вопросы.

- Что можно использовать вместо B-деревьев?
- Стоит ли попытаться воспользоваться каким-либо методом поиска в основной памяти на основе деревьев (в этих методах в основном применяются двоичные деревья) [27, 4.1 — Методы поиска в основной памяти на основе деревьев]?
- Может быть, лучше применить какой-либо метод поиска на основе хэширования [27, 4.2 — Методы хэширования для поиска в основной памяти]?
- Или же лучше поискать или придумать что-нибудь новое?

Глубокого обдумывания заслуживает управление транзакциями в контексте поиска ответов на следующие вопросы.

- Как поддерживать сериализацию транзакций в транзакционных системах?
- Использовать ли версионные алгоритмы и какими они должны быть в данном случае?
- Следует ли в СУБД на основе SCM экономить на сборке мусора, потребность в которой возникает, если не ограничивать число версий объектов баз данных?
- Как вести журнал в SCM?
- Нужны ли логический и физический журналы?
- Какова единица записи в физический журнал?

Нужно думать над обработкой запросов:

- как оптимизировать запросы?
- как строить оценочные функции?

Вопросов великое множество, и на все нужно уметь правильно отвечать, чтобы получить реальные преимущества от разработки СУБД на основе SCM. К сожалению, хотя потребность в энергонезависимой основной памяти еще в 1987 г. отмечал Майкл Стоунбрейкер при разработке Postgres [28], в настоящее время проекты по полномасштабной разработке СУБД на основе SCM практически отсутствуют. Это, в частности, подтверждается тем, что на конференции SIGMOD в 2017 г. лекцию "Как построить систему управления базами данных в основной энергонезависимой памяти" [24] представляли Джой Арулрадж и Эндрю Павло из Карнего-Меллонского университета, являющиеся лидерами проекта Peloton [29].

В списке основных характеристик проекта числится изначальная поддержка технологии хранения данных на основе основной энергонезависимой памяти. К сожалению (с позиций человека, стремящегося к развитию этой технологии), как показывает название проекта, эта цель проекта не является основной. Основной целью является интеграция компонентов искусственного интеллекта для обеспечения возможности автономных (само)оптимизаций системы в зависимости от текущей рабочей нагрузки [30]. Эта задача также очень актуальна, но если учесть, что в настоящее время в проекте работают всего три взрослых специалиста (остальные — аспиранты и студенты), трудно рассчитывать, что в университетском проекте удастся полностью достичь обеих целей.

Тем не менее в настоящее время участники проекта [29], по-видимому, обладают самым большим опытом в области разработки СУБД на основе SCM. Совершенно необходимо начинать новые проекты, активно исследовать возможные подходы, проводить специальные семинары и конференции для обмена идеями и опытом.

В заключение этого подраздела можно отметить, что для автора очевидны потенциальные преимущества подхода СУБД на основе SCM для транзакционных приложений. Скорость обработки транзакций сможет сравняться со скоростью основной памяти, это принципиально новое качество. В качестве аппаратной платформы СУБД на основе SCM подходят компьютеры, процессоры которых имеют многоядерную и/или многопотоковую организацию, включают мощные графические ускорители.

К сожалению, автор не может придумать сценарий, в котором от применения SCM можно получить значительные преимущества для аналитических приложений. Считается признанной идея, что горизонтально масштабируемые аналитические СУБД нужно основывать на использовании массивно-параллельных архитектур и принципа *shared nothing* [31]. Современные аналитические базы данных настолько объемны, что только в кластере, узлы которого обладают весьма емкими средами хранения, можно полностью разместить базу данных. Даже при использовании дисковой памяти накладные расходы на пересылку данных по сети могут оказаться неприемлемыми. Если же в узлах используется SCM, то сетевые накладные расходы могут свести на нет все преимущества SCM.

2.2. SCM в объектно-ориентированных и XML-ориентированных СУБД

В XXI веке объектно-ориентированные СУБД (ООСУБД) практически потеряли пользователей. При этом активно используются разнообразные средства объектно-реляционного отображения (*Object-Relational Mapping*, ORM), позволяющие объектно-ориентированным приложениям в объектной манере взаимодействовать с SQL-ориентированными базами данных [33]. По мнению автора, для хранения объектов лучше было бы использовать ООСУБД, а не средства ORM[4].

Автору представляется, что распространенности ООСУБД во многом помешала свойственная им проблема, частично относящаяся к объектно-

[3] Вообще говоря, кажется странным использование B-деревьев в in-memory-СУБД — все-таки по своей природе это дисковая структура памяти.

[4] Как продемонстрировано в работе [32], с равным успехом можно использовать объектные возможности самого языка SQL, однако эта идея не получила широкого распространения.

ориентированной модели данных [34]. Как известно, одним из основных понятий этой модели данных является объектный идентификатор (*Object Identifier*, OID). Такой идентификатор автоматически генерируется системой при создании любого объекта, позволяя уникально отличить его от всех других объектов любого объектного типа и являясь своего рода абстрактным указателем на этот объект. В частности, с помощью объектных идентификаторов в модели ODMG образуются связи между объектами.

При использовании в ООСУБД для хранения баз данных блочной внешней памяти сложно явно использовать в качестве OID обычные указатели. Кроме того, давно известна проблема преобразования объектных идентификаторов в обычные указатели при перемещении объектов из базы данных в объектно-ориентированную среду клиентских приложений [35]. Если основывать ООСУБД на SCM, обе проблемы, похоже, значительно упростятся, а навигационная природа ООСУБД не будет сильно тормозить ее работу, поскольку затраты на разыменование можно свести практически к нулю.

Аналогично использование SCM может возродить интерес к XML-ориентированным СУБД, в которых для поддержки путевых выражений и пр. приходится поддерживать массу ссылок, а для обеспечения более или менее приемлемой эффективности использовать изощренные схемы хранения [36]. Очевидно, что при наличии 64-разрядной адресации и достаточного объема основной энергонезависимой памяти XML-ориентированные СУБД можно резко упростить и ускорить.

Заключение

Как видно из представленных выше соображений, сценариев, в которых SCM может значительно повысить эффективность СУБД и упростить их организацию, более чем достаточно. Нужно продолжать анализировать разные ветви дисциплины управления данными, чтобы не упустить других благоприятных возможностей применения SCM. Автору было бы очень интересно найти пути использования SCM в аналитических СУБД. И конечно, требуется большое число исследовательских проектов, чтобы найти правильные пути разработки СУБД на основе SCM.

Данная статья является незначительно измененным вариантом материалов [37], представленных на пятую международную конференцию "Актуальные проблемы системной и программной инженерии" в 2017 г.

Список литературы

1. **Bayer R., McCreight E.** Organization and Maintenance of Large Ordered Indexes // Acta Informatica, 1972. Vol. 1, Issue 3. P. 173—189.
2. **Hellerstein J. M., Stonebraker M.** Anatomy of a Database System // Readings in Database Systems. 4th Edition. MIT Press, 2005. P. 42—95.
3. **Кузнецов С. Д.** Базы данных. Академия. Серия: Университетский учебник, 2012. 496 с.
4. **Selinger P. G., Astrahan M. M., Chamberlin D. D.** et al. Access Path Selection in a Relational Database Management System // In Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data. 1979. P. 23—34.
5. **DeWitt D. J., Hawthorn P. B.** A Performance Evaluation of Data Base Machine Architectures (Invited Paper) // In Proceedings of the 7th International Conference on Very Large Data Bases, 1981. P. 199—214.
6. **DeWitt D. J., Gray J.,** Parallel database systems: the future of high performance database systems // Communications of the ACM. 1992. Vol. 35, Issue 6. P. 85—98.
7. **DeWitt D. J., Katz R. H., Olken F.** et al. Implementation techniques for main memory database systems // In Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, 1984. P. 1—8.
8. **Шапоренков Д. А.** Эффективные методы индексирования данных и выполнения запросов в системах управления базами данных в основной памяти: дис. ... канд. физ.-мат. наук. Санкт-Петербургский государственный университет, 2006.
9. **Lahiri T., Neimat M.-A., Folkman S.** Oracle TimesTen: An In-Memory Database for Enterprise Applications // Bulletin of the Technical Committee on Data Engineering. 2013. Vol. 36, No. 2. P. 6—13.
10. **Lindstrem J., Raatikka V., Ruuth J.** et al. IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability // Bulletin of the Technical Committee on Data Engineering. 2013. Vol. 36, No. 2. P. 14—20.
11. **Stonebraker M., Weisberg A.** The VoltDB Main Memory DBMS // Bulletin of the Technical Committee on Data Engineering. 2013. Vol. 36, No. 2. P. 21—27.
12. **Кузнецов С. Д.** Транзакционные параллельные СУБД: новая волна // Труды ИСП РАН. 2011. Т. 20. С. 189—251.
13. **Oracle** TimesTen In-Memory Database High Availability. Using TimesTen Replication to deliver High Availability and Disaster Recovery. Oracle White Paper, October 2015. URL: http://www.oracle.com/technetwork/database/database-technologies/timesten/overview/wp-timesten-ha-2735640.pdf.
14. **Salkosuo S.** Introducing IBM solidDB High Availability and Transparent Connectivity. IBM developerWorks, March 25, 2010. URL: https://www.ibm.com/developerworks/data/library/techarticle/dm-1003soliddbha/dm-1003soliddbha-pdf.pdf.
15. **Novotný R., Kadlec J., Kuchta R.** NAND Flash Memory Organization and Operations // Journal of Information Technology & Software Engineering. 2015. Vol. 5, Issue 1. P. 8.
16. **Проект FlashyDB**, Data Management Lab, Reutlingen University, Germany. URL: http://dblab.reutlingen-university.de/FDB.html.
17. **Petrov I., Gottstein R., Hardock S.** DBMS on modern storage hardware // In Proceedings of the 31st International Conference on Data Engineering (ICDE), 2015. P. 1545—1548.
18. **Кузнецов С. Д., Прохоров А. А.** Алгоритмы управления буферным пулом СУБД при работе с флэш-накопителями // Труды ИСП РАН. 2012. Т. 23. С. 173—194. DOI: 10.15514/ISPRAS-2012-23-11.
19. **Raoux S., Burr G. W., Breitwisch M. J.** et al. Phase-change random access memory: A scalable technology // Journal of Research and Development. 2008. Vol. 52, No. 4/5. P. 465—479.
20. **Strukov D. B., Snider G. S., Stewart D. R., Williams R. S.** The missing memristor found // Nature. 2008. Vol. 453. P. 80—83.
21. **Chi P., Li S., Cheng Yu., Lu Yu.** et al. Architecture Design with STT-RAM: Opportunities and Challenges // In Proc. of the 21st Asia and South Pacific Design Automation Conference, 2016. P. 109—114.
22. **MRAM:** Создание производства магниторезистивной оперативной памяти в России. URL: http://www.rusnano.com/projects/portfolio/crocus-technology.
23. **Lin Yi., Shen J.** DIGITIMES [Tuesday 26 September 2017]. Samsung ready to mass produce MRAM chips using 28nm FD-SOI process. URL: https://digitimes.com/news/a20170925PD206.html
24. **Arulraj J., Pavlo A.** How to Build a Non-Volatile Memory Database Management System // In Proceedings of the 2017 ACM International Conference on Management of Data, 2017. P. 1753—1758.

25. **Intel 64** and IA-32 Architectures. Software Developer's Manual. Documentation Changes. July 2017. URL: https://software.intel.com/sites/default/files/managed/3e/79/252046-sdm-change-document.pdf
26. **Ахо А., Хопкрофт Д., Ульман Д.** Структуры данных и алгоритмы. Вильямс, 2016, 400 с.
27. **Кузнецов С. Д.** Методы сортировки и поиска. URL: http://citforum.ru/programming/theory/sorting/sorting2.shtml.
28. **Stonebraker M.** The Design of the POSTGRES Storage System // In Proceedings of 13th International Conference on Very Large Data Bases. 1987. P. 289—300.
29. **Проект Peloton:** The Self-Driving Database Management System. Database Group, Carnegie Mellon University. URL: http://pelotondb.io/
30. **Pavlo A., Angulo G. Arulraj J.,** et al. Self-Driving Database Management Systems // In Proceedings of the 8th Biennial Conference on Innovative Data Systems Research (CIDR'17), Online Proceedings, 6 p.
31. **Кузнецов С. Д.** К свободе от проблемы больших данных // Открытые системы. 2012. № 2. С. 22—24.
32. **Neward T.** The Vietnam of Computer Science. Ted Neward's Blog, Jun 26, 2006. URL: http://blogs.tedneward.com/post/the-vietnam-of-computer-science/
33. **Кузнецов С. Д.** Объектные модели ODMG и SQL десять лет спустя: нет противоречий // Труды ИСП РАН. 2015. Т. 27, № 1. С. 173—192. DOI: 10.15514/ISPRAS-2015-27(1)-9.
34. **The Object** Data Standard: ODMG 3.0/ Eds by R. G. G. Cattel, D. K. Barry. Morgan Kauffmann Publishers, 2000. 280 p.
35. **Kemper A., Kossmann D.** Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis // The VLDB Journal. 1995. Vol. 4, Issue 3. P. 519—566.
36. **Taranov I., Shcheklein I., Kalinin A.** et al. Sedna: Native XML Database Management System (Internals Overview) // In Proceedings of the 2010 International Conference on Management of Data, 2010. P. 1037—1046.
37. **Кузнецов С. Д.** Перспективы и проблемы использования энергонезависимой памяти // CEUR Workshop Proceedings. 2017. Vol. 1989. P. 7—21.

New Storage Devices and their Impact on Database Technology

S. D. Kuznetsov, kuzloc@ispras.ru, Ivannikov Institute for System Programming of the RAS, Moscow, 109004, Russian Federation, Lomonosov Moscow State University, Moscow, 119991, Russian Federation, Moscow Institute of Physics and Technology, Moscow Region, 141700, Russian Federation, Higher School of Economics, Moscow, 101000, Russian Federation

Corresponding author:

Kuznetsov Sergey D., Senior Researcher, Ivannikov Institute for System Programming of the RAS, Moscow, 109004, Russian Federation, Lomonosov Moscow State University, Moscow, 119991, Russian Federation, Moscow Institute of Physics and Technology, Moscow Region, 141700, Russian Federation, Higher School of Economics, Moscow, 101000, Russian Federation,
E-mail: kuzloc@ispras.ru

Received on December 12, 2017
Accepted on January 10, 2018

At the beginning of the paper, it is demonstrated that the technology of the most widely used SQL-oriented DBMS is inextricably linked with HDD technology. Features of HDD affect the data structures and algorithms for performing operations, methods of managing the buffer pool of the DBMS, transaction management, query optimization, etc. An alternative to a disk DBMS is an in-memory DBMS, storing databases entirely in the main memory. Despite the fact that in-memory DBMS has a number of advantages over disk DBMS, at present there is practically no competition. This, first of all, is due to natural limitations on the size of databases, inherent in in-memory DBMS. At present, new types of data storage hardware have appeared: SSD — block solid-state drives and SCM — storage-class memory (non-volatile main memory). SSD characteristics made it expedient to develop a DBMS in terms of their exclusive use, so far no such DBMS has been created, and SSDs are used simply instead of HDDs in DBMS that do not take their features into account. The availability of SCM allows one to radically simplify the architecture of the database and significantly improve their performance. To do this, you need to review many of the ideas used in disk-based databases.

Keywords: *SQL-based DBMS; Hard Disk Drive with movable Heads; cost-based query optimization; main-memory DBMS; Flash-based Solid State Drive; non-volatile main memory; storage-class memory*

For citation:

Kuznetsov S. D. New Storage Devices and their Impact on Database Technology, *Programmnaya Ingeneria,* 2018, vol. 9, no. 4, pp. 147—155.

DOI: 10.17587/prin.9.147-155

References

1. **Bayer R., McCreight E.** Organization and Maintenance of Large Ordered Indexes, *Acta Informatica*, 1972, vol. 1, issue 3, pp. 173—189.

2. **Hellerstein J. M., Stonebraker M.** Anatomy of a Database System, *Readings in Database Systems*, 4th Edition, MIT Press, 2005, pp. 42—95.

3. **Kuznetsov S. D.** *Bazy dannyh* (Databases), Academia, series: University textbook, 2012, 496 p. (in Russian).

4. **Selinger P. G., Astrahan M. M., Chamberlin D. D., Lorie R. A., Price T. G.** Access Path Selection in a Relational Database Management System, *In Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, 1979, pp. 23—34.

5. **DeWitt D. J., Hawthorn P. B.** A Performance Evaluation of Data Base Machine Architectures (Invited Paper), *In Proceedings of the 7th International, Conference on Very Large Data Bases*, 1981, pp. 199—214.

6. **DeWitt D., Gray J.** Parallel database systems: the future of high performance database systems, *Communications of the ACM*, 1992, vol. 35, issue 6, pp. 85—98.

7. **DeWitt, D. J., Katz R. H., Olken F., Shapiro L. D, Stonebraker M. R., Wood D. A.** Implementation techniques for main memory database systems, *in Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, 1984, pp. 1—8.

8. **Shaporenkov D. A.** Jeffektivnye metody indeksirovanija dannyh i vypolnenija zaprosov v sistemah upravlenija bazami dannyh v osnovnoj pamjati (Efficient methods of data indexing and query execution in in-memory database management systems). PhD Thesis. Saint-Petersburg State University, 2006 (in Russian).

9. **Lahiri T., Neimat M.-A., Folkman S.** Oracle TimesTen: An In-Memory Database for Enterprise Applications, *Bulletin of the Technical Committee on Data Engineering,* 2013, vol. 36, no. 2, pp. 6—13.

10. **Lindstrom J., Raatikka V., Ruuth J., Soini P., Vakkila K.** IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability, *Bulletin of the Technical Committee on Data Engineering*, 2013, vol. 36, no. 2, pp. 14—20.

11. **Stonebraker M., Weisberg A.** The VoltDB Main Memory DBMS, *Bulletin of the Technical Committee on Data Engineering*, 2013, vol. 36, no. 2, pp. 21—27.

12. **Kuznetsov S. D.** Tranzakcionnye parallel'nye SUBD: novaja volna (Transactional Massive-Parallel DBMSs: A New Wave), *Trudy ISP RAN,* 2011, vol. 20, pp. 189—251 (in Russian).

13. **Oracle** TimesTen In-Memory Database High Availability. Using TimesTen Replication to deliver High Availability and Disaster Recovery. Oracle White Paper, October 2015, available at: http://www.oracle.com/technetwork/database/database-technologies/timesten/overview/wp-timesten-ha-2735640.pdf.

14. **Salkosuo S.** Introducing IBM solidDB High Availability and Transparent Connectivity. IBM developerWorks, March 25, 2010, available at: https://www.ibm.com/developerworks/data/library/techarticle/dm-1003soliddbha/dm-1003soliddbha-pdf.pdf.

15. **Novotny R., Kadlec J., Kuchta R.** NAND Flash Memory Organization and Operations, *Journal of Information Technology & Software Engineering,* 2015, vol. 5, issue 1, p. 8.

16. **FlashyDB** project website, Data Management Lab, Reutlingen University, Germany, available at: http://dblab.reutlingen-university.de/FDB.html

17. **Petrov I., Gottstein R., Hardock S.** DBMS on modern storage hardware, *In Proceedings of the 31st International Conference on Data Engineering (ICDE)*, 2015, pp. 1545—1548.

18. **Kuznetsov S. D., Prokhorov A. A.** Algoritmy upravlenija bufernym pulom SUBD pri rabote s fljesh-nakopiteljami (Flash-based algorithms of database buffer management), *Trudy ISP RAN*, 2012, vol. 23, pp. 173—194. DOI: 10.15514/ISPRAS-2012-23-11 (in Russian).

19. **Raoux S., Burr G. W., Breitwisch M. J., Rettner C. T., Chen Y.-C., Shelby R. M., Salinga M., Krebs D., Chen S.-H., Lung H.-L., Lam C. H.** Phase-change random access memory: A scalable technology, *Journal of Research and Development*, 2008, vol. 52, no. 4/5, pp. 465—479.

20. **Strukov D. B., Snider G. S., Stewart D. R., Williams R. S.** The missing memristor found, *Nature,* 2008, vol. 453, pp. 80—83.

21. **Chi P., Li S., Cheng Yu., Lu Yu, Kang S. H., Xie Yu.** Architecture Design with STT-RAM: Opportunities and Challenges, *In Proc. of the 21st Asia and South Pacific Design Automation Conference*, 2016, pp. 109—114.

22. **MRAM:** Sozdanie proizvodstva magnitorezistivnoj operativnoj pamjati v Rossii (MRAM: Organization of manufacturing of magnetoresistive RAM in Russia), available at: http://www.rusnano.com/projects/portfolio/crocus-technology (in Russian).

23. **Lin Yi., Shen J.** DIGITIMES [Tuesday 26 September 2017]. Samsung ready to mass produce MRAM chips using 28nm FD-SOI process, available at: https://digitimes.com/news/a20170925PD206.html

24. **Arulraj J, Pavlo A.** How to Build a Non-Volatile Memory Database Management System, *In Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1753—1758.

25. **Intel 64 and IA-32** Architectures. Software Developer's Manual. Documentation Changes. July 2017, available at: https://software.intel.com/sites/default/files/managed/3e/79/252046-sdm-change-document.pdf

26. **Aho A., Ullman J., Hopcroft J.** *Data Structures and Algorithms,* Pearson, 1983, 427 p.

27. **Kuznetsov S. D.** Metody sortirovki i poiska (Sort and search methods), 2003, available at: http://citforum.ru/programming/theory/sorting/sorting2.shtml (in Russian).

28. **Stonebraker M.** The Design of the POSTGRES Storage System, *In Proceedings of 13th International Conference on Very Large Data Bases*, 1987, pp. 289—300.

29. **Peloton** project website: The Self-Driving Database Management System, Database Group, Carnegie Mellon University, available at: http://pelotondb.io/

30. **Pavlo A., Angulo G., Arulraj J., Lin H., Lin J., Ma L. Menon P., Mowry T. C., Perron M., Quah I., Santurkar S., Tomasic A., Toor S., Van Aken D., Wang Z., Wu Yi., Xian R, Zhang T.** Self-Driving Database Management Systems, *In Proceedings of the 8th Biennial Conference on Innovative Data Systems Research (CIDR'17*), Online Proceedings, 6 p.

31. **Kuznetsov S. D.** K svobode ot problemy bol'shih dannyh (To the freedom of the big data problem), *Otkrytye sistemy*, 2012, no. 2, pp. 22—24 (in Russian).

32. **Neward T.** The Vietnam of Computer Science. Ted Neward's Blog, Jun 26, 2006, available at: http://blogs.tedneward.com/post/the-vietnam-of-computer-science/

33. **Kuznetsov S. D.** Ob'ektnye modeli ODMG i SQL desjat' let spustja: net protivorechij (ODMG and SQL object models ten years later: there are no contradictions), *Trudy ISP RAN*, 2015, vol. 27, issue 1, pp. 173—192. DOI: 10.15514/ISPRAS-2015-27(1)-9 (in Russian).

34. **The Object** *Data Standard:* ODMG 3.0/ Eds by R. G. G. Cattel, D. K. Barry, Morgan Kauffmann Publishers, 2000, 280 p.

35. **Kemper A., Kossmann D.** Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis, *The VLDB Journal,* 1995, vol. 4, issue 3, pp. 519—566.

36. **Taranov I., Shcheklein I., Kalinin A., Novak L., Kuznetsov S., Pastukhov R., Boldakov A., Turdakov D., Antipin K., Fomichev A., Pleshachkov P., Velikhov P., Zavaritski N., Grinev M., Grineva M., Lizorkin D.** Sedna: Native XML Database Management System (Internals Overview), *In Proceedings of the 2010 International Conference on Management of Data*, 2010, pp. 1037—1046.

37. **Kuznetsov S. D.** Perspektivy i problemy ispol'zovanija jenergonezavisimoj pamjati (Prospects and problems of using nonvolatile memory), *CEUR Workshop Proceedings,* 2017, vol. 1989, pp. 7—21 (in Russian).

УДК 004.27 DOI: 10.17587/prin.9.156-162

М. В. Баклановский, ст. преподаватель**,** e-mail: m.baklanovsky@spbu.ru,
Б. Н. Кривошеин, ст. преподаватель, e-mail: bnkv@outlook.com,
А. Н. Терехов, д-р физ.-мат. наук, проф., зав. кафедрой, e-mail: a.terekhov@spbu.ru,
М. А. Терехов, студент, e-mail**:** st054464@student.spbu.ru, Санкт-Петербургский государственный университет, г. Санкт-Петербург, **А. Е. Сибиряков,** ст. преподаватель, e-mail: a.sibiryakov@urfu.ru, Уральский федеральный университет имени первого Президента России Б. Н. Ельцина, г. Екатеринбург

Асимметричный маркерный процессинг

Предложен подход к созданию гетерогенных вычислительных систем с возможностями организации единой вычислительной среды с унифицированным механизмом передачи управления, сквозного маршрута проектирования, автоматической генерации части элементов системы в процессе компиляции и с отладкой на уровне системы. Подход может применяться для элементов аппаратной защиты от обратного анализа, для создания высокопроизводительных систем со специализированными вычислительными ядрами и частичным параллелизмом и для повышения отказоустойчивости.

Ключевые слова: *процессор, передача управления, стек протоколов, надежность, производительность, защищенность, отказоустойчивость*

Для решения одной вычислительной задачи часто используется несколько процессоров. В тех случаях, когда процессоры однотипные и работают над решением задачи одновременно, говорят о параллельных вычислениях. Часто встречается и другая ситуация — процессор, который начал вычислительный процесс, на некоторое время останавливает свое участие в решении задачи, а в это время работает другой процессор, который имеет свою память (общей для этих двух процессоров области памяти может и не быть). Второй процессор при этом называют сопроцессором, а используется он обычно для быстрого выполнения некоторого фрагмента вычислений или для решения части задачи, с решением которой первый процессор справиться не может. Кроме этих двух причин использования сопроцессоров, есть еще одна — часть вычислений можно скрыть, перенеся их в сопроцессор с прошитым внутри него микрокодом. Это важно, если одновременно с решением задачи требуется скрыть алгоритм ее решения или некоторую часть этого алгоритма. Сопроцессоров, работающих над решением задачи вместе с первым, или, как его часто называют, центральным процессором, может быть несколько. Такая ситуация часто возникает на практике при разработке программно-аппаратных вычислительных систем, и именно для построения таких систем предлагается представленное в настоящей работе унифицированное решение — асимметричный маркерный процессинг (АМП).

- Асимметричный означает, что как сами процессоры, так и их роли в общем вычислительном процессе отличаются друг от друга. При этом в операциях АМП все процессоры равноправны.
- Маркерный означает строгую очередность участия разных процессоров в процессе вычислений — в любой момент времени работать над решением задачи может только один из процессоров. Право решать задачу дает маркер, и этот маркер у задачи только один. По желанию программиста маркер может быть передан от одного процессора другому, и новый владелец маркера приступит к решению своего фрагмента задачи, а процессор, отдавший ему маркер, войдет в состояние (приступит к выполнению программы) ожидания маркера. Изначально маркер принадлежит процессору, который по воле программиста первым начал работу над решением задачи.
- Процессинг — это то, что делают процессоры. Это слово можно заменить на синоним "вычисления", и тогда термин будет выглядеть так: асимметричные маркерные вычисления.

Стек протоколов и передача маркера

Для организации коммуникаций между процессорами (как минимум это передача маркера) организуются протоколы на следующих пяти уровнях стека ISO/OSI [1]:

- прикладной;
- уровень представления;
- транспортный уровень с двумя модулями — модуль передачи маркера и отладочный модуль;
- сетевой;
- физический.

На прикладном уровне выполняется решение вычислительной задачи. Когда программист хочет дать поработать другому процессору системы, он с помощью соответствующего модуля организует передачу маркера (ПМ). Для этого на прикладном уровне формируется структура ПМ из четырех полей:

- номер (внутрисистемный) процессора, которому передается маркер;
- размер пакета (изначально не 0, меняется в процессе обработки);
- пакет с номером точки входа (формируется при подготовке исполняемого кода на все процессоры) и, возможно, некоторыми дополнительными данными, например, со значениями регистров, которые другому процессору не видны, но значения которых при вычислениях ему понадобятся;
- результат выполнения транзакции.

На уровне представления пакет может быть закодирован одним из предопределенных при разработке системы способов. После закодирования к пакету добавляется заголовок уровня представления с указанием номера алгоритма кодирования, а значение 0 в этом поле резервируется для случая отсутствия кодирования. Процессор и работающая на нем программа, которые получат данный пакет, должны уметь раскодировать его. После обработки на уровне представления пакет передается на транспортный уровень модулю ПМ.

На транспортном уровне модуль ПМ дает отладочному модулю команду "Остановить отладку", после чего отладочный модуль останавливает формирование любых отладочных сообщений, т. е. останавливаются все отладочные диалоги (ОД). При этом номера процессоров, с которыми не были завершены ОД, запоминаются в списке ОД. После завершения ПМ всем процессорам, номера которых есть в списке ОД, посылается особое отладочное сообщение о том, что отладку прервала ПМ. Транзакция ПМ имеет глобальный номер (НТ), передающийся в каждом пакете, входящим в транзакцию. НТ (изначально 0) инкрементируется перед началом каждой транзакции. Для выполнения транзакции ПМ используются три типа пакетов:

- пакет 1 с маркерным признаком;
- пакет 2 с подтверждением получения маркера;
- пакет 3 о получении подтверждения.

Если во время транзакции участвующему в ней процессору приходят отладочные пакеты, то номера отправителей этих пакетов вносятся в список ОД этого процессора.

Транзакция начинается с передачи пакета 1. Если после передачи пакета 1 прошло заранее заданное время (таймаут), а пакет 2 так и не пришел, то передача пакета 1 повторяется. Принимающий маркер процессор обязан ответить пакетом 2 на все пакеты 1, которые он получил. Более того, он еще раз посылает пакет 2, если таймаут ожидания пакета 3 истек, а пакет 3 все еще не пришел. Передающий маркер процессор обязан ответить пакетом 3 на все полученные им пакеты 2, хотя его часть транзакции считается успешно завершенной после отправки первого из них, а принимающий процессор не может передать маркер до того, как получит хотя бы один пакет 3, после чего считается успешно завершенной и его часть транзакции. После успешного завершения транзакции в структуре ПМ в поле "результат выполнения транзакции" заносится значение 0.

Есть еще один тип пакетов, которые могут использоваться в сочетаниях с пакетами первых трех типов: пакет 4 — аварийное завершение транзакции. Такой пакет может быть ответом на пакет 1 и на пакет 2. Если в транзакции хотя бы один раз был отправлен пакет 4, то эта транзакция уже не может быть завершена успешно. В ответ на все остальные пакеты транзакции с этим номером должны отправляться пакеты 4. В поле данных пакетов типа 4 может указываться причина аварийного завершения. После аварийного завершения транзакции в структуре ПМ в поле "результат выполнения транзакции" заносится значение 1. Кроме того, пакет 4 является ответом на пакеты 2, отправленные без предварительного получения пакета 1, и на пакеты 3, отправленные без предварительного получения пакета 2. Такие ситуации могут возникать, например, в результате повреждения пакетов при передаче.

На сетевом уровне должна быть доступна информация о том, какой модуль транспортного уровня обрабатывал пакет до него. К пакету добавляется заголовок, содержащий номер процессора, которому передается маркер, номер процессора, который передает маркер, признак ПМ и контрольную сумму всего пакета вместе с номерами процессоров. После этого на основании номера процессора, которому передается маркер, выбирается способ доставки и вызывается соответствующая процедура физического уровня.

На физическом уровне пакет переносится из внутреннего буфера стека передающего процессора в буфер пакетов принимающего процессора с указанием длины. Каждый принятый таким образом пакет передается на сетевой уровень путем вызова соответствующего программного модуля.

На сетевом уровне с каждым принятым пакетом выполняются перечисленные далее действия.

1. По контрольной сумме проверяется, не был ли пакет поврежден в процессе доставки. Если установлен факт повреждения пакета, то он удаляется из буфера принятых пакетов.

2. Проверяется, какому процессору адресован пакет. Если в качестве номера получателя установлено значение 0 (широковещательный пакет, который не может использоваться при ПМ), то пакет передается на обработку в отладочный модуль. Если номер получателя совпадает с номером процессора, на который пришел пакет, то пакет принимается и передается на обработку; если в заголовке пакета установлен признак ПМ, то пакет передается модулю ПМ, если нет, то отладочному модулю. Во всех остальных случаях пакет удаляется из буфера принятых пакетов.

Отладочные диалоги

Отладка выполняется на стадии разработки системы, и на этапе эксплуатации она может быть полностью отключена. Однако некоторые ОД могут быть сохранены и использоваться в целях автоматического мониторинга состояния системы и самодиагностики.

Все ОД выполняются без гарантии доставки пакетов. Если на отладочный запрос не приходит ответ в течение некоторого таймаута, то запрос можно повторить. Для отключения отладки можно заменить отладочные модули на программные заглушки (программы, состоящие из команды возврата).

Отладочный модуль (если он не заменен на заглушку) должен поддерживать описанные далее типы ОД.

1. **Поиск маркера.** Запрос широковещательный, отвечает только владелец маркера, в качестве отладочной информации присылается НТ и номер точки входа.

2. **Остановка системы.** Запрос широковещательный, отвечает только владелец маркера, в качестве отладочной информации присылается НТ, номер точки входа и точка возобновления вычислений.

3. **Возобновление работы системы.** Запрос широковещательный или юникастовый, отвечает только владелец маркера, ответ без отладочной информации.

4. **Пинг.** Запрос юникастовый, ответ без отладочной информации.

5. **Отключение процессора от сети.** Запрос юникастовый, ответ без отладочной информации. Отключенный процессор продолжает принимать пакеты, но обрабатывает только отладочные пакеты типа 6.

6. **Восстановление связи процессора с сетью.** Запрос юникастовый, ответ без отладочной информации.

7. **Передача маркера.** Запрос юникастовый, ответа нет. Пакет данного типа с точки зрения транспортного уровня, уровня представления и прикладного уровня совпадает с пакетом 1 транзакции ПМ. Получив его, отладочный модуль создает структуру ПМ (путем расширения аналогичной структуры, создаваемой на сетевом уровне для обработки отладочных пакетов) и передает пакет модулю ПМ. После этого выполняется обычная последовательность действий по получению маркера.

8. **Перезапуск системы (перезагрузка).** Запрос широковещательный, ответа нет.

Форматы пакетов

Пакеты ПМ имеют следующую последовательность заголовков:

- сетевой уровень;
- модуль ПМ на транспортном уровне;
- уровень представления.

После заголовков располагается непустая область данных.

Отладочные пакеты имеют следующую последовательность заголовков:

- сетевой уровень;
- отладочный модуль на транспортном уровне.

После заголовков располагается необязательная область данных.

Структура заголовка сетевого уровня:

- поле "Кому" (1 байт, особое значение 0);
- поле "От кого" (1 байт);
- поле "Модуль транспортного уровня" (1 байт, 1 — модуль ПМ, 0 — отладочный модуль);
- поле "Контрольная сумма" (3 байта, например, CRC-24).

Структура заголовка модуля ПМ (транспортный уровень):

- НТ (4 байта).

Структура заголовка уровня представления:

- номер алгоритма кодирования (2 байта, особое значение 0).

Структура заголовка отладочного модуля:

- тип ОД (1 байт).

Реализация физического уровня на общей памяти

В общей памяти для каждого из процессоров, имеющих к ней доступ, организуется кольцевой буфер (с указателями на начало и на конец заполненной части). Это буферы принятых пакетов. Каждый пакет в буфере принятых пакетов хранится так: сначала длина пакета (2 байта), затем сам пакет.

Юникастовый пакет доставляется в результате работы двух программ. На физическом уровне передающего процессора программа забирает пакет из внутреннего буфера стека и помещает его в буфер принятых пакетов того процессора, которому адресован пакет. Затем программа на физическом уровне получающего процессора, постоянно проверяющая состояние своего буфера принятых пакетов, обнаруживает в нем новый пакет и начинает его обработку в своем стеке.

Если пакет оказывается больше, чем свободная часть буфера принимающих пакетов, то пакет удаляется из внутренней памяти стека отправляющего процессора (как отправленный), и нигде не остается его копии.

Широковещательный пакет переносится в буферы принятых пакетов всех процессоров системы. При этом на физическом уровне передающего процессора поочередно вызываются одна или несколько программ доставки пакетов по одному разу для каждого процессора системы, кроме передающего. Из внутренней памяти стека передающего процессора пакет удаляется только после выполнения последней из процедур доставки. При этом в некоторые из буферов принятых пакетов передаваемый пакет может и не попасть.

Прикладной уровень

На прикладном уровне реализуется функция передачи управления (ФПУ) в пространства инструкций других процессоров. По аналогии с передачей

управления в своем пространстве процессор может выполнить перечисленные далее действия.

- Передать управление на некоторую точку входа в пространстве другого процессора.
- Выполнить вызов процедуры в пространстве другого процессора, т. е. запомнить в специальном стеке адрес инструкции, следующей за вызовом ФПУ, и выполнить ПМ другому процессору с указанием номера точки входа, соответствующей адресу вызываемой процедуры в пространстве этого другого процессора.
- Вернуться из процедуры в пространство вызывавшего процессора, т. е. передать маркер вызывавшему процессору, установив при этом в поле "номер точки входа" значение 0. Вызывавший процессор получит маркер, достанет из своего специального стека адрес в своем пространстве и передаст по нему управление.

Заметим, что при выполнении ФПУ возможны ошибки следующих типов:

- ПМ выполнена, но переданный вместе с маркером номер точки входа процессору, получившему маркер, неизвестен;
- выполнен возврат из процедуры, но специальный стек процессора, получившего маркер, пуст.

Аналогичные перечисленным выше ошибки возможны и при обычных передачах управления в пространстве одного процессора. Как правило, такое бывает вследствие ошибок при подготовке исполняемого кода, исправить которые во время выполнения чаще всего невозможно. Однако ФПУ может их обнаруживать и записывать в отладочный журнал, останавливать работу системы или выполнять перезагрузку.

Возможности по генерации прошивок программируемых логических интегральных схем во время компиляции

Для генерации прошивок программируемых логических интегральных схем (ПЛИС) авторы используют язык собственной разработки HaSCoL (*Hardware and Software CoDesign Language*) [2], программы на котором в 5—6 раз короче, чем на традиционном VHDL [3]. Сгенерированные на HaSCoL прошивки легче и читать, и писать, в том числе и в автоматическом режиме. На этом языке сделана реализация всего стека АМП и ФПУ. На основе компилятора с открытыми исходными текстами (например, Clang) делается такой модифицированный компилятор, который из исходной программы создает два файла: исполняемый файл для основного процессора (обычно это ARM, и для него тоже сделана реализация стека АМП и ФПУ) и программу на языке HaSCoL. Из программы на языке HaSCoL с помощью специальных утилит автоматически синтезируется прошивка для ПЛИС и выполняется операция переноса этой прошивки на ПЛИС. При создании двух выходных файлов модифицированный компилятор выделяет некоторую часть кода, убирает ее из исполняемого файла, а на ее место в исполняемом файле вставляет вызов ФПУ в ПЛИС. Для этой же части кода генерируется программа на языке HaSCoL, которая выполняет все те же действия с данными, которые выполнила бы эта часть кода при исполнении на основном процессоре. При этом возникают трудности — ПЛИС не видит регистры и оперативную память основного процессора. Для устранения этой трудности используется ФПУ из ПЛИС в основной процессор, после чего в ПЛИС передаются необходимые данные или, наоборот, модифицируются регистры и оперативная память с использованием данных, которые передал ПЛИС при выполнении ФПУ.

Все номера процессоров, точки входа и их номера, способы вызова и возврата управления — это данные, которые порождает и использует в своей работе компилятор. Если он нигде не ошибся, то и во время выполнения ошибок с передачей управления не будет. Для ручного контроля компилятор сохраняет ассемблерные листинги частей кода, перенесенных в ПЛИС (для сопоставления с соответствующими программами на языке HaSCoL), и всю схему передач управления через АМП. Кроме этого, в исполняемый код компилятор может вставить несколько отладочных команд, которые во время выполнения сформируют всю трассу передач управления между основным процессором и ПЛИС со всеми необходимыми для отладки параметрами.

Затраты времени на передачу маркера и выполнение участков кода

Измерение времени, которое необходимо для передачи маркера, было проведено на платформе Zybo Zynq-7000 (вычислители — ARM Cortex-A9 и ПЛИС). Выбор платформы обусловлен исключительно ее доступностью и невысоким по сравнению с мощными системами на кристалле (System on a Chip, SoC) быстродействием.

Для передачи данных было выбрано две области памяти — одна в памяти, расположенной на кристалле, и вторая в оперативном запоминающем устройстве (ОЗУ, DDR3). Для сравнения было проведено измерение времени записи тех же самых пакетов с данными в собственную виртуальную память программы (и оно оказалось гораздо меньшим). Однако сравнение между собой времени работы с виртуальной и физической памятью представляется не совсем корректным. Причина в том, что при работе с виртуальной памятью используется кэш и чаще всего запись информации на физический носитель (в ОЗУ) не происходит или происходит вне измеряемого промежутка времени, в то время как при обмене данными с внешними устройствами через общую физическую память такая запись происходит каждый раз, причем с использованием какого-либо интерфейса передачи данных.

Помимо описанных выше областей памяти для хранения общих переменных и флагов (готовность

устройств и пакетов, адреса пакетов в кольцевых буферах, текущие показания таймера ПЛИС) использовалась область памяти, расположенная непосредственно в ПЛИС и отображенная в физическое адресное пространство SoC.

Измерения проводили в среде ОС Linux. Отображение физической памяти в виртуальное адресное пространство задачи со стороны ARM проводили с использованием возможностей системного вызова mmap в ОС Linux.

Программное обеспечение (ПО) для измерений изначально было написано на языках ассемблера и Си. Анализ скорости работы ПО показал, что версия на Си при массовой передаче данных уступает в скорости версии на ассемблере в 1,5—3 раза за счет того, что кросс-компилятор arm-linux-gnueabihf-gcc версии 4.8.3 от Linaro генерирует неоптимальный по скорости исполнения код. Дело в том, что данный кросс-компилятор при передаче пакетов использует большое число инструкций ldr/str вместо ldmia/stmia, сохраняет локальные переменные и промежуточные значения в стеке, генерируется большое число ненужных машинных команд даже при использовании опции максимальной оптимизации кода-Ofast. С учетом этого обстоятельства все ключевые функции использовались только в ассемблерной версии.

Измерение времени проводили с использованием возможностей глобального таймера (*Global Timer*, GT). Для дополнительного контроля корректности проводимых измерений использовали независимый таймер, работающий на ПЛИС. Расположенные на плате таймеры TTC (*Triple Timer Counters*) при измерениях не использовали по причине невысокой разрядности поддерживаемых ими счетчиков.

Результаты замеров вне зависимости от частоты используемых таймеров приводились к наносекундам. Учитывая, что чтение показаний с таймера также занимает некоторое время, каждый раз до начала измерений проводился контрольный замер времени исполнения замера, результат которого в дальнейшем вычитали из общего времени выполнения операции.

За один цикл измерения проводилась передача нескольких (от 1 до 1024) маркеров. Длина пакетов с данными составляла 92 байта.

Общий алгоритм измерения:

- выполнение контрольного замера;
- в цикле — формирование очередного пакета с данными, передача маркера, прием маркера;
- вычисление времени выполнения цикла с учетом контрольного замера;
- вычисление среднего времени на одну передачу маркера.

На каждом типе общей памяти было проведено до 1000 циклов измерения с различным числом передач маркера в каждом цикле. Результаты измерений приведены в табл. 1.

Из представленных в табл. 1 результатов видно, что время передачи маркера существенно зависит от способа обмена информацией. При этом более

Таблица 1

Тип памяти	Время передачи маркера, нс		
	Среднее	Минимальное	Максимальное
Виртуальная	313	310	345
DDR3	4204	4060	4364
Память SoC	4553	4408	4662

детальный анализ показывает, что основное время, затрачиваемое на передачу данных, тратится не на их запись в какую-либо общую память машинными командами процессора ARM, а на физическую передачу информации. Для наглядности в табл. 2 приведено время, необходимое для записи трех аналогичных пакетов с данными в каждую из исследуемых областей памяти без ожидания подтверждения получения данных со стороны получателя. Учитывая, что одна передача маркера сопровождается как раз тремя пакетами данных, разница между аналогичными записями в табл. 1 и 2 позволяет судить о скорости работы каждого из используемых интерфейсов.

Разница около 100 нс в строке с виртуальной памятью обусловлена тем, что и на стороне отправителя, и на стороне получателя не происходит анализа пакетов и формирования корректных ответов на них. Указанную разницу логично добавить к данным по времени работы с DDR3 и памятью SoC в табл. 2. Однако в этом случае среднее время записи пакетов будет отличаться от приведенного в табл. 1 в несколько раз, из чего можно сделать неутешительные выводы о скорости работы физических интерфейсов. Более подробная информация о времени выполнения некоторых типовых операций приведена в табл. 3.

Примечательно, что чтение одного 32-битного слова из памяти ПЛИС занимает столько же времени, сколько и полная передача маркера (со всеми подтверждениями) самому себе через собственную виртуальную память задачи. За время такой передачи при использовании тестового пакета считывается или записывается не меньше 64 32-битных слов, из чего напрашивается вывод о том, что скорость обращения к памяти ПЛИС меньше скорости обращения к собственной виртуальной памяти задачи на ARM в 64 раза. Корректность такого утверждения может

Таблица 2

Тип памяти	Время записи трех пакетов, нс		
	Среднее	Минимальное	Максимальное
Виртуальная	208	207	230
DDR3	359	354	384
Память SoC	1536	1517	1627

Таблица 3

Операция	Время выполнения, нс		
	Среднее	Минимальное	Максимальное
Обращение к одному слову в памяти ПЛИС	334	332	362
Считывание показаний таймера ПЛИС	312	300	420
Считывание показаний таймера TTC	203	203	203
Считывание показаний глобального таймера	80	42	237
Получение информации о готовности пакета от ПЛИС	1003	995	1 085
Выполнение тестового участка на ПЛИС	2170	2046	2419

подвергаться сомнению, так как при работе с виртуальной памятью активно используется кэш-память. Вместе с тем используя данные о времени записи пакетов в память SoC из табл. 2 и времени обращения к одному слову в памяти ПЛИС, происходит примерно в 14 раз медленнее, чем к памяти SoC. Аналогичный расчет для DDR показывает, что одно обращение к памяти ПЛИС происходит почти в 60 раз медленнее, чем аналогичное обращение DDR.

Следует также обратить внимание на достаточно заметное время выполнения тестового участка на ПЛИС. Участок представляет собой цикл, в котором над каждым словом из входных данных поочередно выполняется операция XOR с одним и тем же заранее известным числом, после чего слово отправляется в пакет с результатом. Сложно судить о том, является ли описанная процедура сложной и длинной или, наоборот, простой и короткой. Однако поделив среднее время выполнения тестового участка на его длину (23 слова), можно уверенно утверждать, что в данном случае чистое время выполнения аналога одной машинной команды ARM на ПЛИС занимает чуть более 100 нс (или 5 тактов таймера ПЛИС).

Используя данные табл. 1—3, несложно оценить, что примерное время выполнения одного линейного участка (ЛУ) архитектуры MAK на ПЛИС при использовании в качестве общей памяти ОЗУ DDR3 составит чуть более 10,5 мкс (среднее время передачи маркера 2 плюс время вычислений на ПЛИС). Аналогично время выполнения одного ЛУ при использовании для обмена данными памяти на кристалле составит около 11,3 мкс. При этом передача маркера туда и обратно займет 8,5...9 мкс, а выполнение участка на ПЛИС — чуть более 2 мкс.

Данных, полученных в ходе выполнения тестов, оказалось достаточно для построения еще одного дополнительного среза. В процедуре передачи маркера участвуют программа на стороне ARM, прошивка на стороне ПЛИС и среда передачи данных. Измерение нагрузки на каждую из указанных сторон позволяет оценить и сравнить между собой среднее время работы каждого из участников процесса в ходе передачи. Подробная информация о времени работы

Таблица 4

Операция	Среднее время выполнения, нс, при использовании в качестве общей памяти	
	DDR3	памяти на кристалле
Полное время выполнения участка на ПЛИС	10 577	11 276
В том числе:		
работа на стороне ARM	417	417
работа на стороне ПЛИС	2736	2736
работа каналов связи	7424	8124

вычислителей и каналов связи приведена в табл. 4, и она не является достаточно точной. Причина в том, что многие параметры можно замерить только опосредованно, однако представленные данные вполне могут служить основой для выбора направлений дальнейшей оптимизации процесса передачи маркера и выполнения части программного кода на стороне ПЛИС.

Учитывая, что для тестов была использована одна из самых доступных, а значит не самая быстродействующая платформа, можно рассчитывать на увеличении скорости передачи маркера на профессиональном оборудовании, например, TI Keystone2 (вычислители ARM Cortex-A15 и DSP C66x). У такого оборудования помимо оперативной памяти имеется специальная высокоскоростная память, предназначенная для обмена информацией между вычислителями. Время доступа к такой памяти примерно вдвое ниже времени доступа к расположенной на плате оперативной памяти DDR3, а доступ к ОЗУ на Keystone2 осуществляется примерно в 1,3 раза быстрее, чем на Zybo Zynk-7000. Таким образом, прогноз по времени передачи маркера на Keystone2 при использовании в качестве общей памяти высокоскоростной памяти на кристалле не превышает 250 нс, а время выполнения одного линейного участка на DSP C66x не должно превышать 2...3 мкс.

Выводы и возможности применения

Предложен унифицированный подход к проектированию гетерогенных вычислительных систем, включающий в себя общий механизм передачи управления между ядрами CPU, GPU, DSP, ПЛИС и позволяющий организовать:

- единую вычислительную среду для программируемых ядер и логических структур ПЛИС;
- сквозной маршрут проектирования от модели верхнего уровня к машинному коду системы;
- автоматическую генерацию межпроцессорных коммуникаций и управления распределенными вычислениями;
- отладку на уровне системы.

Предложенный подход может применяться для создания систем:

— требующих применения нескольких вычислителей различной функциональности в целях ускорения выполнения вычислений;

— реализующих алгоритмы обработки данных без исполняемого кода посредством логических структур с автоматным управлением;

— с элементами аппаратной защиты оригинальных алгоритмов и данных от обратного анализа за счет переноса части вычислений в ПЛИС;

— с повышенной отказоустойчивостью за счет физического дублирования вычислительных ядер и передачи управления на то ядро, которое сохранило работоспособность и смогло принять маркер;

— с частичным параллелизмом внутри отдельных участников маркерного процессинга в целях параллельного выполнения части вычислений.

Список литературы

1. **ГОСТ** Р ИСО/МЭК 7498-1—99 ВОС. Базовая эталонная модель. Часть 1. Базовая модель.
2. **Медведев О. В.** Семантика языка описания аппаратуры HaSCoL // Вест. С.-Петербург. ун-та. Сер. 10. Прикл. матем. Информ. Проц. упр. 2012. № 2. С. 81—96.
3. **ГОСТ** Р 50754—95 Язык описания аппаратуры цифровых систем VHDL. Описание языка.

Asymmetric Marker Processing

M. V. Baklanovsky, m.baklanovsky@spbu.ru, **B. N. Krivoshein,** bnkv@outlook.com, **A. N. Terekhov,** a.terekhov@spbu.ru, **M. A. Terekhov,** st054464@student.spbu.ru, Saint Petersburg State University, Saint Petersburg, 199034, Russian Federation, **A. E. Sibiryakov,** a.sibiryakov@urfu.ru, Ural Federal University named after the first President of Russia B. N. Yeltsin, Ekaterinburg, 620002, Russian Federation

Corresponding author:

Terekhov Andrey N., Professor, Saint Petersburg State University, 199034, Saint Petersburg, Russian Federation, E-mail: a.terekhov@spbu.ru

Received on January 29, 2018
Accepted on February 12, 2018

An approach is proposed to create heterogeneous computing systems with the capabilities of organizing a single computing environment with a unified mechanism for transferring control, a through design route, automatic generation of a part of the system elements in the compilation process, and with debugging at the system level. All stack protocols, implementation on shared memory and work with FPGA are described. The results of speed tests of the author's implementation are given. The approach can be used for elements of hardware protection against reverse analysis, for creating high-performance systems with specialized computing cores and partial parallelism and for improving fault tolerance.

Keywords: *processor, control transfer, protocol stack, reliability, performance, security, fault tolerance*

For citation:

Baklanovsky M. V., Krivoshein B. N., Terekhov A. N., Terekhov M. A., Sibiryakov A. E. Asymmetric Marker Processing, *Programmnaya Ingeneria,* 2018, vol. 9, no. 4, pp. 156—162.

DOI: 10.17587/prin.9.156-162.

References

1. **GOST** 7498-1—99 VOS. Bazovaja jetalonnaja model'. Chast' 1. Bazovaja model' (Information technology. Open Systems Interconnection. Basic Reference Model. Part 1. The Basic Model) (in Russian).
2. **Medvedev O. V.** Semantika yazyka opisaniya apparatury HaSCoL (The semantics of a hardware description language HaSCoL), *Vestn. S.-Peterburg. un-ta. Ser. 10. Prikl. matem. Inform. Proc. upr.,* 2012, no. 2, pp. 81—96 (in Russian).
3. **GOST** 50754—95 Jazyk opisanija apparatury cifrovyh sistem VNDL. Opisanie jazyka (VHSIC Hardware Description Language. Language reference manual) (in Russian).

УДК 519.724.2 DOI: 10.17587/prin.9.163-173

И. Б. Казаков, аспирант, e-maill: i_b_kazakov@mail.ru,
Московский государственный университет им. Ломоносова

Кодирование в скрытом канале перестановки пакетов

Представлена модель скрытого канала, основанного на передаче информации между злоумышленниками посредством изменения порядка следования пакетов, которые передаются между санкционированными политикой безопасности пользователями. Введено понятие трех типов моделей ошибок, которые могут возникать в этом скрытом канале, и поставлена задача нахождения оптимальных кодов. Эта задача (с доказательством соответствующих теорем) сведена к задаче о максимальной клике, к которой применены готовые программные средства. Получены результаты для размера групп пакетов от 3 до 9 и числа исправляемых ошибок от 1 до 4. Для графов, порожденных рассматриваемыми типами ошибок, найдены большие клики. В силу того, что задача о максимальной клике NP-полна, а размеры графов значительны (число перестановок из девяти элементов равно 362880), для больших размеров групп пакетов проводили поиск только приближенно максимальных кодов. Для оценки качества решения задачи в таких предположениях получены простые нижние и верхние оценки мощности максимальных кодов.

Ключевые слова: *скрытые каналы; перестановки; пропускная способность; модели ошибок; коды, исправляющие ошибки; максимальные клики*

Введение

Скрытый канал — это коммуникационный канал, пересылающий информацию методом, который изначально не был для этого предназначен. Такое определение означает, что передача информации между злоумышленниками проводится незаметно для санкционированных политикой безопасности пользователей канала и/или стороннего наблюдателя посредством того же канала. Информация передается между злоумышленниками путем несущественной для пользователя и/или наблюдателя модификации информации со стороны злоумышленника-передатчика и считывания этой модификации со стороны злоумышленника-приемника.

Приведем некоторые известные примеры скрытых каналов. Во-первых, это цифровая стеганография: модификация изображений/аудио/видеофайлов с целью встраивания скрытого сообщения. Во-вторых, это ситуация, когда утечки информации реализуются через флаги доступа к разделяемым ресурсам: если один пользователь поставил к объекту в памяти/файлу некий флаг доступа, то другой пользователь может считать информацию, содержащуюся в значении этого флага. В-третьих, это пример, когда передача осуществляется через манипуляцию загруженностью процессора и объемом доступной памяти: один пользователь загружает процессор до 100 %, а другой этот факт регистрирует. Эти и другие подходы к организации таких утечек информации описаны, например, в работе [1].

Однако во всех перечисленных выше, а также в других случаях для практической организации скрытого канала есть препятствие. Оно состоит в том, что в потоке данных, передаваемых по этим каналам, существенно и постоянно создаются ошибки в связи с активностью санкционированных политикой безопасности пользователей и/или передаточных систем. Например, изображение подвергается компрессии, права доступа независимо назначаются сторонними приложениями, прочие процессы сами нагружают центральный процессор. Такие явления уменьшают пропускную способность канала, и это обстоятельство делает актуальной задачу, аналогичную задаче классической теории кодирования о построении оптимального кода, исправляющего ошибки. По информации, полученной в ходе библиометрических исследований, этот вопрос (в стеганографических терминах — это вопрос о робастной стеганографии) ранее мало изучался. В таких случаях "теоретики" исследовали лишь возможность передать 1 бит информации через помехи. Для открытого канала, который рассматривается в настоящей работе и именуется каналом перестановки пакетов, решение такой задачи было приведено в работе [2]. Таким образом, решив на соответствующих моделях задачу о построении оптимального кода, попытаемся развить теорию скрытых каналов от "качественного" состояния к "количественному".

Рассмотрим скрытый канал перестановки пакетов. Есть два санкционированных политикой безопасности пользователя, обменивающихся между собой информацией посредством последовательной отправки ее частей — пакетов. Например, это могут быть клиент и сервер, которые обмениваются инфор-

мацией. Злоумышленниками при этом могут быть троянские программы, установленные на клиенте и сервере. В целях незаметного обмена информацией они изменяют порядок передачи пакетов информации между сторонами. Один злоумышленник может изменить порядок отправки пакетов, оказывая влияние, например, на сетевую карту, с которой санкционированный пользователь отправляет эти пакеты. Второй злоумышленник считывает порядок полученных пакетов, сверяя порядок сборки пакетов с порядком их прибытия, или непосредственно ориентируясь на время отправки, записанное в самом пакете. Будем предполагать, что за один раз передается n пакетов, т. е. таким образом злоумышленники передают между собой перестановку $\sigma \in S_n$, где S_n — множество перестановок n элементов.

Далее рассмотрим модели (типы) возникающих в передаче и отмеченных выше ошибок. На основании этих моделей построим соответствующие им графы и метрические пространства. Формализуем задачу как процесс построения множества непересекающихся шаров и сведем ее к общеизвестной и хорошо изученной задаче нахождения максимальной клики/независимого множества, которую в основном приближенно, а на малых параметрах — точно численно решим на ЭВМ с использованием готовых инструментальных средств.

Отметим, что в настоящей работе представлены результаты исследований передачи группами от трех до девяти пакетов, и коды, исправляющие от одной до четырех ошибок.

1. Ошибки и их модели

Будем считать, что с переданной перестановкой пакетов могут в принципе произойти следующие два вида явлений:

- могут быть переставлены два соседних пакета;
- какой-нибудь пакет при передаче может вообще потеряться.

Или же может произойти и то, и другое, и притом несколько раз.

Изучим возможность составления кода на перестановках, исправляющего *err* ошибок, т. е. такого подмножества S_n, которое второй злоумышленник может распознать, несмотря на *err* нарушений порядка передачи пакетов в канале. Это означает, что нет такого элемента пространства, которого можно достичь одновременно из двух разных элементов кода не более, чем за *err* искажений.

В соответствии с изложенным выше дадим понятие трех моделей ошибок, а также определим соответствующие им пространства, элементами которых являются возможные результаты считывания последовательности пакетов на стороне приемника.

Пронумеруем пересылаемые пакеты 1...n и будем рассматривать пересылку n пакетов в порядке, заданном злоумышленником, как пересылку последовательности из n различных чисел от 1 до n. Эту последовательность назовем сообщением скрытого канала. Дадим несколько определений, которые необходимы для дальнейшего изложения материала.

Модель ошибки 1 — это модель, которая учитывает возможность перестановок только двух соседних пакетов. Такое определение означает, что в терминах этой модели два сообщения скрытого канала получены одной ошибкой друг из друга, когда одно из них можно получить из другого, переставив две соседние позиции. Например, от сообщения 1234 на одну ошибку отличаются 2134, 1324, 1243. Соответствующее пространство S_n — это множество перестановок с заданной на нем структурой графа, в котором две перестановки смежны тогда и только тогда, когда они различаются на транспозицию двух соседних позиций.

Модель ошибки 2 — это модель, которая учитывает возможность только потери одного пакета. Соответственно, два сообщения определены как полученные одной ошибкой, если одно из них получено пропуском какой-либо позиции другого сообщения. Например, от сообщения 1234 на одну ошибку отличны 123, 124, 234, а от сообщения 123 на одну ошибку отличны 1234, 12, 23, 13 (при $n = 4$).

Соответствующее пространство определяется как $H_n^k = S_n^0 \sqcup S_n^1 \sqcup ... S_n^k$, где S_n^k — множество последовательностей из $n - k$ различных элементов, выбранных из множества {1, 2, 3,..., n}. Это означает, что это пространство состоит из $k + 1$ слоя, а каждый слой состоит из (всех) перестановок, из которых выкинуто i позиций, где i меняется от 0 до k ($k = err$ — числу исправляемых ошибок).

Структура графа на этом пространстве следующая: два элемента смежны тогда и только тогда, когда они лежат в некотором i-м и $i + 1$-м слоях S_n^i, S_n^{i+1}, и элемент из $i + 1$-го слоя получается из элемента i-го слоя посредством выбрасывания одной позиции. Пример такой смежности отмечен ранее.

Модель ошибки 3 — это комбинированная модель, которая учитывает возможность как перестановки соседних пакетов, так и пропуска пакетов. Отличающиеся на одну ошибку сообщения определены как отличающиеся на одну ошибку или в терминах первой модели, или в терминах второй. Например, от сообщения 1234 на одну ошибку отличны 123, 124, 234, 2134, 1324, 1243. Соответствующее пространство то же самое, что и в модели 2, т. е. H_n^k. Структура графа определена более плотно: два элемента смежны тогда и только тогда, когда они смежны в смысле модели 2, или когда они лежат в одном слое и различаются на транспозицию двух соседних позиций (подобно модели 1). Примеры отмечены выше.

2. Формализация и сведение к задаче о максимальной клике

В предыдущем разделе были определены пространства S_n, H_n^k, на которых введено понятие смежности элементов, что позволяет рассматривать их как некие конечные графы. Для трех представленных выше моделей ошибок получено три вида графов соответственно.

Рассмотрим заново процесс передачи перестановки. Передатчик передает перестановку из S_n или верхнего слоя (S_n^0) H_n^k, по ходу передачи она искажается от 0 до *err* раз. По принятому определению структуры графа каждому искажению соответствует переход от данного к смежному элементу вдоль ребра, и, следовательно, приемник получит элемент этого же пространства, такой что в графе существует путь длины не более чем *err* между переданным и полученным элементами пространства.

Возможный код — это, очевидно, такое подмножество перестановок, передаваемых передатчиком, что ни для какой пары из них не может случиться так, чтобы после искажения они перешли в один и тот же элемент пространств S_n, H_n^k. Заметим, что это есть необходимое и достаточное условие того, чтобы приемник мог однозначно определить, какой же кодовый элемент был отправлен. В свете изложенного выше этот факт означает, что не должно быть таких элементов пространства, к которым из двух различных кодовых элементов одновременно есть пути длины не более *err*. Это значит, что шары с центрами в кодовых элементах и радиуса *err* не пересекаются. Далее определим понятие шара и докажем строго интуитивно очевидное утверждение: совместимость наличия пары элементов в одном коде равносильна отсутствию пути длины 2*err* между ними.

Для начала зададим метрику на наших пространствах.

Определение. Пусть $a, b \in S_n$ (H_n^k) Тогда определим $d(a, b)$ как длину наикратчайшего пути между вершинами графа ошибок, заданного на этом пространстве.

Утверждение. Длина d удовлетворяет свойствам метрики.

Доказательство.

1. Очевидно $d(a, a) = 0$, так как из a в a идет нулевой путь.

2. Выполняется равенство $d(a, b) = d(b, a)$ — в силу неориентированности графа. Любой путь из a в b является также путем из b в a.

3. Пусть l_1 — кратчайший путь из a в b, а l_2 — кратчайший путь из b в c. Тогда $|l_1| = d(a, b)$, $|l_2| = d(b, c)$.

4. Рассмотрим путь l_3 из a в c, полученный совмещением конца пути l_1, с началом пути l_2. $|l_3| = |l_1| + |l_2|$.

5. Тогда выполнено $d(a, c) \leqslant |l_3| = |l_1| + |l_2| = d(a, b) + d(b, c)$, что и доказывает неравенство треугольника. Что и требовалось доказать.

Теперь можно определить, что такое шар.

Определение. $B_{err}(a)$ — шар радиуса *err* с центром в a (т. е. множество таких b, что $d(a, b) \leqslant err$). Это равносильно тому, что в шар радиуса *err* с центром a входят те и только те элементы, к которым от a есть путь длины не более *err*. А этот факт означает, что теперь можно формально повторить данное ранее определение кода.

Определение. Множество K — кодовое, исправляющее *err* ошибок, если для любых $a, b \in K$ шары $B_{err}(a)$ и $B_{err}(b)$ не пересекаются.

Примечание. Код — это не всегда то же самое, что и кодовое множество. В модели ошибки 1 это тождественные понятия, а в моделях ошибок 2, 3 кодовое множество является кодом тогда и только тогда, когда оно лежит целиком в верхнем слое S_n^0.

Таким образом, задача поиска кода сводится к отысканию множества непересекающихся шаров радиуса *err* в заданном метрическом пространстве Хемминга. Докажем далее основную теорему. Она является аналогом общеизвестного факта классической теории кодов, исправляющих ошибки (например, [3], часть IV, теорема 10).

Теорема. Шары радиуса *err* в заданном метрическом пространстве пересекаются тогда и только тогда, когда расстояние между их центрами менее или равно 2*err*.

Доказательство.

I. Необходимость.

1. Пусть a, b — центры шаров.

2. Пусть выполнено $d(a, b) > 2err$ и шары пересекаются.

3. Пусть c входит в пересечение шаров. Тогда выполнено $d(a, c) \leqslant err$, $d(c, b) \leqslant err$.

4. Откуда $d(a, b) \leqslant d(a, c) + d(c, b) \leqslant err + err = 2err$. Противоречие. Следовательно, при $d(a, b) > 2err$ шары не пересекаются.

II. Достаточность.

5. Пусть теперь $d(a, b) \leqslant 2err$, и шары не пересекаются.

6. Пусть l — кратчайший путь из a в b. Тогда $|l| \leqslant 2err$.

7. Положим L_1 — множество таких элементов пути l, которые лежат в $B_{err}(a)$, L_2 — множество таких элементов пути l, которые лежат в $B_{err}(b)$.

Примечание. В L_1, L_2 не лежат сами a, b.

8. По п. 5 $L_1 \cap L_2 = \varnothing$.

9. Также известно, что $d(a, b) \geqslant |L_1| + |L_2| + 1$.

Для окончания доказательства теоремы исследуем свойства множеств L_1, L_2.

Лемма 1. $|L_1| \leqslant err$, $|L_2| \leqslant err$.

Доказательство.

1. Предположим обратное, а именно: $|L_1| > err$.

2. Пусть $c \in L_1$. Обозначим $d_1(a, c)$ длину отрезка пути l между a и c.

3. Так как $|L_1| > err$, то найдется такое $c' \in L_1$, что $d_1(a, c') > err$. Это выполнено, так как для различных $c_1, c_2 \in L_1$ верно $d_1(a, c_1)! = d_1(a, c_2)$, $d(a, c_1) \geqslant 1$, и такое c' найдется по принципу Дирихле.

4. Выберем опять некое произвольное $c \in l$.

5. Тогда очевидно, что $d(a, c) \leqslant d_1(a, c)$.

6. Предположим также, что $d(a, c) < d_1(a, c)$.

7. Тогда заменим отрезок l между a и c на кратчайший путь от a до c. И заметим, что после этой операции путь l укоротился.

8. Что противоречит тому, что он кратчайший. И, следовательно, выполнено равенство $\forall c \in l\ d(a, c) = d_1(a, c)$.

9. В том числе и $d(a, c') = d_1(a, c')$. Следовательно, $d(a, c') > err$.

10. По определению L_1 c' не лежит в L_1. Противоречие доказывает лемму, L_2 рассматривается аналогично.

Введем упорядочение вдоль пути l.

Определение. Пусть $c_1, c_2 \in l$. Положим, что $c_1 < c_2$, если c_1 лежит на отрезке от a до c_2 (не включая c_2). Очевидно, что этот порядок линеен.

Определение. $A \subset l$ замкнуто вниз, если $c \in A$, $c' < c \Rightarrow c' \in A$.

Определение. $A \subset l$ замкнуто вверх, если $c \in A$, $c' > c \Rightarrow c' \in A$.

Лемма 2. $L_1 \cup \{a\}$ замкнуто вниз, $L_2 \cup \{b\}$ замкнуто вверх.

Доказательство.

1. Пусть $c \in L_1 \cup \{a\}$, $c' < c$.
2. $c' < c \Rightarrow c! = a \Rightarrow c \in L_1$.
3. Если выполнено $c' = a$, то все очевидно. Считаем далее, что $c'! = a$.
4. Пусть l_c, $l_{c'}$ — отрезки l от a до c, c' соответственно. Тогда $|l_c| < |l_{c'}|$.
5. В п. 8 леммы 1 было доказано, что $|l_c| = d_1(a, c) = d(a, c)$, $|l_{c'}| = d_1(a, c') = d(a, c')$.
6. И следовательно, верно $d(a, c') = |l_{c'}| < |l_c| = d(a, c)$.
7. Однако выполнено $d(a, c) \leqslant err$.
8. Откуда $d(a, c') \leqslant err$. И по определению L_1 $c' \in L_1 \Rightarrow c' \in L_1 \cup \{a\}$.
9. $L_2 \cup \{b\}$ рассматривается аналогично.

Таким образом, лемма доказана.

Определение. Назовем p-м элементом пути l такой, что длина отрезка от a до p-го элемента равна p. Если $p \leqslant |l|$, то такой элемент существует, достаточно пройти по пути p шагов, начиная с конца пути a.

Лемма 3. Справедливы неравенства $|L_1| \geqslant err$, $|L_2| \geqslant err$.

Доказательство.

1. Предположим, что $|L_1| < err$.
2. Пусть $c \in l$ — err-й элемент.
3. Тогда $d(a, c) = d_1(a, c) = err$ (см. Лемму 1 выше, п. 8).
4. Следовательно, $c \in L_1$.
5. Отрезок от a до c (не включая a, включая c) содержит err различных элементов. В силу замкнутости вниз все они лежат в L_1.
6. Но $|L_1| < err$, получаем противоречие. Следовательно, исходное предположение было ложным.
7. Доказательство для L_2 рассматривается аналогично. Лемма доказана.

10. Итак, из п. **9** и доказанной выше леммы 3 следует, что $d(a, b) \geqslant |L_1| + |L_2| + 1 \geqslant err + err + 1 = 2err + 1 \Rightarrow d(a, b) > 2err$. Это противоречит предположению из п. **5**, а следовательно, оно является ложным. Таким образом, верно утверждение теоремы: если выполнено $d(a, b) \leqslant 2err$, то шары пересекаются.

Теорема доказана.

По нашему определению метрики из доказанной теоремы следует, что множество является кодовым тогда и только тогда, когда любые два его элемента удалены друг от друга на расстояние более $2err$. Введем еще одно ключевое понятие для графов, сводящее непересекающиеся шары к независимым множествам, и следовательно, поиск кода к задаче о максимальной клике.

Определение. Пусть дан граф $G = (V, E)$. k-й степенью графа G^k назовем такой надграф G (надграф — граф с теми же вершинами и с множеством ребер, являющимся надмножеством ребер исходного), в котором две вершины смежны тогда и только тогда, когда существует путь между ними в графе G длины не более k.

Удаленность двух элементов друг от друга на расстояние не более чем $2err$ в исходном графе ошибок G равносильна их смежности в графе G^{2err}. Таким образом, код — это такое множество элементов, что любые два из них не смежны в графе G^{2err}. Другими словами, код, исправляющий err ошибок, — это независимое множество графа G^{2err}, или иначе — клика дополнения этого графа.

Примечание. Для моделей ошибки 2, 3 еще должно быть выполнено условие того, что элементы независимого множества, которое задает код, лежат в слое S_n^0 пространства H_n^k, так как пересылаются лишь целые перестановки. Это условие означает, что независимое множество будет искаться среди подграфа $\left(H_n^k\right)_{|S_n^0|}^{2err}$.

Конкретным представлением графа, пригодным к вычислению максимального независимого множества, является его матрица смежности.

Определение. Матрица **A** кодовой несовместимости — это матрица смежности графа G^{2err}. Строки и столбцы этой матрицы соответствуют элементам $S_n(H_n^k)$, а значения $a_{ij} = 1$, если выполнено $d(a_i, a_j) \leqslant 2err$, и $a_{ij} = 0$, если выполнено $d(a_i, a_j) > 2err$.

Таким образом, цель исследования свелась теперь к известной задаче поиска клик/независимых множеств графа. Далее коды будут вычислены согласно способу, изложенному выше, в две стадии:

1) формирование матрицы кодовой несовместимости;

2) поиск как можно большего независимого множества/наиболее эффективного кода по данной матрице кодовой несовместимости.

3. Перечисление элементов кодовых пространств

Для того чтобы сформировать матрицу кодовой несовместимости, сначала нужно уметь перечислять соответствующие элементы H_n^k, а также и S_n как частный случай, так как $S_n = H_n^0$. Для этой цели будет применена модификация известного алгоритма Нарайаны (представлен в работе [4]) перечисления перестановок.

Будем перечислять элементы S_n^k в лексикографическом порядке, используя определенные далее три базовые операции над подстановками. Эти операции представлены как последовательности чисел для получения следующего в лексикографическом порядке элемента.

Определение. Увеличиваемым элементом последовательности a_i назовем такой, для которого выполнено условие: $\exists b \in (\sim A)\ a_i < b$, где $A = \{a_1,..., a_i\}$, $\sim A = \{1, 2,..., n\} \backslash A$.

Операция 1: нахождение последнего увеличиваемого элемента.

Начиная с конца последовательности a_k, будем последовательно проверять элементы на увеличиваемость, пока не найдем увеличиваемый элемент. Если такого элемента нет, то конец перечисления S_n^k уже был достигнут.

Операция 2: увеличение найденного a_i.

Пусть a_i — последний увеличиваемый элемент. Заменим его на наименьшее такое $b \in (\sim A)$, что $a_i < b$.

Операция 3: достройка минимального хвоста.

Зададим новое множество $A' = \{a_1,..., a'_i\}$, где $a'_i = b$. Далее упорядочим по возрастанию $\sim(A')$, и заменим элементы $a_{i+1},..., a_k$ на первые $k - i$ элементов $\sim(A')$. Таковые всегда найдутся, так как $|\sim(A')| = n - |A'| = n - i \geqslant k - i$, так как $n \geqslant k$.

Алгоритм начинает работу с последовательности 1...k и последовательно применяет описанные выше шаги, пока это возможно.

Докажем теперь корректность работы этого аглоритма.

Теорема. Заданный алгоритм перечисляет все элементы S_n^k в лексикографическом порядке.

Для формализации доказательства этой теоремы введем ряд вспомогательных понятий и докажем представленные далее вспомогательные утверждения.

- Относительно элементов S_n^k и порядка их следования.

Определение. Последовательность σ назовем регулярной, если в ней все элементы различны.

Определение. Пусть σ — некая регулярная последовательность. Обозначим через *inc*(σ) лексикографически следующую последовательность, т. е. наименьшую из регулярных последовательностей той же длинны, лексикографически больших, чем σ.

Утверждение (очевидно). Пусть σ — некая регулярная последовательность, такая что ее последний элемент является увеличиваемым. Тогда *inc*(σ) получается увеличением последнего элемента на 1.

- Относительно регулярных последовательностей, начинающихся с некоторого префикса.

Определение. Пусть σ, ν — две последовательности. Последовательностью σν назовем их конкатенацию.

Определение. Последовательностями из R_σ^k длины k с префиксом σ назовем последовательности длины k вида σν, такие что $\sigma\nu \in S_n^{n-k}$.

Определение. Пусть σ — последовательность. Обозначим [σ] множество всех ее элементов.

Определение. $\sim[\sigma] = \{1,..., n\}/[\sigma]$.

Определение. Последовательность ν согласована с последовательностью σ, если $[\nu] \subset \sim[\sigma]$.

Определение. Пусть σ — последовательность, $\nu_\sigma^{\min}$ — минимальная регулярная последовательность из заранее заданного числа элементов s, согласованная с σ. Очевидно, она состоит из первых s минимальных элементов множества ~[σ].

Определение. $\nu_\sigma^{\max}$ определяется аналогично, как первые s максимальных элементов ~[σ].

Утверждение. Лексикографически минимальным элементом R_σ^k является $\sigma\nu_\sigma^{\min}$, а максимальным $\sigma\nu_\sigma^{\max}$. Это очевидно, так как на элементах вида σν лексикографический порядок совпадает с лексикографическим порядком на таких ν.

Рассмотрим теорему об увеличиваемом элементе в постфиксе.

Теорема. Пусть ν — регулярная последовательность, она согласована с регулярной последовательностью σ, и $\nu! = \nu_\sigma^{\max}$. Тогда в σν в части ν есть увеличиваемый элемент.

Доказательство.

1. Возьмем ν′ — согласованную с σ последовательность, такую что ν′ > ν (по лексикографическому порядку).

2. Пусть ν′ больше чем ν в некоторой i-й позиции: $\nu_i < \nu_{i'}$.

3. Рассмотрим множество $A = [\sigma] \cup \{\nu_1,..., \nu_i\}$.

4. Предположим, что $\forall b \in A$, $b \leqslant \nu_i$, т. е. что ν_i — неувеличиваемый элемент.

5. Пусть r' — часть последовательности σν′ до элемента ν_i'; $r' = \sigma_1,..., \sigma_p, \nu,..., \nu_{i'}$.

6. r' — последовательность, все элементы которой различны. Также $[r'] = (A/\{\nu_i\}) \cup \{\nu_{i'}\}$.

7. Так как $\nu_i! = \nu_i'$, то (в силу регулярности) ν_i' не лежит в $A/\{\nu_i\}$ и, следовательно, ν_i' не лежит в A.

8. ν_i' не лежит в A, следовательно, $\nu_i' \in (\sim A) \Rightarrow \nu_i' \leqslant \nu_i$ (по п. 4). Что противоречит п. 2.

Теорема доказана.

Следствие. Пусть последовательность имеет вид σν (σ, ν — регулярны, ν согласованно с σ), где последний элемент σ является одновременно и последним увеличиваемым элементом самого σν.

Тогда $\nu = \nu_\sigma^{\max}$.

Итак, при очередном применении алгоритма возможны описаные далее три случая.

1. В последовательности вообще нет увеличиваемых элементов. В таком случае эта последовательность — уже максимальная из S_n^k. Этот факт означает, что она имеет вид $n, n-1,..., k+1$.

2. Последний элемент последовательности является увеличиваемым. В таком случае алгоритм просто увеличивает его на 1 и получается, согласно отмеченному выше утверждению, лексикографически следующий элемент.

3. Последовательность имеет вид, как в следствии представленной выше теоремы. Тогда алгоритм преобразует ее в $inc(\sigma)\nu_{inc(\sigma)}^{\min}$, так как последний элемент σ увеличивается на 1, а в операции 3, как выяснилось, выстраивается именно такая минимальная последовательность ν, согласованная с *inc*(σ).

Но $inc(\sigma)\nu_{inc(\sigma)} = inc(\sigma\nu_\sigma^{\max})$, потому что никаких других последовательностей между максимальной регулярной последовательностью с префиксом σ и минимальной регулярной последовательностью с префиксом *inc*(σ) нет.

Таким образом, алгоритм всегда преобразует σ в $inc(\sigma)$, если это возможно. Применяем его, начиная от последовательности 1,..., $n - k$ и до максимальной. Таким образом, все S_n^k будет пройдено в лексикографическом порядке. Будем проходить через все H_n^k послойно, от S_n^0 до S_n^k, и каждому элементу H_n^k будет присвоен последовательный порядковый номер.

4. Формирование матрицы кодовой несовместимости

Чтобы сформировать матрицу смежности графа ошибок, сначала нужно по заданной перестановке уметь определять ее номер, а также по номеру перечисления восстанавливать саму перестановку. Для решения второй из этих задач просто запишем в памяти ЭВМ в порядке перечисления, определенном посредством изложенного выше алгоритма, все перестановки. В итоге всегда будем иметь функцию, принимающую номер, и возвращающую перестановку с данным номером. Далее опишем способ решения первой задачи, основанный на том, что в памяти уже имеется массив перестановок, отсортированных в лексикографическом порядке.

Рассмотрим алгоритм бинарного поиска.

Шаг 1. Возьмем первоначально как концы отрезка минимальную и максимальную перестановки среди всех перестановок такой же длины, что и данная.

Шаг 2. Разделим этот отрезок пополам, т. е. выберем перестановку с номером, лежащим между номеров концов отрезка. Например, целую часть от среднего арифметического. Номера перестановок — концов отрезка, разумеется, известны. Если перестановка с этим номером тождественна изначально данной, то можно вернуть этот номер и завершить работу.

Шаг 3. Определим, в какой из частей отрезка находится исходная перестановка: если она лексикографически меньше, чем выбранная середина — то в левой (от начала до середины) части, если больше — то в правой (от середины до конца) части.

Шаг 4. Повторим шаги 2 и 3, пока не получим ответ. Или отрезок стягивается в одну точку, или изначальная перестановка может оказаться тождественной какой-нибудь из середины последовательности стягивающихся отрезков.

Итак, далее будем считать, что дана и функция определения номера перестановки.

По некоторой перестановке легко перечислить все смежные с ней перестановки в модели ошибки 1 или 2: нужно или переставить две соседние позиции (в случае модели ошибки 1), или какую-либо позицию исключить (в случае модели ошибки 2). Как следствие, так как даны функции конвертации перестановок в их номера, то дана и функция определения по данному номеру перестановки номеров смежных с ней перестановок в соответствующей модели ошибки.

Сформируем в памяти матрицу размера $|H_n^k| \times |H_n^k|$, заполнив ее нулями. Далее на каждой строке заполним единицами те столбцы, которые представляют собой номера перестановок, смежных с перестановкой с номером данной строки.

Примечание (о симметризации). В моделях ошибки 2, 3 нужно при каждой установке где-либо $a_{ij} = 1$ тут же установить и $a_{ji} = 1$, так как при перечислении смежных по удалению элементов перечисляются не все из них, а только результаты удаления позиции, а не вставки новой. В результате будет получена некая симметрическая матрица, которая является матрицей смежности графа ошибок G.

Для получения данных, непосредственно пригодных к началу вычисления кода как максимального независимого множества, необходимо еще вычислить матрицу смежности G^{2err} по уже данной матрице смежности G.

Определение. Пусть ν — вершина графа. Окрестность вершины определим как $N_\nu = \{\nu\} \cup \{w \mid w$ смежно с $\nu\}$.

Определение. Пусть A — множество вершин графа. Окрестность множества определена как объединение окрестностей его элементов.

В соответствии с матрицей смежности определена функция, которая по данной вершине возвращает ее окрестность. Следовательно, также дана и функция по множеству, возвращающая окрестность этого множества. Для того чтобы для заданной вершины ν получить множество вершин, смежных с ней в G^{2err}, нужно от $\{\nu\}$ $2err$ раз взять окрестность, а также удалить из множества само ν. В случае работы с моделями ошибки 2, 3 после получения матрицы смежности графа G^{2err} нужно обрезать ее до минора первых $n!$ столбцов и строк, которые соответствуют слою S_n^0.

5. Основные результаты вычислений

Для получения основных результатов был проведен расчет на пространствах S_3—S_7 (S_3^0—S_7^0-слоях в моделях ошибки 2, 3) и рассмотрено число ошибок *err* от 1 до 4. По описанному выше алгоритму для всех n и *err* была рассчитана матрица кодовой несовместимости, которую и сохранили в отдельном файле. Всего рассматривали три модели ошибок, пять размерностей пространств и четыре значения *err*. В результате было получено 3×5×4 = 60 файлов с матрицами смежности.

Примечание. При расчете в моделях ошибки 2, 3 следует брать пространство H_n^{err}, которое потом ограничивается до слоя S_n^0.

Был использован программный код, представленный на сайте **insilab.org/maxclique** в архиве **insilab.org/maxclique/mcqd.zip**, который реализует алгоритм, изложенный в работе [5]. Полученная программа была модифицирована таким образом, чтобы она получала на вход файл с матрицей смежности графа, и записывала найденные независимые множества в выходной файл. Были проведены запуски описанного инструментального средства на всех 60 исходных матрицах. Получены результаты двух типов: максимальные (т. е. неулучшаемые в принципе) независимые

множества и множества, приближающиеся к максимальным независимым. Инструментарий в процессе своего функционирования искал все большие множества, и с учетом этого обстоятельства время его работы составляло около полутора часов. После этого работа принудительно прерывалась. В табл. 1—3 представлены мощности найденных максимальных или приближающихся к максимальным кодов. Полужирным курсивом в них выделены именно те случаи, где выполнение программы было прервано.

Таблица 1

Мощности найденных кодов для модели ошибки 1

Число исправляемых ошибок	Пространство				
	S_3	S_4	S_5	S_6	S_7
$err = 1$	2	5	20	***93***	***513***
$err = 2$	1	2	6	***25***	***106***
$err = 3$	1	1	2	11	***36***
$err = 4$	1	1	2	4	***16***

Таблица 2

Мощности найденных кодов для модели ошибки 2

Число исправляемых ошибок	Пространство				
	S_3	S_4	S_5	S_6	S_7
$err = 1$	2	6	24	***97***	***501***
$err = 2$	1	2	4	***24***	***57***
$err = 3$	1	1	2	4	***12***
$err = 4$	1	1	1	2	4

Таблица 3

Мощности найденных кодов для модели ошибки 3

Число исправляемых ошибок	Пространство				
	S_3	S_4	S_5	S_6	S_7
$err = 1$	2	5	18	***79***	***401***
$err = 2$	1	2	4	***12***	***42***
$err = 3$	1	1	2	***3***	***9***
$err = 4$	1	1	1	2	2

6. "Жадный" алгоритм для S_8, S_9

Аналогичный представленному в предыдущем разделе расчет на пространствах S_8, S_9 непосредственно неосуществим. Причина в том, что уже на S_8 в матрице смежности возникает переполнение памяти, так как нужно хранить $(n!)^2$: $8! \times 8! = (40320)^2 = 1\,625\,702\,400$ элементов. Это требует примерно 1,51 Гбайт, если на каждую позицию матрицы нужно по байту. Таким образом, с S_8 в принципе еще возможно работать точно также как с S_3 — S_7 (хотя это нецелесообразно), но с S_9 это уже невозможно (необходимо 122,6 Гбайт). Поэтому для нахождения какого-нибудь независимого множества воспользуемся описанным далее простым "жадным" алгоритмом. При этом заметим, что он не дает никаких гарантий насчет того, будет ли полученное с его помощью независимое множество близко к максимальному.

Шаг 1. Найдем в графе какую-нибудь вершину с максимальной степенью.

Шаг 2. Удаляем эту вершину, а также все ребра, которым она инцидента.

Шаг 3. Повторяем шаги 1, 2 до тех пор пока не будет получен граф без единого ребра.

Для того чтобы получить вершину с максимальной степенью, отдельно будем запоминать степени всех вершин. Создадим массив множеств-контейнеров, в каждом из которых на каждом шаге выполнения программы будем хранить номера вершин, которые имеют степень, равную индексу этого контейнера в массиве контейнеров. Тем самым всегда будет известна высшая степень вершины, и будет легко находиться какая-либо вершина с высшей степенью. Далее, когда удаляется вершина вместе со всеми ребрами, она исключается из контейнера, в котором содержится, а также переставляются все смежные с ней вершины (т. е. их номера) в контейнер на один ниже (т. е. с индексом на один меньше в массиве контейнеров).

Тем самым свойство корректности системы контейнеров сохраняется на всех стадиях работы алгоритма. При этом алгоритм останавливается, когда все контейнеры пусты, кроме контейнера с нулевым индексом, в котором находится ответ — некое независимое множество.

Для программной реализации описанного выше алгоритма требуется (в S_8, S_9, где нет готовой матрицы кодовой несовместимости) также и функция, которая для данной вершины возвращает множество таких вершин, которые удалены от нее на расстояние не более $2err$.

Примечание. В моделях ошибки 2, 3 при начальной инициализации контейнеров нужно записывать только слой S_n^0, а от результата функции, возвращающей шар радиуса $2err$, требуется выделить лишь его пересечение с S_n^0.

Опишем такую функцию для всех заданных моделей ошибок.

А. Модель ошибки 1. Можно найти окрестность множества, как объединение окрестностей точек, определение окрестности которых дано в разд. 4. Возьмем $\{\nu\}$ и $2err$ раз возьмем окрестность.

Б. Модель ошибки 2. Каждая вершина $\nu \in S_n^k$ смежна либо с вершиной из слоя выше — S_n^{k-1}, либо с вершиной из слоя ниже — S_n^{k+1}.

Определение. Пусть $\nu \in S_n^k$ — некая вершина. Множество $down(\nu) = \{w \in S_n^{k+1} \mid w$ смежно с ν в $H_n^{err}\}$.

Для того чтобы получить из ν множество *down*(ν), надо последовательно удалять из ν по одной позиции (перестановки).

Определение. Пусть $\nu \in S_n^k$ — некая вершина. Множество $up(\nu) = \{w \in S_n^{k-1} \mid w$ смежно с ν в $H_n^{err}\}$.

Здесь надо вставить одну позицию. Для этого вставляем любое число от 1 до *n*, которого нет еще в данной перестановке.

Примечание. Если ν лежит в самом верхнем или в самом нижнем слое, то считаем, что $up(\nu) = \varnothing$ или $down(\nu) = \varnothing$ соответственно.

Собственно, алгоритм заключается в следующем: возьмем {ν}, применим к нему *err* раз операцию *up*, а затем *err* раз операцию *down*. Результат применения операции к множеству представляет объединение результатов применения ее к элементам этого множества. Необходимо доказать корректность этого алгоритма, т. е. тот факт, что в результате этих действий получится действительно искомый шар радиуса 2*err* (точнее его ограничение на верхний слой S_n^0).

Лемма 1. Пусть $a, b \in S_n^0$ — некие перестановки и между ними в H_n^{err} есть путь длины 2*p*. Тогда существует $c \in S_n^p$ такое, что $a, b \in up^{p'}(\{c\})$, где $p' \leqslant p$.

Доказательство.

1. Условие леммы означает, что *b* получается из *a p* вставками и удалениями позиции ($|a| = |b| = n$).

2. И в *a*, и в *b* есть такие позиции, которые не менялись (т. е. не были ни удалены, ни вставлены в ходе прохода пути по слоям S_n^k). Эти позиции образуют одну и ту же подпоследовательность *c*.

3. Было удалено/вставлено *p* позиций. Значит $|c| \geqslant n - p$. $|c|$ может быть более чем $n - p$, так как однажды вставленная позиция может быть впоследствии удалена.

4. Так как *c* — подпоследовательность *a*, *b*, $|c| \geqslant n - p$, то *a* или *b* могут быть получены из *c* вставкой не более чем *p* позиций, откуда очевидно следует вывод леммы. Лемма доказана.

Лемма 2. Пусть $a, b \in up^{p'}(\{c\})$, где $p' \leqslant err$, $c \in S_n^{p'}$. Тогда есть такое $c' \in S_n^{err}$, что $a, b \in up^{err}(\{c'\})$.

Доказательство.

1. Действительно, спустимся из *c* каким-либо образом (по функции *down*) до слоя S_n^{err}, и в качестве *c'* возьмем произвольный элемент $down^{err-p'}(\{c\})$.

2. Тогда в *a* или *b* можно попасть из *c'* через *err* последовательных ходов вверх. При этом $err - p'$ первых ходов идут из *c'* до *c*, а из *c* по условию леммы есть *p'* ходов до *a* или *b*. Что требовалось доказать.

Следствие. Пусть между $a, b \in S_n^0$ есть путь длины не более чем 2*err*. Тогда $\exists c \in S_n^{err}$, $a, b \in up^{err}(\{c\})$.

Примечание. Разумеется, между $a, b \in S_n^0$ есть только пути четной длины, так как на каждом шаге пути четность номера слоя меняется, при этом *a* и *b* лежат в одном слое.

Следствие. В этих же условиях $b \in up^{err}(down^{err}(\{a\}))$, что и означает корректность описанного выше алгоритма. Отметим, что обратная импликация очевидна; взятые вместе они и означают, что пересечение шара $B_{2err}(a)$ с верхним слоем S_n^0 совпадает с $up^{err}(down^{err}(\{a\}))$.

Доказательство.

1. $a \in up^{err}(\{c\}) \Rightarrow c \in down^{err}(\{a\}) \Rightarrow up^{err}(\{c\}) \in up^{err}(down^{err}(\{a\}))$.

2. $b \in up^{err}(\{c\}) \in up^{err}(down^{err}(\{a\}))$.

Корректность алгоритма, таким образом, доказана.

B. Модель ошибки 3. Здесь смежным с $\nu \in H_n^k$ являются элементы из множеств *up*(ν), *down*(ν), *neib*(ν).

Определение. Множество *neib*(ν) — вершины из того же слоя, что и ν, смежные с ν по типу смежности модели ошибки 1, т. е. полученные перестановкой каких-нибудь двух соседних позиций.

Шар радиуса 2*err* строим точно также, как в модели ошибки 1 (окрестность ν — это $\{\nu\} \cup up(\{\nu\}) \cup down(\{\nu\}) \cup neib(\{\nu\})$), а далее берем и урезаем полученный результат до его пересечения со слоем S_n^0, как это выше и требовалось.

Были проведены испытания описанного выше алгоритма на пространствах S_8, S_9 с числом ошибок от 1 до 4. В табл. 4—6 приведены результаты этих дополнительных вычислений для каждой из трех моделей ошибок, т. е. мощности найденных кодов. Эти вычисления проводились для случаев, в которых удалось за приемлемое время, от полутора до трех часов, досчитать до ответа. Прочерки в ячейках табл. 5 и 6 означают, что время расчета в соответствующих случаях оказалось слишком велико, чтобы этот расчет можно было бы проводить. Автор проводил испытания на ЭВМ с процессором Intel® Core i3-4005U CPU @ 1.70GHz × 4.

Таблица 4

Мощности (дополнительные) найденных кодов для модели ошибки 1

Число исправляемых ошибок	Пространство	
	S_8	S_9
err = 1	3114	23 400
err = 2	471	2963
err = 3	101	583
err = 4	32	150

Таблица 5

Мощности (дополнительные) найденных кодов для модели ошибки 2

Число исправляемых ошибок	Пространство	
	S_8	S_9
err = 1	2878	20 962
err = 2	211	1176
err = 3	26	—
err = 4	—	—

Таблица 6

Мощности (дополнительные) найденных кодов для модели ошибки 3

Число исправляемых ошибок	Пространство	
	S_8	S_9
$err = 1$	2249	16 273
$err = 2$	144	757
$err = 3$	—	—
$err = 4$	—	—

7. Априорные тривиальные оценки

В предыдущих разделах были приведены расчеты, в результате которых были получены конкретные коды. Однако возможно также определить по известным формулам верхние и нижние пределы, в которых может находиться мощность максимального независимого множества. Сравнение с этими данными позволит оценить эффективность выполненного вычисления.

7.1. Тривиальные верхние оценки

Модель ошибки 1. Код в этой модели — это множество непересекающихся шаров радиуса *err*. Из соображений симметрии ясно, что все они имеют одинаковый объем (число элементов в них). Очевидно также, что произведение числа этих шаров на объем каждого шара не может превышать $|S_n|$. Отсюда следует сформулированное далее утверждение.

Утверждение. Справедливо неравенство $k_{err}(S_n) \leqslant |S_n|/V_n^{err}$, где $k_{err}(S_n)$ — мощность максимального кода, исправляющего *err* ошибок, V_n^{err} — объем шара радиуса *err* в S_n. В табл. 7 представлены полученные верхние оценки.

Таблица 7

Оценочная таблица (сверху) для модели ошибки 1 со значением $[|S_n|/V_n^{err}]$ в ячейках

Число исправляемых ошибок	Пространство						
	S_3	S_4	S_5	S_6	S_7	S_8	S_9
$err = 1$	2	6	24	120	720	5040	40 320
$err = 2$	1	2	8	36	186	1152	8247
$err = 3$	1	1	4	14	66	363	2341
$err = 4$	1	1	2	7	28	141	824

Примечание. Для $err = 1$ все оценки $(n - 1)!$ Это следует из того, что в S_n степень любой вершины равна $n - 1$.

Модель ошибки 2. Здесь шар радиуса *err* и с центром в самом верхнем слое S_n^0 состоит из $err + 1$ частей в разных слоях — это множества $\{\nu\}$, $down(\{\nu\})$, $down_2(\{\nu\})...down_{err}(\{\nu\})$, где ν — центр шара.

Для того чтобы какое-нибудь $A \subset S_n^0$ было кодом, нужно чтобы $\forall \nu$, $w \in A$, $down_i(\{\nu\}) \cap down_i(\{w\}) = \varnothing$ для всех $i = 0, 1,..., err$. Откуда следует следующее утверждение.

Утверждение. Справедливо неравенство $k_{err}(H_n^k) \leqslant |S_n^i| / |down_i(\{\nu\})|$, $\forall i = 0, 1,..., err$.

Множество S_n^i состоит из последовательностей длины $n - i$, в которых все элементы различны и взяты из $\{1...n\}$. Следовательно, $|S_n^i| = n(n-1)...(i+1) = n!/i!$. Все элементы $down_i(\{\nu\})$ также получаются вычеркиванием i позиций из n-элементной последовательности ν. Откуда $down_i(\{\nu\}) = C_n^i$.

Как следствие, $k_{err} \leqslant (n!/i!)/(n!/i!(n-i)!) = (n-i)!$ $\forall i = 0, 1,..., err$. Среди них наименьшее — $(n - err)!$ В табл. 8 представлены полученные верхние оценки.

Таблица 8

Оценочная таблица (сверху) для модели ошибки 2 со значением $(n - err)!$ в ячейках

Число исправляемых ошибок	Пространство						
	S_3	S_4	S_5	S_6	S_7	S_8	S_9
$err = 1$	2	6	24	120	720	5040	40 320
$err = 2$	1	2	6	24	120	720	5040
$err = 3$	1	1	2	6	24	120	720
$err = 4$	1	1	1	2	6	24	120

Модель ошибки 3. Здесь код — это множество непересекающихся шаров радиуса *err* в пространстве H_n^k с центрами в верхнем слое S_n^0. Их число оценивает следующее утверждение.

Утверждение. Справедливо неравенство $k_{err} \leqslant |H_n^{err}|/V_n^{err}$, V_n^{err} — объем шара в модели ошибки 3. В табл. 9 представлены полученные верхние оценки.

Таблица 9

Оценочная таблица (сверху) для модели ошибки 3 со значением $[|H_n^{err}|/V_n^{err}]$ в ячейках

Число исправляемых ошибок	Пространство						
	S_3	S_4	S_5	S_6	S_7	S_8	S_9
$err = 1$	2	6	24	120	720	5040	40 320
$err = 2$	1	2	6	23	114	676	4676
$err = 3$	1	1	2	6	24	118	691
$err = 4$	1	1	1	2	7	27	131

7.2. Тривиальные нижние оценки

Воспользуемся оценкой из работы [6] (см. формулу (21)): $\omega(G) \geqslant 1/(1-\delta)$, где $\delta = 2m/s^2$, s — число вершин G; m — число ребер G. Во всех рассматриваемых графах (S_n^{2err}, $\left(H_n^k\right)_{|S_n^0}^{2err}$ в моделях ошибки 2 и 3) степени всех вершин одинаковы. Обозначим эту степень за $\deg = V_n^{2err} - 1$, V_n^{2err} — объем шара радиуса $2err$ (его пересечения с верхним слоем S_n^0). Число вершин в этих графах $s = n!$

Таблица 10

Оценочная таблица (снизу) для модели ошибки 1 со значением $[n!/V_n^{2err}]$ в ячейках

Число исправляемых ошибок	Пространство						
	S_3	S_4	S_5	S_6	S_7	S_8	S_9
$err = 1$	2	3	9	36	187	1152	8248
$err = 2$	1	2	3	8	29	142	825
$err = 3$	1	1	2	3	9	33	158
$err = 4$	1	1	1	2	4	12	45

Таблица 11

Оценочная таблица (снизу) для модели ошибки 2 со значением $[n!/V_n^{2err}]$ в ячейках

Число исправляемых ошибок	Пространство						
	S_3	S_4	S_5	S_6	S_7	S_8	S_9
$err = 1$	2	3	8	28	137	807	5583
$err = 2$	1	2	2	4	12	46	230
$err = 3$	1	1	2	2	3	6	21
$err = 4$	1	1	1	2	2	3	4

Таблица 12

Оценочная таблица (снизу) для модели ошибки 3 со значением $[n!/V_n^{2err}]$ в ячейках

Число исправляемых ошибок	Пространство						
	S_3	S_4	S_5	S_6	S_7	S_8	S_9
$err = 1$	2	3	6	23	108	621	4220
$err = 2$	1	2	2	3	9	32	154
$err = 3$	1	1	2	2	2	5	14
$err = 4$	1	1	1	2	2	2	3

В рассматриваемом случае ищется кликовое число графа-дополнения. Всего ребер имеется $s\deg/2$, в полном графе их было бы $s(s-1)/2$. Как следствие, $m = (s/2)(s - 1 - \deg)$, $\delta = (1/s)(s - 1 - \deg)$, $1/(1-\delta) = s/(\deg + 1) = s/V_n^{2err} = n!/V_n^{2err}$.

Рассмотрим оценочные табл. 10—12. В ячейках этих таблиц прописано значение $[n!/V_n^{2err}]$.

Заключение

Рассмотрена модель организации скрытого канала на основе перестановки пакетов. Проанализированы пространства трех моделей (типов) ошибок размерности от 3 до 9, число ошибок от 1 до 4. Получены оценки кодов сверху и снизу, а также, в большинстве случаев, и сами эти коды, являющиеся множествами номеров перестановок. После декодирования номеров в соответствующие им перестановки эти коды могут быть непосредственно использованы для организации скрытого канала. Однако на этом возможное исследование не закончено, так как, во-первых, обнаружены пространства S_n, H_n^k, а во-вторых, существуют и другие способы (модели) организации скрытых каналов.

Что касается первого направления исследований, прежде всего следует изучить вопрос об эвристических независимых множествах (четных степеней графа). Такое изучение предполагает постановку вопроса о алгоритмически порождаемых кодах, а также об алгоритмах и сложности декодирования по ним, т. е. об аналоге классической теории кодирования в данных пространствах, например, на тех же перестановках. По этой теме готовятся дальнейшие публикации, с углубленным изучением данного вопроса. В представленном в настоящей работе подходе оставлен за рамками вероятностный подход к числу ошибок, а именно он очень важен в реальных скрытых каналах.

Список литературы

1. **Lampson B. W.** A note on the confinement problem// Commun. ACM. 1973. Vol. 16, No. 10. P. 613—615.

2. **Galatenko A., Grusho A., Kniazev A., Timonina E.** Statistical covert channels through proxy server // Lecture Notes in Computer Science. 2005. Vol. 3685. P. 424—429.

3. **Яблонский С. В.** Введение в дискретную математику. М.: Наука, 1986. 384 с.

4. **Knuth D. E.** The Art of Computer Programming, Boston, Addison-Wesley. Vol. 4, Fascicle 2, 2005. 128 p.

5. **Janez K., Dusanka J.** An improved branch and bound algorithm for the maximum clique problem//MATCH Commun. Math. Comput. Chem. 2007. Vol. 58. P. 569—590.

6. **Bomze I. M., Budinich M., Pardalos P. M., Pelillo M.** The maximum clique problem. Handbook of combinatorial optimization. 1999. Vol. 4 (1). 74 p.

Coding in a Covert Channel of Data Packages′ Permutations

I. B. Kazakov, i_b_kazakov@mail.ru, Lomonosov Moscow State University, Moscow, 119234, Russian Federation

Corresponding author:

Kazakov Ilia B., Postgraduate Student, Lomonosov Moscow State University, Moscow, 119234, Russian Federation, E-mail: i_b_kazakov@mail.ru

Received on January 19, 2018
Accepted on February 13, 2018

We investigate a covert channel based on permutation of network packets. Consider n packets with increasing packet numbers. A malicious agent can swap the order in which packets are transmitted, thus allowing to encode n! values. However during transmission packets can be further swapped, so the permutation sent can differ from the permutation received. In order to make the channel error tolerant we construct a number of error-correcting codes. We consider three error models. In the first model we only allow to change packet order; in the second model we only allow packet insertion/deletion; the third model combines the first and the second one. The value of n varies from 3 to 9; the number of errors varies from 1 to 4. The problem is to construct an error-correcting code with the maximal cardinality (in other words, to find the maximal code). We prove that this problem is reduced to the well-known maximal clique problem, and search for large cliques in graphs generated by our error models. Since the maximal clique problem is NP-hard, and graph sizes are significant (the number of permutations on 9 elements is equal to 362880), for large values of n we found only approximately maximal codes. In order to estimate the quality we introduce simple lower and upper bounds on the cardinality of maximal codes.

Keywords: *covert channels, permutations, channel capacity, models of error, error correcting codes, maximum clique problem*

For citation:

Kazakov I. B. Coding in a Covert Channel of Data Packages′ Permutations, *Programmnaya Ingeneria,* 2018, vol. 9, no. 4, pp. 163—173.

DOI: 10.17587/prin.9.163-173

References

1. **Lampson B. W.** A note on the confinement problem, *Commun. ACM,* 1973, vol. 16, no. 10, pp. 613—615.
2. **Galatenko A., Grusho A., Kniazev A., Timonina E.** Statistical covert channels through proxy server, *Lecture Notes in Computer Science*, 2005, vol. 3685, pp. 424—429.
3. **Jablonskij S. V.** *Vvedenie v diskretnuju matematiku* (Introduction to Discrete Mathematics), Moscow, Nauka, 1986, 384 p. (in Russian).
4. **Knuth D. E.** *The Art of Computer Programming*, Boston, Addison-Wesley, 2005, vol. 4, fascicle 2, 128 p.
5. **Janez K., Dusanka J.** An improved branch and bound algorithm for the maximum clique problem, *MATCH Commun. Math. Comput. Chem.,* 2007, vol. 58, pp. 569—590.
6. **Bomze I. M., Budinich M., Pardalos P. M., Pelillo M.** *The maximum clique proble*m, Handbook of combinatorial optimization, 1999, vol. 4 (1), 74 p.

УДК 004.82 DOI: 10.17587/prin.9.174-184

К. И. Костенко, канд. физ.-мат. наук, зав. каф., e-mail: kostenko@kubsu.ru,
Кубанский государственный университет, г. Краснодар

Операции когнитивного синтеза формализованных знаний

Определены содержательно полные семейства независимых классов когнитивных целей и операций над знаниями. Они связаны с фундаментальной проблемой синтеза знаний в интеллектуальных информационных системах, являющихся реализациями когнитивных целей в различных областях знаний. Формальные определения основаны на аналогах теоретико-множественных, алгебраических, логических и топологических инвариантов математических систем. Полнота классификаторов обеспечивается унификацией с инвариантами формализмов представления знаний, философскими, лингвистическими и психологическими моделями процессов мышления. Независимость классификаторов достигается сужением определения содержания отдельных классов такими свойствами, для которых возможны реализации всех элементов одного класса без использования элементов других классов. Построенные классификаторы операций и целей расширяют возможности использования конструктов формализмов представления знаний в прикладных интеллектуальных системах при автоматизации процессов синтеза знаний со сложной семантической структурой, составляющих решения различных профессиональных задач.

Ключевые слова: *формализм знаний, алгебраическая структура, задача, обработка знания, когнитивная цель, гомоморфное расширение, синтез знаний, онтология*

Введение

Для моделирования содержания областей деятельности применяют разные форматы представления знаний и задач, которые решаются с использованием формализованных знаний. Унифицированное определение формализма знаний, обобщающее существующее многообразие известных формализмов, основано на инвариантах знания, фрагмента знания, операции композиции и отношения вложения фрагментов знаний, порождающих четверку $(M, D_M, \circ, \subseteq)$ [1]. Здесь M — множество отдельных знаний, являющееся разрешимым подмножеством множества фрагментов знаний D_M; $\circ : D_M \times D_M \to D_M$ — вычислимое отображение композиции фрагментов знаний; $\subseteq$ — разрешимое бинарное отношение на D_M, моделирующее сравнение вложения содержания отдельных фрагментов знаний. Комбинациями композиций элементов D_M определяются алгебраические структуры фрагментов знаний, составляемых из этих элементов. Такие структуры в общем случае не единственные для получаемых фрагментов. Они определяют формы представления знаний и являются основой алгоритмов обработки знаний, связанных с обходами структур представления знаний. Отношение вложения определяет семантическую структуру множества D_M. В этой структуре отражаются представления о расширении содержания фрагментов знаний.

Дополнительные типы алгебраических и логических инструментальных средств (далее — инструментов) формализации знаний связаны с моделированием процессов обработки знаний, соответствующих различным способам мышления. Точные определения указанных операций и процессов, реализованные с использованием элементов логико-математического языка, образуют фундамент интеллектуальных систем и технологий. Основу классификации операций и предикатов моделей таких систем составляют когнитивные цели разных типов, моделирующие отдельные аспекты процесса мышления. Признанная система видов когнитивных целей включает: понимание, запоминание, применение, оценивание, анализ, синтез и обобщение [2]. Дополнительные операции и предикаты формализации (далее — формализмов) знаний позволяют моделировать процессы достижения когнитивных целей с помощью унифицированных структур информационных объектов. При этом понятие когнитивной цели в произвольном формализме знаний основано на инварианте алгебраической структуры (подструктуры) фрагмента знаний. Пусть $\Im = (M, D_M, \circ, \subseteq)$ — формализм представления знаний; V — перечислимый алфавит переменных; t — алгебраическая структура в формализме $\Im$. Если t это структура, то t является подструктурой структуры t. Если $t = (t_1 \circ t_2)$, то алгебраические структуры t_1 и t_2, а также подструктуры этих структур — это подструктуры структуры t.

Определение. Когнитивной целью в формализме $\Im$ называется пара $G = (t, P)$, где t — терм, составленный из элементов D_M и символов из V с помощью операции композиции, а P — вычислимый предикат для переменных в t.

Цель определяет структуру фрагмента знаний, представимого композицией подходящих знаний модели содержания предметной области, в котором символы переменных обозначают неизвестные компоненты такого фрагмента, и предикат P является истинным для значений переменных в таком знании. Унифицированное уточнение понятия когнитивной цели определяет инвариант элементов класса формализмов представления знаний, связанный с исследованием и применением универсального формата представления профессиональных задач, а также инструменты их решения, как способы реализации когнитивных целей. Процесс построения фрагмента знания, представляемого шаблоном, для которого оказывается истинным предикат P, называется синтезом. Этим процессом моделируется построение сложного знания из элементов разрешимого подмножества множества D_M. Использование унифицированного формата представления произвольных целей позволяет анализировать и комбинировать цели разных типов, если реализация одной цели может быть задана как последовательность реализаций других целей. Концепция формализма знаний как математического формализма делает возможным исследование процессов достижения целей с помощью точных инструментов. В частности, ими моделируются этапы процессов достижения целей. Каждой операции соответствует отображение $\mu : A \to B$, для которого A и B — это специальные подмножества D_M, называемые базами операции. Операции группируются в классы отображений с точно формулируемыми общими свойствами и допускающими формальное исследование, результаты которого переносятся в технологии работы со знаниями. Базы операций представляются перечислимыми множествами фрагментов знаний. Каждую базу A удобно рассматривать как множество значений некоторого вычислимого отображения $\mu : D_M \to A$. При этом само множество D_M также является базой всякого формализма знаний, порождаемой тождественным отображением.

Абстрактные пространства знаний

Абстрактное пространство знаний — это специальный класс формализмов представления знаний, оперирующий объектами иерархической структуры, называемыми конфигурациями [1]. Этот формализм является существенным расширением дескрипционных логик. Структуры образуют бинарные нагруженные деревья, листья которых соответствуют элементарным (неделимым) конфигурациям. Внутренние вершины деревьев размечаются отношениями, выполняющимися между конфигурациями, представленными левым и правым поддеревьями таких вершин. Определение абстрактного пространства знаний как формальной системы составляют перечислимые множества конфигураций M (содержащее пустую конфигурацию Λ) и бинарных отношений между конфигурациями R (включающее пустое отношение $\perp$), а также отображения разложения и связывания конфигураций $\varepsilon : M \to M \times M$ и $\psi : M_1 \to R$. Здесь $M_1 = \{z \mid \varepsilon(z) \neq (\Lambda, \Lambda)\}$ — множество неэлементарных конфигураций. На множестве элементарных конфигураций определено разрешимое отношение вложения содержания ρ_0. Всякая конфигурация $z \in M_1$ представляется тройкой (z_1, z_2, r), где $\varepsilon(z) = (z_1, z_2)$ и $\psi(z) = r$. Фрагменты конфигураций из $D_M \backslash M$ представляются неполными тройками, в которых отсутствует либо отношение, либо одна из конфигураций, а также парами отношений из R. Для R являются разрешимыми свойство вложения элементов этого семейства, обозначаемое как ρ_1, а также принадлежность произвольных пар конфигураций отношениям из R [1]. Отображения ε и ψ определяют структурные представления конфигураций нагруженными бинарными деревьями. Вершинами деревьев являются конечные двоичные наборы, так что наборами кодируются пути из корня в вершины и пустой набор является корнем дерева. Если $z \in M_1$, то корень дерева, представляющего z, размечен отношением $\psi(z)$, содержащим пару $\varepsilon(z) = (z_1, z_2)$. Левое и правое поддеревья корня представляют конфигурации z_1 и z_2. Множество вершин (висячих вершин) дерева, представляющего $z \in M_1$, обозначается как $D(z)$ ($O(z)$). Висячие вершины дерева размечены элементарными конфигурациями, а внутренние — отношениями, выполняющимися для конфигураций, представляемых левым и правым поддеревьями этих вершин. Если $\alpha \in D(z)$, то $[z]_\alpha$ обозначает разметку вершины α, $(z)_\alpha$ — фрагмент знания, представляемый поддеревом с корнем α. Операция композиции фрагментов конфигураций моделирует процесс конструирования представлений произвольных конфигураций снизу вверх. Композиция фрагментов конфигураций определяется с помощью следующей схемы:

$$z_1 \circ z_2 = \begin{cases} (z_1, z_2, \perp), & \text{если } z_1, z_2 \in M; \\ (z_1, \_, r_2), & \text{если } z_1 \in M \ \&\ z_2 = (r_1, r_2) \ \&\ [z_1]_\lambda = r_1; \\ (\_, z_1, r_1), & \text{если } z_1 \in M \ \&\ z_2 = (r_1, r_2) \ \&\ [z_1]_\lambda = r_2 \ \&\ [z_1]_\lambda \neq r_1; \\ z, & \text{если } z_1 = (z_3, \_, r) \ \&\ \varepsilon(z) = (z_3, z_2) \ \&\ (z_3, z_2) \in r; \\ z, & \text{если } z_2 = (\_, z_3, r) \ \&\ \varepsilon(z) = (z_1, z_3) \ \&\ (z_1, z_3) \in r; \\ z, & \text{если } z_1 = (z_3, \_, r) \ \&\ z_2 = (\_, z_4, r) \ \&\ \varepsilon(z) = (z_3, z_4) \ \&\ (z_3, z_4) \in r; \\ \Lambda, & \text{в остальных случаях.} \end{cases}$$

Для конструирования произвольной конфигурации по приведенным правилам достаточно построить пару конфигураций, составляющих ее разложение, добавить к одной из полученных конфигураций отношение, связывающее эти конфигурации, а затем подсоединить к полученному фрагменту вторую конфигурацию разложения. Отношение вложения фрагментов конфигураций основано на существовании изотонного отображения множеств вершин сравниваемых фрагментов конфигураций, переводящего внутренние (висячие) вершины представления одного фрагмента во внутренние (висячие) вершины другого фрагмента, для которого разметки сопоставляемых вершин оказываются сравнимы в отношении ρ_0 (ρ_1) [1]. Частные случаи вложения связаны с использованием специальных изотонных отображений, к которым относятся растяжения и сжатия. В формализме абстрактного пространства знаний трассирования являются универсальным инструментом для операций моделирования процессов извлечения знаний из знаний при моделировании процессов мышления.

Неделимыми (элементарными) знаниями всякого формализма абстрактного пространства знаний являются элементарные конфигурации и отношения из R. Символы переменных в термах представлений когнитивных целей соответствуют сущностям следующих типов: конфигурациям, фрагментам конфигураций, получаемым из конфигураций удалением одного из поддеревьев корня, а также отношениям между конфигурациями.

Будем использовать структурные представления конфигураций абстрактных пространств знаний для моделирования задач синтеза сложных знаний. Конструируемые структуры являются реализациями когнитивных целей. Классы таких структур составляют базы (области определения) операций, моделирующих реализации других когнитивных целей.

Формату представлений конфигураций абстрактного пространства знаний близки конструкты дескрипционных логик и языка RDF. Модели содержания конкретных областей знаний составляют системы неделимых (элементарных) знаний, образующие перечислимые множества конфигураций абстрактного пространства знаний. Эти множества структурируются в системы классов элементарных знаний, составляющих базисы операций над знаниями. Каждый элемент множества неэлементарных конфигураций $z \in M_1$ задается тройкой $(z_1, z_2, \psi(z))$, являющейся представлением отношения, выполняющегося между элементарными знаниями z_1 и z_2. Тройки группируются в отношения между классами знаний. Классы элементарных знаний и отношений между такими знаниями составляют модели содержания (онтологии) областей знаний. Каждое отдельное знание обозначает целостную семантическую сущность, содержание которой проявляется через многообразие отношений с другими знаниями. Содержание всякого сложного знания реализуется его полным структурным представлением. В последнем случае система операций над знаниями должна обеспечивать возможность конструирования семантических структур сложных знаний.

Формальная классификация когнитивных целей

Фундаментальная система базовых классов многообразия когнитивных целей призвана ограничить содержательные представления о целях разных типов требованием их полноты и независимости. Тогда остальные цели можно рассматривать как комбинации целей, принадлежащих заданным классам. Независимость целей из разных классов состоит в том, что цели любого такого класса можно реализовать без агрегирования процессов достижения целей других классов. Исходными данными для операций, реализующих такие цели, являются представления конкретных целей, множества знаний, составляющих модель содержания соответствующей области знаний, и фрагменты сложных знаний, принадлежащие базам операций. Для обозначения семейства формализованных знаний, соответствующего модели содержания рассматриваемой области деятельности, далее будем использовать символ Δ. Реализация когнитивной цели может не только основываться на Δ, но и изменять это семейство, являясь базой значений операции. Уточним описания перечисленных ранее типов когнитивных целей, основанных на классификации этапов процессов мышления [1], позволяющих считать систему целей содержательно полной, а сами цели из разных классов — независимыми. Эта классификация представлена иерархией классов целей, отражающих разные представления о назначении целей, приведенной на рис. 1. Она основана на общих классах целей селекции и трансформации. Целями селекции являются поиск и обратимая интеграция знаний в связные семантические структуры. Класс целей селекции обозначается как $\Upsilon(S)$. Цели второго типа соответствуют преобразованиям представлений сложных знаний, управляемым целями и содержанием таких знаний. Они изменяют структурные фрагменты обрабатываемых знаний на фрагменты, вычисляемые с помощью различных схем. Основными подклассами класса целей выбора являются классы целей понимания, запоминания, анализа и извлечения знаний.

Цель понимания состоит в установлении возможности представления заданного фрагмента знания (постановки цели) в виде комбинации элементов Δ. Это реализуется проверкой существования в Δ таких объектов, композицией которых является постановка цели, представляющая произвольный фрагмент знания в используемом формализме знаний. Класс таких целей будем обозначать как $\Upsilon(SU)$. Реализацией конкретной цели понимания g является структура фрагментов знаний, извлеченных из Δ. Многообразие целей данного класса определяют используемые форматы представления знаний, а также требования к элементам Δ, из которых составляются реализации целей понимания. Понимание может быть неполным,

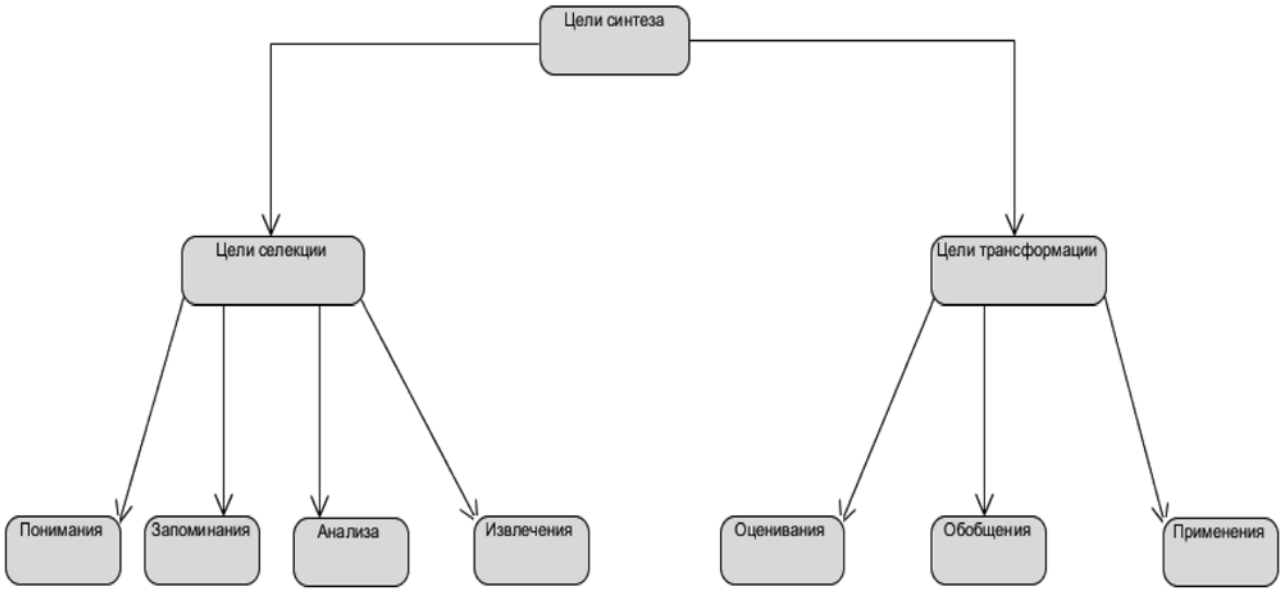


Рис. 1. Классы когнитивных целей формализованных моделей областей знаний

если распознанных элементов композиции недостаточно для законченного и целостного представления постановки цели из элементов Δ.

Постановками целей запоминания являются произвольные знания в используемых формализмах. Достижение цели запоминания реализуется через добавление к содержимому Δ новых фрагментов знаний, которые не могут быть составлены (выведены) из элементов Δ. В частности, если запоминание применяется к реализации цели понимания, то в Δ добавляются структурные фрагменты рассматриваемого знания, не вошедшие в семейство распознанных фрагментов при реализации понимания. Для обозначения класса целей запоминания будем использовать выражение ϒ(SR). Разнообразие целей запоминания связано с существованием нескольких атрибутов процессов формирования семейств извлекаемых новых знаний и встраивания таких знаний в конкретные системы Δ. Например, это могут быть отличающиеся по выразительным возможностям средства проверки выводимости фрагментов знаний из элементов Δ.

Назначением целей анализа является интеграция в единых структурах многообразий знаний, извлекаемых из Δ и используемых в последующем для решения различных задач. Такие структуры составляют одну из баз операций над знаниями. Интерпретации целей этого класса связаны со схемами обработки описаний отдельных целей совместно с моделью содержания области знаний. Реализация каждой цели анализа формирует структурное представление сложного знания, обратимое в семейство элементов множества Δ, использованных для его построения. Условие обратимости определяет уровень трансформаций знаний при реализации целей анализа, которые сводятся к отбору необходимых знаний, связываемых в единые семантические структуры. Проведение указанных преобразований включает распознавание знаний из Δ, из которых с использованием вспомогательных отношений конструируются интегрированные представления систем отобранных знаний. Цели рассматриваемого класса различаются детальностью и полнотой реализации отбора и связывания. Полная реализация цели анализа связана с распознаванием и интеграцией в составляемую структуру максимального множества знаний, связанных с представлением цели. Ограниченный анализ управляется параметрами глубины и ширины, задающими количественные ограничения на свойства отбираемых знаний и глубину конструируемых семантических представлений. Примером цели анализа является поиск, реализуемый построением неупорядоченной серии фрагментов знаний из Δ, обладающих заданным свойством. Класс таких целей обозначается как ϒ(SA).

Извлечение знаний из интеллектуальных ресурсов реализуют процессы нахождения фрагментов с требуемыми свойствами, включенными в заданные знания, принадлежащие специальным классам таких знаний. Из знаний начальных данных может извлекаться несколько знаний с заданными свойствами, представляемых совместно в виде серии. К важным содержательным аспектам целей извлечения знаний относится удаление избыточных или несущественных элементов обрабатываемых знаний. Подклассами класса целей трансформации знаний (обозначается как ϒ(T)) являются классы целей оценивания (ϒ(TE)), применения (ϒ(TA)) и обобщения (ϒ(TG)), реализуемые специальными видами изменения содержания структур обрабатываемых знаний, включающими вычислимые замены и перестановки элементов структурных представлений знаний.

Реализации целей оценивания связаны с моделированием преобразований структур знаний начальных

данных, фрагменты которых изменяют положение в этих структурах. Как сказано выше, класс целей оценивания обозначается как ϒ(ТЕ). Исходными данными таких целей являются фрагменты сложных знаний, представляющие серии фрагментов знаний, возможно дополненные моделями содержания областей знаний, перестраиваемые в структуры, отражающие результаты сравнений. Оценивание фрагментов знаний, содержащихся в начальных данных, может быть связано с вычислением значений атрибутов таких фрагментов, получаемых по содержанию моделей областей знаний. Такие значения определяют формат интеграции фрагментов знаний в структуры реализаций целей из рассматриваемого класса.

Содержанием целей применения и обобщения знаний являются трансформации знаний, составляющих начальные данные целей, в решения двух противоположных видов задач в моделируемых областях знаний. Примерами трансформаций, связанных с применением знаний, являются преобразования, моделирующие элементы вывода для исчисления предикатов, представленные конкретными схемами. Такие преобразования определяют фрагменты знаний, выводимые из серий фрагментов, представленных в знаниях начальных данных, с использованием подходящих схем. Выведенный фрагмент знания встраивается в обрабатываемое знание, а серия фрагментов, к которым применена схема вывода, может удаляться. Цели обобщения знаний связаны с преобразованием фрагментов представлений сложных знаний во фрагменты, имеющие не меньшее содержание. При обобщении возможно уменьшение избыточности и детальности содержания трансформируемых знаний. Частью класса целей применения знаний являются адаптации, реализующие схемы согласования фрагментов знаний начальных данных целей адаптации с конкретными ситуациями, представленными другими фрагментами начальных данных. Более сложными примерами применения знания являются реализации преобразований, использующих алгебраические и логические знания, интегрированные в начальное данное. В первом случае знание включает алгебраическое соотношение, а также значения параметров этого соотношения. Применение выражения состоит в замене этого выражения на его значение, вычисленное по значениям параметров. Полное алгебраическое применение знания состоит в замене алгебраических выражений на принимаемые ими значения везде, где это возможно. Цели логической трансформации моделируют элементы логического вывода на множествах фрагментов сложных знаний, в которых фрагменты знания начального данного составляют посылки схемы правила вывода. Логическое применение знания заключается в добавлении в обрабатываемое знание заключения применяемой схемы, вычисляемого с учетом ее интерпретации.

Приведенные содержательные определения классов когнитивных целей сужают представления об отдельных классах. Были выбраны такие аспекты классов, которые позволяют считать их независимыми, допускающими использование для конструирования сложных целей в виде комбинаций целей из рассмотренных классов. Примером комбинации когнитивных целей, составляющей сложную цель, является последовательность из элементов классов анализа, применения и извлечения. Указанные цели реализуют процессы отбора знаний, необходимых для достижения заданной цели, применения интегрированной системы отобранных знаний и извлечение результата. Последний получается из синтезированного знания выбором фрагмента, составляющего решение, и удалением избыточных фрагментов, не связанных с основной целью.

Регулярные структуры сложных знаний

Построение сложных знаний реализуется процессами включения в их алгебраические структуры новых элементов, выбираемых из семейства Δ. Регулярная алгебраическая структура таких знаний имеет вид иерархии серий близких по свойствам знаний, собираемой с использованием операций синтеза [3]. Многообразие видов серий определяется предположениями о дополнительных связях между их элементами. Если таких связей нет, то серия называется свободной. Всякая такая серия соответствует множеству независимых знаний, а алгебраически реализуется композицией, формирующей серию как последовательность ее элементов, обеспечивающую возможность доступа к ним. Для связывания элементов свободных серий в последовательности используется специальное вспомогательное отношение, обозначаемое как ω.

Если элементы серии связаны отношением ρ с некоторым элементарным знанием *a*, то они составляют окрестность радиуса один знания *a* в отношении ρ, обозначаемую как $O^1_\rho(a)$. Если при этом элементы серии не связаны между собой, то окрестность называется свободной окрестностью *a* в ρ. Порядок следования элементов представлений свободных серий и окрестностей элементов в отношениях не существенен. Регулярные расширения свободных окрестностей связаны с заменой элементов из таких окрестностей на окрестности, соответствующие этим элементам в некоторых отношениях. При этом образуются окрестности радиуса два. Допускается частичное расширение окрестностей, определяемое условием на такие элементы, для которых выполняется расширение. Многократное расширение начальной окрестности позволяет определить последовательность окрестностей элемента *a*. При этом если расширение выполняется для всех элементов окрестностей и при расширениях используется только одно отношение ρ, то свободная окрестность *a* в отношении ρ радиуса *k* обозначается как $O^k_\rho(a)$.

Если на множестве элементов $O^1_\rho(a)$ определено отношение линейного порядка (эквивалентности), то окрестность называется линейно упорядоченной (разбиением). Примерами других видов серий и используемых для этого вспомогательных отношений

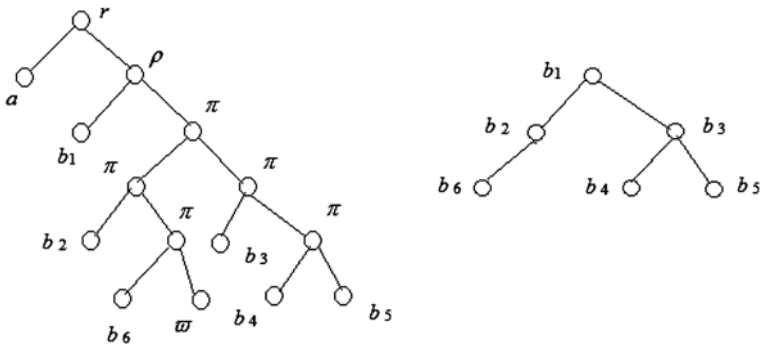


Рис. 2. Дерево окрестности элемента *a* в отношении *r*

являются иерархии, представляемые деревьями. Для представления бинарного дерева, вершины которого соответствуют элементам серии, определим вспомогательное отношение π. Пример представления окрестности *a* в отношении ρ, реализующей структуру бинарного дерева, приведен на рис. 2.

В левой части рис. 2 в формате конфигурации абстрактного пространства знаний *z* изображено представление окрестности элемента *a* в отношении *r*. В отношении ρ элементы этой окрестности составляют иерархию, представленную бинарным деревом, изображенным в правой части рисунка. Отношение ρ, связывающее элементы окрестности в иерархию, приписано корню правого поддерева представления *z*. Представляемое поддеревом семейство элементов окрестности *a* в отношении ρ приписано висячим вершинам этого поддерева. Внутренние вершины поддерева размечены отношением π, а корень — отношением ρ. Разметка корня иерархии элементов размещается в вершине 10, т. е. $[z]_{10} = b_1$. Правое и левое поддеревья представления *z* с корнем 11 представляют левое и правое поддеревья моделируемой иерархии элементов окрестности с корнями b_2 и b_3.

Аналогичные правила можно определить для окрестностей и серий, структурированных другими видами отношений.

Базовые операции когнитивного синтеза

Унификация понятий, связанных с операциями, реализующими достижение произвольных когнитивных целей, предполагает определение семейства классов отображений, комбинации элементов которых моделируют этапы процессов реализации целей из рассмотренных классов. Формальные уточнения классов операций составляют логико-математические описания требований к их свойствам, включающие форматы элементов областей определения и значения (баз) операций.

Рассмотрим систему классов алгебраических операций, используя базы, основанные на конструктах алгебраической структуры знаний, серий и окрестностей знаний. Из операций этих классов конструируются процессы достижения когнитивных целей. В основе классификации лежат аналогии с теоретико-множественными, структурирующими, адаптирующими и топологическими операциями в математических системах. Области определения и значения (базы) класса теоретико-множественных операций составляют модель области знаний и (или) множества конфигураций абстрактного пространства знаний, которыми представляются семейства конечных множеств знаний. Операциями структуризации реализуются преобразования конфигураций начальных данных в конфигурации, которые расширяют или перестраивают содержание таких конфигураций. Операции адаптации моделируют отображения конфигураций в конфигурации, основанные на применении содержания фрагментов обрабатываемых знаний. Топологическими операциями моделируются вычисления, результаты которых соответствуют топологическим инвариантам и конструктам для элементов баз таких операций. Фрагмент унифицированного классификатора алгебраических операций над формализованным содержанием областей знаний, адаптирующих систему классов морфизмов в абстрактных пространствах знаний для задачи достижения когнитивных целей, приведен на рис. 3.

Рассмотрим точные определения классов операций приведенной иерархии, представленные с использованием логико-математических конструкций. Области определения и значения рассматриваемых далее теоретико-множественных операций составляет семейство свободных серий *S*. Определения операций могут быть обобщены на множества произвольных серий и окрестностей отдельных знаний в произвольных отношениях, а также на модели содержания областей знаний Δ. Для конструирования конфигураций, составляющих наборы множеств (свободных серий или окрестностей), применим операцию композиции фрагментов конфигураций абстрактного пространства знаний. Класс теоретико-множественных операций составляют подклассы булевских операций и операций фильтрации. Булевскими операциями моделируются аналоги операций над наборами множеств, композиции которых составляют конфигурации начальных данных. В первую очередь это относится к операциям объединения, пересечения, разности и произведения множеств. Приводимые далее определения используют конструкцию вложения множеств конфигураций, основанную на понятиях трассирования и вложения конфигураций [1]. Если z_1 и z_2 — конфигурации из $S \cup O$, представляющие множества конфигураций $S(z_1)$ и $S(z_2)$, то вложение $S(z_1) \subseteq_I S(z_2)$ означает, что $\forall z' \in S(z_1) \exists z'' \in S(z_1)(z' \subseteq_I z'')$. Здесь $\subseteq_I$ — обозначение отношения вложения конфигураций, где $I \in \{p, c, o, k\}$ обозначает тип используемых отображений трассирования (растяжения, сжатия, общего вида и окончаний) [1]. Обозначим как $S \circ S$ множество композиций серий $\{z_1 \circ z_2 \mid z_1, z_2 \in S\}$.

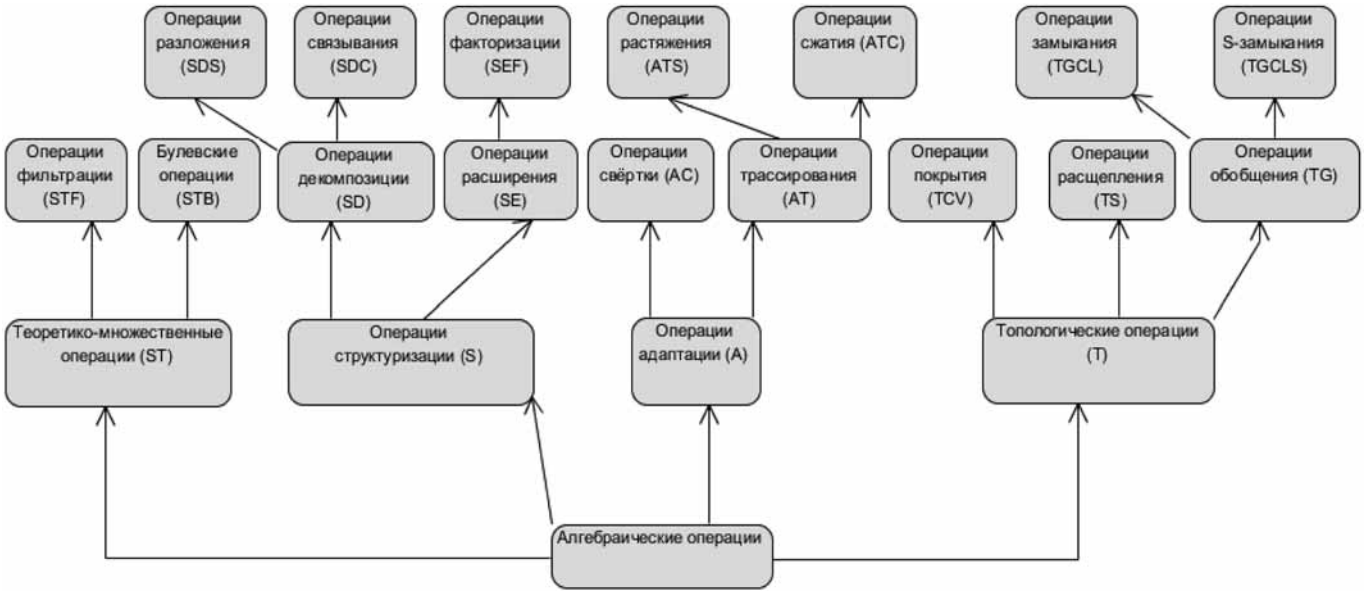


Рис. 3. Классы алгебраических операций моделирования когнитивных целей

Определим классы операций семантического пересечения, объединения и разности множеств (свободных серий) конфигураций.

Определение. Вычислимое отображение $\mu : S \circ S \to S$ называется семантическим объединением, если для любых $z_1, z_2 \in S$ выполняются условия

- $S(z_1) \cup S(z_2) \subseteq_I S(\mu(z_1 \circ z_2))$;
- $\forall \{z\} \in M\left(z \in S(\mu(z_1 \circ z_2)) \to \right.$
  $\left. \to \{z\} \subseteq_I S(z_1) \vee \{z\} \subseteq_I S(z_2)\right)$.

Определение. Вычислимое отображение $\mu : S \circ S \to S$ называется семантическим пересечением, если для любых $z_1, z_2 \in S$ выполняются условия

- $S(z_1) \cap S(z_2) \subseteq_I S(\mu(z_1 \circ z_2))$;
- $\forall z \in M\left(z \in S(\mu(z_1 \circ z_2)) \to \right.$
  $\left. \{z\} \subseteq_I S(z_1) \,\&\, \{z\} \subseteq_I S(z_2)\right)$.

Определение. Вычислимое отображение $\mu : S \circ S \to S$ называется семантической разностью, если для любых $z_1, z_2 \in S$ выполняются условия

- $S(\mu(z_1 \circ z_2)) \subseteq_I S(z_1)$;
- $\forall z \in S(\mu(z_1 \circ z_2))(\{z\} \ \overline{\subseteq}_I S(z_2))$;
- $\forall z \in S(z_1)(\{z\} \overline{\subseteq}_I S(z_2) \to z \in S(\mu(z_1 \circ z_2)))$.

Операции приведенных классов моделируют аналоги соответствующих теоретико-множественных операций с точностью до вложения множеств, представляемых фрагментами конфигураций. Операциями реализуются соответственно интеграция всех, сохранение общих и распознавание уникальных знаний, содержащихся в знаниях из пар заданных множеств. Результатом всякой операции семантического объединения является множество конфигураций, в которое вложены объединяемые множества и всякий элемент семантического объединения вложен в одно из этих множеств. Результатом операции семантического пересечения множеств конфигураций является множество, в которое вложено пересечение множеств начальных данных и каждая конфигурация семантического пересечения вложена в каждое из множеств начальных данных. Семантическую разность двух множеств составляют конфигурации, вложенные в первое и не вложенные во второе множество начального данного каждой такой операции. Каждая конфигурация первого множества, которая семантически не вкладывается во второе множество, принадлежит семантической разности начальных данных.

Определение. Произведением называется всякое вычислимое отображение $\mu : S \circ S \to S$, для которого

- $\forall z_1, z_2 \in S(\forall z' \in S(z_1),$
  $z'' \in S(z_2) \exists z \in S(\mu(z_1 \circ z_2))(z' \subseteq_I z \,\&\, z'' \subseteq_I z))$;
- $\forall z_1, z_2 \in S((\forall z \in S(\mu(z_1 \circ z_2)) \exists z' \in S(z_1),$
  $z'' \in S(z_2)(z' \subseteq_I z \,\&\, z'' \subseteq_I z))$.

Фильтрации составляют специальный класс теоретико-множественных операций, реализующих поэлементное построение подмножеств множеств конфигураций, являющихся начальными данными таких операций. Примером произведения является операция, конструирующая серию, элементами которой являются все прямые суммы пар конфигураций из множеств начальных данных [1].

Определение. Вычислимое отображение $\mu : S \to S$ называется семантическим фильтром, если

- $\forall z \in S(S(\mu(z)) \subseteq S(z))$;

- $\forall z_1,\ z_2 \in S \forall z \in M \left(z \in S\left(\mu\left(S\left(z_1\right)\right)\right) \& S\left(z_1\right) \subseteq \right.$

$$\left. \subseteq S\left(z_2\right) \rightarrow z \in S\left(\mu\left(S\left(z_2\right)\right)\right)\right).$$

Фильтрами реализуются простые схемы извлечения подмножеств произвольных семейств конфигураций. Каждая такая схема реализует одноместный предикат, истинность которого для произвольной конфигурации из области определения этой операции означает, что эта конфигурация включается во множество (серию), являющееся значением операции на этом начальном данном. Аналогично можно определить понятие фильтра модели содержания области знаний $\mu : \Delta \rightarrow S$, извлекающего требуемые множества знаний из элементов моделей.

Класс операций структуризации составляют преобразования трансформации структур фрагментов знаний. Они расширяют и перестраивают структуры конфигураций начальных данных включением новых фрагментов конфигураций и семантических отношений, а также изменением порядка фрагментов таких структур. Добавляемые новые элементы выявляются с помощью операций данного класса, извлекающих их из Δ. Добавление новых элементов реализуется через изменения алгебраических структур конфигураций, пополняемых новыми фрагментами. Такие изменения могут различаться, что позволяет определить несколько важных видов операций структуризации. Примерами таких классов операций являются операции декомпозиции, расширения, факторизации и перестановки. Из них операции декомпозиции реализуют изменения алгебраических структур фрагментов знаний, в которых происходит замена отдельных элементарных знаний на алгебраические структуры фрагментов знаний. Операциями расширения реализуются встраивания в алгебраические структуры конфигураций структур добавляемых фрагментов знаний. Факторизации являются частным случаем класса расширений. Они моделируют схемы группирования фрагментов знаний в серии, дополняемые элементарными знаниями, обозначающими такие серии. Перестановки — это операции, реализующие перестановки поддеревьев алгебраических структур конфигураций.

Рассмотрим уточняющие определения операций перечисленных типов, в которых использованы специальные выражения, связанные с алгебраическими структурами конфигураций. Если $z \in M$ — произвольная конфигурация, то выражение $\Sigma(z)$ обозначает алгебраическую структуру z. Множество всех (висячих) вершин $\Sigma(z)$ обозначается как $D(z)$ ($O(z)$). Разметка каждой такой вершины $\alpha \in D(z)$ обозначается как $[\Sigma(z)]_\alpha$.

Определение. Вычислимое отображение $\mu : M \rightarrow M$ называется элементарной декомпозицией, если

$$\forall z \in M \exists z' \in M \exists \alpha \in O(z)\left(\Sigma\left(\mu\left(z\right)\right)\right)_\alpha =$$

$$= \Sigma\left(z'\right) \& \ \forall \beta \in D(z) \backslash \{\alpha\}\left(\left[\Sigma(z)\right]_\beta = \left[\Sigma\left(\mu\left(z\right)\right)\right]_\beta\right).$$

Определение. Отображение $\mu : M \rightarrow M$ называется декомпозицией, если для любой $z \in M$ конфигурация $\mu(z)$ получается из z с помощью конечной последовательности применений операции элементарной декомпозиции.

Декомпозиции — это замены семейств элементарных конфигураций в алгебраических структурах конфигураций начальных данных на алгебраические структуры конфигураций. Эти операции позволяют включать в структуры знаний фрагменты, реализующие расширенные описания заменяемых знаний, необходимые для достижения моделируемых целей. Взаимно дополняющими случаями декомпозиции являются операции разложения и связывания, составляющие определения понятия абстрактного пространства знаний [1]. Операции первого типа определяют совокупности элементарных знаний в структурах, заменяющих элементарные конфигурации, удаляемые из алгебраических структур конфигураций начальных данных. Операциями второго типа реализуются допустимые изменения семантических отношений в висячих вершинах алгебраических структур конфигураций.

Еще один класс операций структуризации составляют расширения, основанные на операциях вставки конфигураций в конфигурации. Пусть $z_1, z_2, z_3 \in M$. Конфигурация z_3 получается из z_1 и z_2 с помощью $\beta\sigma$ — вставки конфигурации z_2 ($\beta \in D(z)$, $\sigma \in \{0,1\}$) в конфигурацию z_1 (обозначается как $z_3 = z_1 \oplus_{\beta\sigma} z_2$), если

$$\Sigma\left(z_3\right) =$$

$$= \begin{cases} \left(\ldots\left(\left(z_1\right)_\alpha \circ z_2\right)\ldots\right), & \text{если } z_1 = \left(\ldots\left(z_1\right)_\alpha \ldots\right) \& \ \sigma = 0, \\ \left(\ldots\left(z_2 \circ \left(z_1\right)_\alpha\right)\ldots\right), & \text{если } z_1 = \left(\ldots\left(z_1\right)_\alpha \ldots\right) \& \ \sigma = 1. \end{cases}$$

Определение. Вычислимое отображение $\mu : M \rightarrow M$ называется элементарным расширением конфигураций, если

$$\forall z \in M \exists z' \in M \exists \alpha \in D(z)\left(\mu(z) = z \oplus_{\beta\sigma} z'\right).$$

Определение. Вычислимое отображение $\mu : M \rightarrow M$ называется расширением конфигураций, если для любой $z \in M$ конфигурация $\mu(z)$ получается из z с помощью конечной последовательности применений операции вставки конфигураций.

Частным случаем отображений расширения являются факторизации, для которых выполняется соотношение $\forall z \in M \exists z' \in M\left(\mu(z) = z \oplus_0 z'\right)$. Отображениями факторизации моделируются распределение конфигураций по классам, ставящие в соответствие отдельным конфигурациям значение $(\mu(z))_1$, определяющее класс принадлежности конфигурации z.

Определение. Вычислимое отображение $\mu : M \rightarrow M$ называется элементарной перестановкой, если

$$\forall z \in M \exists \alpha,\ \beta \in D(z)\left(\left(\mu\left(z\right)\right)_\beta = (z)_\alpha \ \& \ \left(\mu\left(z\right)\right)_\alpha =\right.$$

$$\left. = (z)_\beta \ \& \ \forall \gamma \in D(z)\left(\alpha \not\subset \gamma \ \& \ \beta \not\subset \gamma \rightarrow \left[\mu(z)\right]_\gamma = \left[z\right]_\gamma\right)\right).$$

Определение. Вычислимое отображение $\mu : M \to M$ называется перестановкой, если для любой $z \in M$ конфигурация $\mu(z)$ получается из z с помощью конечной последовательности применений операции элементарной перестановки.

Содержание адаптации состоит в согласовании и конкретизации систем фрагментов обрабатываемых знаний. Трансформации конфигураций, реализуемые операциями данного класса, состоят в неинъективном изменении форм представления знаний, связанным с уменьшением содержания изменяемых фрагментов. Семейство операций адаптации составляют подклассы операций свертки и трассирования. Отображения первого класса реализуют удаление или замену фрагментов знаний на объекты, представляющие элементарные знания. Указанные виды операций свертки будем обозначать как $\Psi(z, \alpha)$ (удаление фрагмента алгебраической структуры z с корнем α) и $\Phi(z, \alpha, c)$ (замена фрагмента алгебраической структуры z с корнем α на c). Они аналогичны операциям, обратным к операциям расширения и декомпозиции. Этими операциями моделируются преобразования замены фрагментов знаний, представляющих алгебраические выражения или схемы логического вывода, на результаты их применения. Операции трассирования моделируют схемы извлечения знаний из знаний. Они основаны на изотонных отображениях множеств вершин конфигураций результатов операций во множества вершин конфигураций начальных данных таких операций. Значениями операций трассирования являются конфигурации, разметки вершин которых составляются из разметок вершин конфигураций начальных данных, являющихся образами изотонных отображений [1]. Для формализма абстрактных пространств знаний такие отображения конфигураций в конфигурации называются эндоморфизмами конфигураций [1]. По типам используемых изотонных отображений они относятся к эндоморфизмам растяжения, сжатия и общего вида. Пусть I — множество двоичных наборов, представляющих вершины конфигураций. Если $z \in M$ и $\xi : I \to I$ — изотонное отображение, то извлекаемая в соответствии с типом этого отображения конфигурация обозначается как $T(z, \xi)$. Множество всех изотонных отображений $\xi : I \to I$ будем обозначать как Ξ.

Определение. Вычислимое отображение $\mu : M \to M$ называется сверткой, если $\forall z \in M\ \exists \alpha \in D(z) \times \times \left(\exists\, c \in M\left(\mu(z) = \Phi(z, \alpha, c)\right) \vee \mu(z) = \Psi(z, \alpha)\right)$.

Определение. Вычислимое отображение $\mu : M \to M$ называется трассированием, если $\forall z \in M\ \exists \xi \in \Xi\ \left(\mu(z) = T(z, \xi)\right)$.

Основные подклассы класса топологических операций над знаниями составляют операции покрытия, расщепления и замыкания, подобные аналогично именуемым операциям в разных областях математики. С помощью операций покрытия из элементов серий конфигураций и семантических отношений составляются конфигурации, интегрирующие элементы серий. Всякая операция расщепления преобразует конфигурации начальных данных в серии подконфигураций и семантических отношений, композициями которых являются эти конфигурации. Операции замыкания моделируют схемы обобщения знаний.

Базы топологических операций разных типов составляют множества конфигураций M, серий конфигураций S, серий семантических отношений SR.

Определение. Вычислимое отображение $\mu : S \to M$ называется покрытием, если для любых $z \in S$ и $\alpha \in D\left(\Sigma\left(\mu(z)\right)\right)$ выполняется одно из соотношений

- $[\mu(z)]_\alpha = \circ$;
- $\exists r \in SR\left([\mu(z)]_\alpha \in r\right)$;
- $\exists z'' \in S(z)\, \exists \beta \in D(z'')\left(\mu(z)_\alpha = (z'')_\beta\right)$.

Определение покрытия означает, что алгебраическая структура конфигурации $\mu(z)$ может быть составлена как композиция элементов серии $z \in S$ с использованием подходящих семантических отношений.

Определение. Вычислимое отображение $\mu : M \to S$ называется расщеплением, если для любых $z \in M$ и $\alpha \in D(z)$ выполняется одно из соотношений

- $[z]_\alpha = \circ$;
- $\exists r \in SR\left([z]_\alpha \in r\right)$;
- $\exists z' \in S\left(\mu(z)\right) \exists \beta \in D(z')\left(\mu(z)_\alpha = (z')_\beta\right)$.

Расщепления моделируют преобразования, обратные операциям покрытия, заменяя конфигурации на серии конфигураций, из которых с помощью композиции и семантического связывания можно составить исходные конфигурации.

Определение. Вычислимое отображение $\mu : S \to M$ называется обобщением, если $\forall z \in S \forall z' \in \in S(z)\left(z' \subseteq_I \mu(z)\right)$.

Определение. Вычислимое отображение $\mu : S \to M$ называется замыканием, если для любой $z \in S$

- $S(z) \subseteq_I \{\mu(z)\}$;
- $\forall z' \in M\left(z' \subseteq_I \mu(z)\ \&\ \mu(z) \ \supseteq_I z' \to \right.$
  $\left. \to \exists z'' \in S(z)\left(z'' \supseteq_I z'\right)\right)$.

Замыкание произвольных множеств конфигураций моделирует построение минимальных в отношении $\subseteq_I$ конфигураций, в которые вкладываются все конфигурации заданных семейств. Для общего случая формализмов представления знаний отображения замыкания могут оказаться невычислимыми. Поэтому оправдано применение специальных схем обобщения, содержащих ограничения на фрагменты конфигураций, из которых составляются значения отображений замыкания. Например, это может быть условие конструирования обобщений только из элементов конфигураций начальных данных.

Определение. Вычислимое отображение $\mu : S \to M$ называется *s*-замыканием, если для любой $z \in S$

- $S(z) \subseteq_I \{\mu(z)\}$;
- $\forall z' \in M\,(z' \subseteq_I \mu(z)\ \&\ \mu(z) \supseteq_I z' \to$
  $\to \exists z'' \in S(z)(z'' \supseteq_I z'))$;
- $\forall \alpha \in D(\mu(z))\big(([\mu(z)]_\alpha = \circ) \vee \exists z' \in$
  $\in \Sigma(\mu(z))\exists \beta \in D(z')(\mu(z))_\alpha = (z')_\beta\big)$.

Последнее определение реализует специальное уточнение понятия замыкания множества конфигураций как конфигурации, в которую вкладывается каждая конфигурация из заданного множества, составленная только из фрагментов элементов этого множества.

Реализации сложных когнитивных целей

Для именования классов построенной иерархии и ее возможных расширений будем использовать составные имена, являющиеся расширениями имен классов предков. При этом имена подбираются так, чтобы расширения имен классов при переходе к вершинам-потомкам сохраняли свойство префиксности, когда имена несравнимых вершин иерархии классов не являются началами друг друга.

Классы теоретико-множественных, структурирующих, адаптирующих и топологических операций получают имена ST, S, A и TY. Подклассам класса ST соответствуют имена STF — для операций фильтрации и STB — для булевских операций. Имена других рассмотренных классов операций приведены на рис. 3.

Когнитивные цели разных типов реализуются комбинированием операций над формализованными знаниями. Всякая комбинация операций может быть уточнена схемой, составленной классами операций и порядком их применения. Сама схема составляется с использованием нескольких типов конструктов. Примерами таких конструктов являются двуместные композиция и альтернирование, а также одноместная итерация, обозначаемые символами $\circ$, | и *, соответственно.

Заключение

Фундаментом модели содержания понятия системы искусственного интеллекта является семейство точных инвариантов для структурных и функциональных элементов такой системы. Определения инвариантов, уточняемые с использованием элементов логико-математического языка, делают их близкими сущностям из таких разделов математики, как теория множеств, общая алгебра, логика, топология. Теоретическое исследование отобранных инвариантов обеспечивает лучшее понимание возможностей интеллектуальных систем, способствует формированию системы порождающих принципов для процессов построения и использования таких систем. При этом только математических аспектов недостаточно для полного отражения содержания всевозможных представлений об интеллектуальных системах. Их дополняют атрибуты, связанные с моделированием процессов мышления, отражающие философские (гносеология и онтология), лингвистические и психологические аспекты таких процессов. Согласование сущностей разных областей знаний является необходимым условием для построения продуктивной и сбалансированной теории систем искусственного интеллекта.

Работа выполнена при поддержке РФФИ, проект № 16-01-00214.

Список литературы

1. **Костенко К. И.** Формализмы представления знаний и модели интеллектуальных систем. Краснодар: Кубанский гос. ун-т, 2015. 300 с.

2. **Bloom B. S. (Ed.), Engelhart M. D., Furst E. J., Hill W. H., Krathwohl D. R.** Taxonomy of educational objectives: The classification Taxonomy of educational goals. Handbook 1: Cognitive domain. New York: David McKay, 1956.

3. **Костенко К. И.** О синтезе реализаций когнитивных целей для задачи управления содержанием областей знаний // Программная инженерия. 2017. Т. 8, № 7. С. 319—327.

Operations of Formalized Knowledge Cognitive Synthesis

K. I. Kostenko, kostenko@kubsu.ru, Kuban State University, Krasnodar, 350040, Russian Federation

Corresponding author:

Kostenko Konstantin I., Assistant Professor, Kuban State University, Krasnodar, 350040, Russian Federation, E-mail: kostenko@kubsu.ru

Received on November 21, 2017
Accepted on December 21, 2017

The cognitive goals classification problem relates to constructing the substantially full system of independent classes of such goals correlated with various philosophical, linguistic and psychological concepts and models. The formalized and unified systems of universal basic qualifiers of cognitive goals and knowledge synthesis operations,

agreed with concept of knowledge representation formalisms, are considered. It allows presenting of any cognitive goal as combinations of the goals from basic classes and defining complex knowledge synthesis processes. The created classifications are based on specifications of weakly formalized and ambiguous concepts of cognitive goal and cognitive operation, offered by B. Bloom.

Such a classification is broad, tangled and ambiguous. Realization of goal from any offered class often suppose realization of goals from other classes and needs accommodation of a hardly realized multiplicity of relations between goals and operations.

Coordination of the created classification with knowledge representation formalisms invariants is realized by such a compression of cognitive goal and operation concepts that imply independence of goals and operations of different classes.

Relation of aggregation does not connect the elements of defined classes. It allows considering the simulated classes of cognitive goals in the narrow sense bounded by specific formalized requirements to their general properties and realization, own for each such class. The universal hierarchy of formalized classes of computable knowledge processing operations is unique for all knowledge representation formalisms. It is adapted to hierarchy of cognitive goals classes and is based on set-theoretic, algebraic, logical and topological aspects of such operations.

Transforming the mathematical concepts into the functional invariants for cognitive goals realizations for any knowledge representation formalism is coordinated with the universal invariants of knowledge algebraic and semantic structures.

The constructed versions of qualifiers expand theoretical and application-oriented opportunities for the concept of knowledge representation formalisms, keeping its generality, abstractness and universality.

Keywords: *knowledge representation formalism, algebraic structure, task, knowledge processing, cognitive goal, homomorphic extension, knowledge synthesis, ontology*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project nos. 16-01-00214*

For citation:

Kostenko K. I. Operations of Formalized Knowledge Cognitive Synthesis, *Programmnaya Ingeneria,* 2018, vol. 9, no. 4, pp. 174—184.

DOI: 10.17587/prin.9.174-184

References

1. **Kostenko K. I.** *Formalizmy predstavlenija znanij i modeli intellektualnyh sistem* (Knowledge representation formalisms and intelligent systems models), Krasnodar, Kuban state university, 2015, 300 p. (in Russian).

2. **Bloom B. S. (Ed.), Engelhart M. D., Furst E. J., Hill W. H., Krathwohl D. R.** *Taxonomy of educational objectives: The classification Taxonomy of educational goals.* Handbook 1: Cognitive domain, New York, David McKay, 1956.

3. **Kostenko K. I.** O sinteze realizacij kognitivnyh celej dlja zadachi upravlenija soderjaniem oblastei znanij (The Synthesis of Cognitive Goals Implementation for Tasks of Subject Domains Content Management), *Programmnaya Ingeneria*, 2017, vol. 8, no. 7, pp. 424—431 (in Russian).

ИНФОРМАЦИЯ

Начинается подписка на журнал "Программная инженерия" на второе полугодие 2018 г.

Оформить подписку можно через подписные агентства или непосредственно в редакции журнала.

Подписные индексы по каталогам:

Роспечать — 22765; Пресса России — 39795

Адрес редакции: 107076, Москва, Стромынский пер., д. 4,
Издательство "Новые технологии",
редакция журнала "Программная инженерия"

Тел.: (499) 269-53-97. Факс: (499) 269-55-10. E-mail: prin@novtex.ru

УДК 004.5; 004.94 DOI: 10.17587/prin.9.185-190

Я. А. Туровский, канд. мед. наук, доц., e-mail: yaroslav_turovsk@mail.ru, Воронежский государственный университет, **А. А. Адаменко,** аспирант, e-mail: adamenko.artem@gmail.com, Воронежский государственный университет инженерных технологий

Сравнительный анализ эволюционного метода с использованием "изолятов" и метода имитации отжига при обучении искусственных нейронных сетей

Проведена серия вычислительных экспериментов по обучению искусственных нейронных сетей с применением эволюционного метода их обучения с использованием "изолятов" и метода имитации отжига. Результаты сравнительного анализа этих экспериментов показали возможность применения разработанного программного пакета ANNBuilder для моделирования обучения нейрочипа в целях восстановления поврежденной нервной ткани. В первой группе этих экспериментов использовался простой эволюционный алгоритм обучения искусственных нейронных сетей с применением механизма "изоляции". Во второй группе экспериментов использовался метод имитации отжига. По результатам анализа сделано заключение, что при использовании эволюционного метода обучения искусственных нейронных сетей с использованием механизма "изоляции" по сравнению со второй группой экспериментов, в большинстве случаев достигается лучшее значение функции ошибки искусственных нейронных сетей. Это обстоятельство оправдывает приоритет использования первого метода по сравнению со вторым при моделировании обучения нейрочипа.

Ключевые слова: *нейрочип, эволюционный алгоритм, метод имитации отжига, изоляция, изолят, искусственные нейронные сети*

Введение

В настоящее время актуальным является создание методов восстановления поврежденной по той или иной причине нервной ткани. Ряд перспективных методов восстановления функций мозга основан на внедрении в нервную ткань устройства входа/выхода. Такое специальное устройство называется нейрочипом (НЧ) и оно подключается непосредственно к нервной ткани в целях получения возможности регистрации активности микросистемы нейронов этой ткани. Нейрочип осуществляет обработку поступающих из ткани сигналов и передает их в компьютер для формирования команд на изменение активности микросистемы нейронов. Это делается для того, чтобы скорректировать изменения в процессе функционирования микросистемы нейронов, связанные с утратой части функций [1—6]. После интеграции НЧ в нервную ткань требуется провести его обучение, т. е. осуществить подстройку под ту микрообласть нервной ткани, в которую он был вживлен. Один из вариантов обучения НЧ основывается на использовании хорошо зарекомендовавших себя в области нейропротезирования [7, 8] искусственных нейронных сетей (ИНС). Среди существующих методов обучения ИНС наибольшую популярность в настоящее время имеют методы глубинного обучения. При этом реализованная тем или иным способом архитектура НЧ должна будет обеспечивать решение тех задач, которые выполнял поврежденный участок мозга.

В методах глубинного обучения ИНС большую роль играют алгоритмы обучения ИНС. В настоящее время существует несколько хорошо зарекомендовавших себя для различных типов задач алгоритмов обучения глубинных ИНС, а именно стохастический градиентный спуск [9]; метод сопряженных градиентов [10]; метод имитации отжига [9, 11].

Метод имитации отжига является одним из наиболее популярных методов глубинного обучения, который способен обходить локальные минимумы функции ошибки обучения ИНС с помощью вероятностного подхода к выбору направления движения функции ошибки. Реализованный в программном пакете *ANNBuilder* эволюционный алгоритм, так же как и метод имитации отжига использует в первом приближении, вероятностный подход. Кроме этого, он использует механизм "мутации", который с определенной осторожностью можно сравнить с механизмом скачка температуры. Для эволюционного алгоритма разработан и реализован механизм "изоляции" [12]. Данный механизм во многих случаях

помогает найти направление движения функции ошибки в сторону глобального минимума пространства ошибок ИНС и выйти из локального минимума.

Целью данной работы является сравнение метода имитации отжига и эволюционного алгоритма с применением механизма "изоляции" в обучении ИНС для задач обработки сигналов.

Описание используемых алгоритмов

В рамках моделирования обучения НЧ, внедренного в нервную ткань в целях восстановления ее функций, использовался разработанный авторами программный пакет *ANNBuilder* [13—15]. В основу алгоритма моделирования обучения НЧ положен эволюционный алгоритм. Его суть в том, что ИНС "скрещиваются" между собой, порождая популяции ИНС. Популяции, в свою очередь, проходят этап отбора, оставляя для скрещивания в следующей популяции ИНС с лучшим показателем функции ошибки. В большинстве случаев такой алгоритм является наиболее подходящим. Его удобно применять в тех ситуациях, когда в силу особенностей работы НЧ для него неизвестны входные и выходные сигналы, генерируемые нервной тканью. Это обстоятельство связано с недостаточной изученностью особенностей функционирования микросистем нейронов мозга человека. Проведенные авторами вычислительные эксперименты доказали способность разработанного программного пакета *ANNBuilder* осуществить необходимое моделирование обучения НЧ [16, 17]. В целях усовершенствования поиска локального минимума функции ошибки эволюционного алгоритма обучения НЧ использовано понятие "изолят". В основу понятия "изолят" положен эффект биологической или социальной изоляции, заключающийся в том, что объекты скрещиваются только внутри изолированной географической или социальной группы.

Для программного пакета *ANNBuilder* реализован алгоритм метода имитации отжига в целях проведения серии экспериментов по обучению ИНС и сравнения этих результатов с результатами обучения ИНС с помощью эволюционного алгоритма. Метод имитации отжига — это техника оптимизации, использующая упорядоченный случайный поиск на основе аналогии с процессом образования в веществе кристаллической структуры с минимальной энергией при охлаждении [11].

Опишем реализованный алгоритм метода имитации отжига [9, 11]. Пусть S — множество всех состояний весовых коэффициентов (ВК) ИНС в пространстве значений ВК. Пусть S_i — состояние на i-м шаге алгоритма; $S_i \in S$; $t_i \in R$ — "температура" на i-м шаге. Для того чтобы использовать имитацию отжига, необходимо определить следующие функции:

- функцию энергии, в нашем случае это функция ошибки ИНС — $E = f(S_i)$;
- функцию изменения "температуры" на i-м шаге — $T = f(i)$;
- функцию, порождающую новое состояние — $G = f(S_i, t_i)$;
- вероятность принятия нового состояния — $h = f(\Delta E, t_i)$, где i — номер шага.

В программном пакете *ANNBuilder* для уменьшения значения "температуры" на каждой i-й итерации в методе имитации отжига использовалась следующая формула:

$$t_i = \frac{t_1 0{,}1}{i},$$

где t_1 — начальная "температура", заданная пользователем; i — номер итерации.

Алгоритм при этом содержит следующую последовательность действий:

1. *Случайным образом выбирается начальная точка* $S_0 \in S$
2. *Устанавливается начальная "температура"* $t = t_0$, *заданная пользователем*
3. **while** *не достигнут заданный минимум "температуры"* **do**
4. *Генерируется новое состояние системы* $S_i = G(S_i, t_i)$
5. *Сравнить функцию ошибки E в состоянии* S_i *с найденным на текущий момент глобальным минимумом* $\Delta E = E(S_i) - E(S_0)$
6. **if** $\Delta E < 0$, *т. е. у нового состояния значение функции ошибки лучше, чем у текущего глобального минимума* **then**
7. $S_0 = S_i$
8. **else**
9. *Сгенерировать случайное число* β *из интервала [0; 1]*
10. **if** $\beta \leqslant h(\Delta E, t_i)$ **then**
11. *Совершаем переход в новое состояние* $S_0 = S_i$
12. **end**
13. **end**
14. *Уменьшаем "температуру"* $t_i = T(i)$
15. **end**
16. *Возвращаем последнее состояние* S_0

Сравнительный анализ метода имитации отжига и эволюционного метода обучения ИНС

Проведена серия исследований, состоящая из 12 экспериментов. Для каждого из экспериментов выбраны разные обучающие выборки. В табл. 1 приведена информация об использовавшихся обучающих выборках (ОВ).

Выборки 1—8 и 11 являются линейно разделимыми, выборки 9, 10 и 12 являются линейно неразделимыми [18, 19].

При проведении этапа обучения ИНС методом имитации отжига и методом эволюционного обучения с "изолятами" использовался программный пакет *ANNBuilder*.

Для эволюционного метода обучения были заданы следующие параметры:

- число скрытых слоев — 1;
- число нейронов в скрытом слое — 15;
- параметр α — 2 (он характеризует угол наклона сигмоидальной функции активации нейронов [20]);
- число поколений ИНС — 50;
- число скрещиваний в каждом поколении — 4;
- число ИНС в каждом поколении — 200;
- уровень коэффициента вариации (КВ) для мутации — 0,8;
- число лучших ИНС для мутации — 50 %;
- число случайных ИНС из числа лучших ИНС для мутации — 30 % (в нашем случае "мутация" ИНС — изменение *i*-х весовых коэффициентов, для которых коэффициент вариации по всей популяции ниже, чем заданный пользователем в заданном диапазоне);
- порог изменения средней ошибки валидации эволюционного алгоритма для запуска "изоляции" — 0,25 %;
- число лучших ИНС для создания "изолятов" — 10;
- число ИНС в одном "изоляте" — 19 (+ 1 родительская ИНС);
- изменение ВК родительской ИНС при создании дочерних ИНС внутри "изолята" — на 10 %;
- число поколений обучения "изолятов" — 100;
- число скрещиваний в каждом поколении — 4;
- порог числа "неудачных" поколений для обновления "изолятов" — 20;
- допустимое число обновлений "изолятов" — 5.

Параметры метода имитации отжига при обучении ИНС:

- число скрытых слоев — 1;
- число нейронов в скрытом слое — 15;
- параметр $\alpha = 2$;
- начальная "температура" (условные градусы) — 1 000 000;
- конечная "температура" (условные градусы) — 10.

В табл. 2 приведены результаты обучения ИНС методом эволюционного обучения с применением "изолятов" и методом имитации отжига.

После проведения серии экспериментов по обучению ИНС выполнены статистические тесты Уилкоксона для парных случаев [21]. Уровень величины $p < 0,05$ [22, 23] показывает статистическую значимость различий показателей обучения эволюционного метода с "изолятами" и метода имитации отжига.

Данные проведенного сравнительного анализа показывают, что в 10 экспериментах из 12 эволюционный метод с использованием механизма "изолятов" показал лучшие результаты по сравнению с методом имитации отжига. "Изолят" — это множество дочерних ИНС одного поколения, имеющих одну и ту же родительскую ИНС. При создании дочерних ИНС от родительской происходит клонирование последней. Затем происходит изменение значений ВК у дочерней ИНС в пределах заданного диапазона уже имеющихся значений этих же ВК. Имеющиеся на момент клонирования ВК, в свою очередь, хранятся в векторе ВК родительской ИНС. В итоге в *n*-мерном пространстве ВК в окрестностях функции ошибки родительской ИНС формируется пул дочерних ИНС. Исходя из того, что значение функции ошибки родительской ИНС находится вблизи минимума функции ошибки, наличие дочерних ИНС позволяет определить направление, в котором нужно двигаться для достижения этого минимума [12], тем временем как метод имитации отжига носит вероятностный характер, перемещая значение функции ошибки ИНС в пространстве ошибок в пределах заданных температур. Таким образом, эволюционный метод обучения с использованием механизма "изолятов" по результатам сравнительного анализа имеет преимущество перед методом имитации отжига, что и показывает проведенный сравнительный анализ.

Таблица 1

Сведения об обучающих выборках, используемых для обучения искусственных нейронных сетей

Выборка	Число классов	Число входов	Число строк
1	4	3	10 000
2	4	4	10 000
3	4	4	10 000
4	4	4	10 000
5	4	4	10 000
6	2	2	500
7	2	2	500
8	6	3	10 000
9	4	9	10 150
10	2	10	198
11	4	5	10 000
12	2	20	400

Таблица 2

Результаты обучения искусственных нейронных сетей методом эволюционного обучения с применением "изолятов" и методом имитации отжига

Выборка	Медиана (Мин; Макс), %	
	Эволюционный метод обучения искусственных нейронных сетей с применением "изоляции"	Обучение искусственных нейронных сетей методом имитации отжига
1 ($p < 0,05$)	0,28 (0,06; 1,81)	27,19 (25,35; 28,66)
2 ($p < 0,05$)	0,05 (0; 0,25)	27,43 (24,85; 30,55)
3 ($p < 0,05$)	0,185 (0,15; 0,2)	28,12 (25,22; 29,65)
4 ($p < 0,05$)	0 (0; 0)	25 (25; 26,06)
5 ($p < 0,05$	0 (0; 4,78)	24,2 (14,61; 31,4)
6 ($p = –$)	0 (0; 0)	0 (0; 0)
7 ($p = –$)	0 (0; 0)	0 (0; 0)
8 ($p < 0,05$)	2,5 (1,05; 5)	33,61 (11,11; 37,5)
9 ($p < 0,05$)	9,19 (6,74; 10,96)	43,35 (39,96; 47,45)
10 ($p = 0,87$)	39,56 (34,81; 44,3)	40,18 (37,97; 43,67)
11 ($p < 0,05$)	0 (0; 0,04)	25,98 (25; 35,78)
12 ($p < 0,05$)	36,32 (34,59; 38,05)	43,23 (41,19; 43,71)
Примечание: $p = –$ означает отсутствие статистически значимых различий		

Заключение

На основе теоретических предпосылок в направлении обучения искусственных нейронных сетей проведена серия вычислительных экспериментов. В основу данных экспериментов положено обучение двух групп таких ИНС: с применением эволюционного метода и с применением метода имитации отжига. Вычислительные эксперименты проведены для апробации возможности применения разработанного программного пакета *ANNBuilder* в рамках моделирования обучения нейрочипа в целях восстановления поврежденной нервной ткани. Сущность моделирования НЧ заключается в классификации сигналов, полученных с головного мозга. Результаты статистических тестов показали, что в большинстве случаев обучение ИНС с использованием эволюционного метода с применением механизма "изоляции" показывает лучший результат функции ошибки ИНС по сравнению с обучением ИНС методом имитации отжига. В большинстве случаев можно добиться уменьшения числа ошибок классификации входных сигналов и приблизиться к глобальному минимуму функции ошибки ИНС, что сократит число ошибочных классификаций. Так как при моделировании обучения нейрочипа используются методы глубокого обучения, становится актуальным сравнение существующих методов глубокого обучения и разработанного метода эволюционного обучения.

Применение эволюционного алгоритма и механизма "изоляции" для нейрочипа позволяет увеличить точность его обучения, что приводит к уменьшению ошибок классификации сигналов нейрочипа. Данный факт дает возможность осуществить переход к экспериментам на физических и биофизических моделях. Полученный результат сравнения обучения ИНС позволяет также рассматривать применение генетических алгоритмов с "изоляцией" в качестве одного из перспективных направлений использования информационных технологий для задач нейропротезирования.

Список литературы

1. **Neurochip** technology developed: Advances to further brain research of diseases such as Alzheimer's and Parkinson's. URL: https://www.sciencedaily.com/releases/2010/08/100810094619.htm (дата обращения 30.01.2018).
2. **Thakor N. V.** NeuroChip — Platform for Neuronal Injury, Repair and Regeneration. URL: http://lifesciences.ieee.org/lifesciences-newsletter/2014/august-2014/neurochip-platform-for-neuronal-injury-repair-and-regeneration/ (дата обращения 30.01.2018).
3. **Hu C., Dillon J., Kearn J.** et al. NeuroChip: A Microfluidic Electrophysiological Device for Genetic and Chemical Biology Screening of *Caenorhabditis elegans* Adult and Larvae // PLoSONE.

2013. No 8 (5), e64297. URL: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0064297 (дата обращения 30.01.2018).
4. **Khamsi R.** Paralysed man sends e-mail by thought // Nature. 2004. October 13. URL: http://www.nature.com/news/2004/041011/full/news041011-9.html (дата обращения 30.01.2018).
5. **Shilling R.** Paralyzed man with breakthrough brain chip plays guitar video game, swipes credit card // Ohio State University Wexner Medical Center. 2016. April 15. URL: https://wexnermedical.osu.edu/blog/new-tech-helps-paralyzed-man-move-hand-with-mind (дата обращения 30.01.2018).
6. **Baker M.** Tissue models: A living system on a chip // Nature. 2011. March 30. URL: http://www.nature.com/nature/journal/v471/n7340/full/471661a.html (дата обращения 30.01.2018).
7. **Kocaturk M. Gulcur H. O., Canbeyli R.** Toward Building Hybrid Biological in silico Neural Networks for Motor Neuroprosthetic Control // Front Neurorobot. 2015. Vol. 9, No. 8. P. 1—5.
8. **Wan E. A., Kovacs G. T., Rosen M. J., Widrow B.** Development of Neural Network Interfaces for Direct Control of Neuroprostheses. Stanford University Departments of Electrical Engineering and Surgery. 1990. 10 p.
9. **Ayumi V., Rasdi Rere L. M., Fanany M. I., Aniati A. M.** Optimization of Convolutional Neural Network using Microcanonical Annealing Algorithm // Computing Research Repository. 2016. Vol. abs/1610.02306. P. 2—3.
10. **Quoc V. L., Ngiam J., Coates A., Lahiri A., Ng Y. A.** On optimization methods for deep learning // ICML'11 Proceedings of the 28th International Conference on International Conference on Machine Learning, Washington, Computer Science Department, Stanford University, Washington. 2011. P. 1—3.
11. **Лопатин А. С.** Метод отжига // Стохастическая оптимизация в информатике. 2005. № 1. С. 133—149.
12. **Кургалин С. Д., Туровский Я. А., Борзунов С. В., Адаменко А. А.** Теоретические аспекты оптимизации эволюционного обучения нейрочипов с использованием "изолятов" // Информационные технологии. 2016. Т. 22, № 11. С. 888—889.
13. **Туровский Я. А., Кургалин С. Д., Адаменко А. А.** Свидетельство о государственной регистрации программы для ЭВМ № 2015619800 ANNBuilder 1.4.9. Воронеж, 2015.
14. **Туровский Я. А., Кургалин С. Д., Адаменко А. А.** Свидетельство о государственной регистрации программы для ЭВМ № 2016614262 ANNBuilder 1.8.8. Воронеж, 2015.
15. **Туровский Я. А., Кургалин С. Д., Адаменко А. А.** Свидетельство о государственной регистрации программы для ЭВМ № 2016619398 ANNBuilder 2.1.0. Воронеж, 2016.
16. **Туровский Я. А., Кургалин С. Д., Адаменко А. А.** Сравнительный анализ программных пакетов для работы с искусственными нейронными сетями // Вестник Воронежского государственного университета. Серия "Системный анализ и информационные технологии". 2016. Вып. 1. С. 161—168.
17. **Туровский Я. А., Адаменко А. А.** Сравнительный анализ результатов обучения искусственных нейронных сетей в задачах обработки сигналов на основе эволюционного алгоритма с применением и без применения "изоляции" // Цифровая обработка сигналов. 2017. Вып. 4. С. 46—50.
18. **Уоссермен Ф.** Нейрокомпьютерная техника: Теория и практика / Пер. с англ. Ю. А. Зуева, В. А. Точенова. М.: Мир, 1992. С. 31—33.
19. **Глоссарий** BaseGroupLabs, URL: https://basegroup.ru/community/glossary/linearpart (дата обращения 30.01.2018).
20. **Activation** function, article in Wikipedia. URL: https://en.wikipedia.org/wiki/Activation_function (дата обращения 30.01.2018).
21. **Nonparametrics** Statistics Notes — Wilcoxon Matched Pairs Test. URL: http://documentation.statsoft.com/STATISTICAHelp.aspx?path=Nonparametrics/NonparametricAnalysis/Dialogs/StartupPanel/NonparametricsStatisticsStartupPanelWilcoxonmatchedpairstest (дата обращения 30.01.2018).
22. **Лапач С. Н., Чубенко А. В., Бабич П. Н.** Статистика в науке и бизнесе. Киев: Морион, 2002. С. 164—166.
23. **Кобзарь А. И.** Прикладная математическая статистика. Справочник для инженеров и научных работников. М.: Физматлит, 2006. С. 453—460.

Comparative Analysis of the Evolution Method with use of "Isolates" and the Annealing Simulation Method in Learning of Artificial Neural Networks

Ya. A. Turovsky, yaroslav_turovsk@mail.ru, Voronezh State University, Voronezh, 394018, Russian Federation, **A. A. Adamenko,** adamenko.artem@gmail.com, Voronezh State University of Engineering Technologies, Voronezh, 394036, Russian Federation

Corresponding author:

Turovsky Yaroslav A., Associate Professor, Voronezh State University, Voronezh, 394018, Russian Federation, E-mail: yaroslav_turovsk@mail.ru

Received on February 06, 2018
Accepted on February 13, 2018

A series of computational experiments on the training of artificial neural networks was carried out using the evolutionary method of teaching ANN with the use of "isolates" and an annealing simulation method. The results of a comparative analysis of these experiments showed the possibility of using the developed software package ANNBuilder to simulate the training of a neurochip (NP) with the aim of restoring damaged neural tissue. The first group of these experiments used a simple evolutionary algorithm for teaching ANN with the use of the "isolation" mechanism, which consists in crossing the parent ANN and the emergence of a daughter ANN inside one "isolate" — the spatially restricted region of weight coefficients (WC) of the ANN. The second group used the simulated annealing method, which is based on the physical process of heating the metal to a certain temperature, which forces the atoms of the crystal lattice to leave their positions. It is concluded that when using the evolutionary method of teaching ANN using

the "isolation" mechanism in comparison with the second group of experiments, in most cases the best value of the ANN error function is achieved, which justifies the use of this method in simulating the training of low frequencies.

Keywords: *neurochip, evolutionary algorithm, annealing simulation method, isolation, isolate, artificial neural networks*

For citation:

Turovsky Ya. A., Adamenko A. A. Comparative Analysis of the Evolution Method with use of "Isolates" and the Annealing Simulation Method in Learning of Artificial Neural Networks, *Programmnaya Ingeneria,* 2018, vol. 9, no. 4, pp. 185—190.

DOI: 10.17587/prin.9.185-190

References

1. ***Neurochip*** *technology developed: Advances to further brain research of diseases such as Alzheimer's and Parkinson's*, available at: https://www.sciencedaily.com/releases/2010/08/100810094619.htm

2. **Thakor N. V.** *NeuroChip — Platform for Neuronal Injury, Repair and Regeneration,* available at: http://lifesciences.ieee.org/lifesciences-newsletter/2014/august-2014/neurochip-platform-for-neuronal-injury-repair-and-regeneration/

3. **Hu C., Dillon J., Kearn J., Murray C., O'Connor V., Holden-Dye L.** et al. NeuroChip: A Microfluidic Electrophysiological Device for Genetic and Chemical Biology Screening of *Caenorhabditis elegans* Adult and Larvae, *PLoSONE,* 2013, no. 8 (5), e64297, available at: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0064297

4. **Khamsi R.** Paralysed man sends e-mail by thought, *Nature,* 2004, October 13, available at: http://www.nature.com/news/2004/041011/full/news041011-9.html

5. **Shilling R.** Paralyzed man with breakthrough brain chip plays guitar video game, swipes credit card, *The Ohio State University Wexner Medical Center,* 2016, April 15, available at: https://wexnermedical.osu.edu/blog/new-tech-helps-paralyzed-man-move-hand-with-mind

6. **Baker M.** Tissue models: A living system on a chip, *Nature,* 2011, March 30, available at: http://www.nature.com/nature/journal/v471/n7340/full/471661a.html

7. **Kocaturk M., Gulcur H. O., Canbeyli R.** Toward Building Hybrid Biological in silico Neural Networks for Motor Neuroprosthetic Control, *Front Neurorobot.* 2015, vol. 9, no. 8, pp. 1—5.

8. **Wan E. A., Kovacs G. T., Rosen M. J., Widrow B.** *Development of Neural Network Interfaces for Direct Control of Neuroprostheses*, Stanford University Departments of Electrical Engineering and Surgery, 1990, 10 p.

9. **Ayumi V., Rasdi Rere L. M., Fanany M. I., Aniati A. M.** Optimization of Convolutional Neural Network using Microcanonical Annealing Algorithm, *Computing Research Repository*, 2016, vol. abs/1610.02306, pp. 2—3.

10. **Quoc V. L., Ngiam J., Coates A., Lahiri A., Ng Y. A.** On optimization methods for deep learning Computer Science Department, *ICML'11 Proceedings of the 28th International Conference on International Conference on Machine Learning,* Stanford University, Washington, 2011, 3 p.

11. **Lopatin A. S.** Metod otzhiga (Method of Annealing), *Stohasticheskaya optimizaciya v informatike,* 2005, no. 1, pp. 133—149 (in Russian).

12. **Kurgalin S. D., Turovsky Ya. A., Borzunov S. V., Adamenko A. A.** Teoreticheskie aspekty optimizacii e'volyucionnogo obucheniya nejrochipov s ispol'zovaniem "izolyatov" (Theoretical Aspects of Optimization of Evolutionary Learning of Neurochips Using "Isolates"), *Informacionnye tehnologii*, 2016, vol. 22, no. 11, pp. 888—889 (in Russian).

13. **Kurgalin S. D., Turovskij Ya.A., Adamenko A. A.** Svidetel'stvo o gosudarstvennoj registracii programmy dlya E'VM № 2015619800 ANNBuilder 1.4.9. Voronezh, 2015 (in Russian).

14. **Turovsky Ya. A., Kurgalin S. D., Adamenko A. A.** Svidetel'stvo o gosudarstvennoj registracii programmy dlya E'VM № 2016614262 ANNBuilder 1.8.8. Voronezh, 2015 (in Russian).

15. **Turovsky Ya. A., Kurgalin S. D., Adamenko A. A.** Svidetel'stvo o gosudarstvennoj registracii programmy dlya E'VM № 2016619398 ANNBuilder 2.1.0. Voronezh, 2016 (in Russian).

16. **Turovsky Ya. A., Kurgalin S. D., Adamenko A. A.** Sravnitel'nyj analiz programmnyx paketov dlya raboty s iskusstvennymi nejronnymi setyami (Comparative Analysis of Software Package for Working with Artificial Neural Networks), *Vestnik Voronezhskogo gosudarstvennogo universiteta. Seriya "Sistemnyj analiz i informacionnye texnologii"*, 2016, vol. 1, pp. 161—168 (in Russian).

17. **Turovsky Ya. A., Adamenko A. A.** Sravnitel'nyj analiz rezul'tatov obuchenija iskusstvennyh nejronnyh setej v zadachah obrabotki signalov na osnove jevoljucionnogo algoritma s primeneniem i bez primenenija "izoljacii" (Comparative analysis of the results of training artificial neural networks in signal processing problems based on the evolutionary algorithm with and without the use of " isolation"), *Cifrovaja obrabotka signalov,* 2017, no. 4, pp. 46—50 (in Russian).

18. **Uossermen F.** *Nejrokomp'yuternaya texnika: Teoriya i praktika* (Neurocomputer technique: Theory and practice). Per. s angl. Yu. A. Zueva, V. A. Tochenova, Moscow, Mir, 1992, pp. 31—33 (in Russian).

19. **Glossary** BaseGroupLabs, available at: https://basegroup.ru/community/glossary/linearpart

20. **Activation** function, article in Wikipedia, available at: https://en.wikipedia.org/wiki/Activation_function

21. **Nonparametrics** Statistics Notes — Wilcoxon Matched Pairs Test, available at: http://documentation.statsoft.com/STATISTICAHelp.aspx?path=Nonparametrics/NonparametricAnalysis/Dialogs/StartupPanel/NonparametricsStatisticsStartupPanelWilcoxonmatchedpairstest

22. **Lapach S. N., Chubenko A. V., Babich P. N.** *Statistika v nauke i biznese* (Statistics in science and business), Kiev, Morion, 2002, pp. 164—166 (in Russian).

23. **Kobzar' A. I.** *Prikladnaya matematicheskaya statistika. Spravochnik dlya inzhenerov i nauchnyx rabotnikov* (Applied mathematical statistics. A Handbook for engineers and researchers), Moscow, Fizmatlit, 2006, pp. 453—460.

24 мая 2018 г. в отеле “Holiday Inn Suschevsky", Москва,
информационно-аналитическое агентство TelecomDaily
проводит Вторую Всероссийскую конференцию

"Интернет вещей в жилищно-коммунальном хозяйстве — IoT в ЖКХ 2018"

На предстоящей конференции будут обсуждены следующие вопросы

- Состояния рынка Интернета вещей для ЖКХ. Перспективы и основные направления развития
- Цифровизация ЖКХ как направление реализация программы Цифровой экономики, ее влияние на развитие ЖКХ-услуг и деятельность участников коммунального рынка
- Перспективные направления дальнейшего развития ГИС ЖКХ
- Построение информационной инфраструктуры для создания экосистемы IoT ЖКХ и предоставление цифровых услуг для управления и обслуживания коммунального хозяйства
- Государственно-частное партнерство для развития и цифровой трансформации ЖКХ
- Интернет вещей как новая среда социально-экономических отношений для жителей России
- Развитие и внедрение сетей четвертого и пятого поколения и их влияние на экосистему IoT и услуг M2M в ЖКХ
- Построение федеральной сети узкополосной связи по технологии LPWAN для сбора и обработки телематической информации
- Создание и реализация концепции "Умные города России"
- Интернет вещей в строительстве и ЖКХ — наступившая доподлинная реальность?
- Использование беспилотных и интеллектуальных робототехнических комплексов
- Создание инфраструктуры для сбора и хранения информации
- Использование технологий блокчейна в ЖКХ
- Искусственный интеллект и машинное обучение: разработки и первые подходы к практическому применению в ЖКХ
- Цифровые платформы для интеллектуального управления энергосбережением в сфере ЖКХ
- Оценка возможностей и перспективы массового производства конкурентоспособных IoT-устройств для отрасли ЖКХ с использованием отечественной элементной базы
- Использование навигационных систем ГЛОНАСС/GPS в IoT/M2M-системах ЖКХ
- Концепция интеллектуального дома в умном городе на основе решений M2M/IoT
- Интеллектуальные системы поддержки принятия решений с использованием IoT в чрезвычайных и кризисных ситуациях

Подробности: http://www.tmtconferences.ru/iot2018.html

25—26 мая 2018 г. в Минске пройдет 23-я международная конференция в области обеспечения качества ПО "Software Quality Assurance Days".

Конференция посвящена следующим вопросам, связанным с тестированием и обеспечением качества программного обеспечения:

- функциональное тестирование;
- интеграционное тестирование;
- тестирование производительности;
- автоматизация тестирования и инструментальные средства;
- конфигурационное тестирование;
- тестирование удобства использования (usability);
- тестирование защищенности (security);
- статические методы обеспечения качества;
- внедрение процессов тестирования на предприятии;
- управление процессами обеспечения качества ПО;
- менеджмент команд тестировщиков и инженеров качества ПО;
- аутсорсинг тестирования;
- тестирование системных приложений (не web), а также тестирование игр и приложений для мобильных устройств;
- мотивация проектной команды и сертификация специалистов в области обеспечения качества ПО.

Для многих специалистов и руководителей "Software Quality Assurance Days" —это реальная возможность заявить о себе, повысить профессиональный уровень сотрудников, которые отвечают за ПО и, тем самым, укрепить конкурентные позиции и создать преимущество. SQA Days — это платформа общения и обмена опытом для людей, вовлеченных в сферу тестирования ПО. Ведущие профессионалы смогут рассказать о своих достижениях, показать, как эффективно использовать инструменты, методики и методологии. Для начинающих — это отличный шанс приобрести новые полезные знакомства в профессиональной среде.

Сайт конференции: https://sqadays.com

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

Сдано в набор 15.02.2018 г. Подписано в печать 26.03.2018 г. Формат 60×88 1/8. Заказ PI418
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**

XX Всероссийская конференция

НАУЧНЫЙ СЕРВИС В СЕТИ ИНТЕРНЕТ

с 17 по 22 сентября 2018 г.

Конференцию проводит
Институт прикладной математики
им. М.В. Келдыша РАН

Конференция посвящена основным направлениям и тенденциям использования интернет-технологий в современных научных исследованиях. Основная цель конференции — предоставить возможность для обсуждения, апробации и обмена мнениями о наиболее значимых результатах, полученных ведущими российскими учеными за последнее время в данной области деятельности.

Конференция серии «Научный сервис в сети Интернет» проводится в двадцатый раз. В 2017 г. в ее работе приняли участие свыше 150 человек.

Тематика конференции

- Научные исследования и интернет, интернет-представительство научных организаций и проектов.
- Решение задач и обработка данных на суперкомпьютерах центров коллективного пользования.
- Интернет-проекты в области параллельных вычислений, математическое моделирование, вычислительные сервисы.
- Интернет-проекты для биомедицины.
- Модели и методы построения поисковых систем и систем навигации в интернете, технологии и системы распределенного хранения и обработки данных.
- Технологии и опыт построения информационных систем и баз данных, документации и результатов эксперимента на основе интернет-технологий.
- Цифровые библиотеки и библиографические базы, семантический веб, наукометрия в интернете.
- Онлайновая научная публикация, открытая наука, живая публикация, онлайновое рецензирование, мультимедийные иллюстрации.
- Популярный научный интернет, онлайновые энциклопедии, история науки в интернете.
- Интернет-активность ученого, персональная страница, профили ученого в библиографических базах, аттестация в интернете.
- Системное и инструментальное программное обеспечение, языки и модели программирования, формальные методы для интернет-технологий.

Конференция проводится с 17 по 22 сентября в пансионате «Моряк», расположенном в 20 километрах от Новороссийска в живописном месте на берегу Черного моря недалеко от поселка Дюрсо.

Сайт конференции: http://agora.guru.ru/abrau2018

Издательство «НОВЫЕ ТЕХНОЛОГИИ»

выпускает *научно-технические журналы*

Теоретический и прикладной научно-технический журнал

ПРОГРАММНАЯ ИНЖЕНЕРИЯ

В журнале освещаются состояние и тенденции развития основных направлений индустрии программного обеспечения, связанных с проектированием, конструированием, архитектурой, обеспечением качества и сопровождением жизненного цикла программного обеспечения, а также рассматриваются достижения в области создания и эксплуатации прикладных программно-информационных систем во всех областях челове ческой деятельности.

Подписные индексы по каталогам:
«Роспечать» – 22765; «Пресса России» – 39795

Ежемесячный теоретический и прикладной научно-технический журнал

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

В журнале освещаются современное состояние, тенденции и перспективы развития основных направлений в области разработки, производства и применения информационных технологий.

Подписные индексы по каталогам:
«Роспечать» – 72656;
«Пресса России» – 94033

Ежемесячный междисциплинарный теоретический и прикладной научно-технический журнал

НАНО- и МИКРОСИСТЕМНАЯ ТЕХНИКА

В журнале освещаются современное состояние, тенденции и перспективы развития нано- и микросистемной техники, рассматриваются вопрос разработки и внедрения нанои микросистем в различные области науки, технологии и производства.

Подписные индексы по каталогам:
«Роспечать» – 79493;
«Пресса России» – 27849

Ежемесячный теоретический и прикладной научно-технический журнал

МЕХАТРОНИКА, АВТОМАТИЗАЦИЯ, УПРАВЛЕНИЕ

В журнале освещаются достижения в области мехатроники, интегрирующей механику, электронику, автоматику и информатику в целях совершенствования технологий производства и создания техники новых поколений. Рассматриваются актуальные проблемы теории и практики автомати ческого и автоматизированного управления техническими объектами и технологическими процессами в промышленности, энергетике и на транспорте.

Подписные индексы по каталогам:
«Роспечать» – 79492;
«Пресса России» – 27848

Научно-практический и учебно-методический журнал

БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

В журнале освещаются достижения и перспективы в области исследований, обеспечения и совершенствования защиты человека от всех видов опасностей производственной и природной среды, их контроля, мониторинга, предотвращения, ликвидации последствий аварий и катастроф, образования в сфере безопасности жизнедеятельности.

Подписные индексы по каталогам:
«Роспечать» – 79963;
«Пресса России» – 94032

Адрес редакции журналов для авторов и подписчиков:
107076, Москва, Стромынский пер., 4. Издательство "НОВЫЕ ТЕХНОЛОГИИ".
Тел.: (499) 269-55-10, 269-53-97. Факс: (499) 269-55-10. E-mail: antonov@novtex.ru

ISSN 2220-3397. Программная инженерия. 2018. Том 9. № 4. 145–192. Индексы: «Роспечать» – 22765, «Пресса России» – 39795