ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

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

СОДЕРЖАНИЕ

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

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

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

№ 3
March
2015

Published since September 2010

CONTENTS

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

УДК 004.4

А. А. Бомбин, инженер, **В. А. Галатенко,** д-р физ.-мат. наук, зав. сектором,
К. А. Костюхин, канд. физ.-мат. наук, ст. науч. сотр., e-mail: kost@niisi.msk.ru,
Научно-исследовательский институт системных исследований РАН (НИИСИ РАН), Москва

К вопросу самовосстановления программного обеспечения в ARINC- и POSIX-системах

Рассмотрены существующие механизмы самоуправления в программном обеспечении сложных комплексов и описаны разработанные при участии авторов средства самолечения и самовосстановления ARINC- и POSIX-систем на примере отечественной ОС реального времени "Багет".

Ключевые слова: *самолечение, самоуправление, самовосстановление, POSIX, ARINC*

Введение

Внутренняя сложность, разнородный характер и динамичное изменение современного, большого по объему кода программного обеспечения приводят к сложностям, а в ряде случаев к отсутствию возможности реализовать подход к администрированию компьютерных систем, в основе которого находится человек. Еще в 2001 г. Пол Хорн (Paul Horn), вице-президент IBM, говорил, что основным препятствием в развитии IT-индустрии будет сложность программного обеспечения [1]. Любая сложная система, функционирующая продолжительное время, рано или поздно, в силу тех или иных причин, отклоняется от регламентированного режима функционирования и это приводит к необходимости восстанавливать такой режим. В подобных системах могут возникать различные экстренные ситуации, начиная с выхода из строя отдельных компонентов и заканчивая аварийной остановкой всей системы. Они требуют адекватных методов выявления, исследования и выбора подходящих действий для исправления.

Осознание отмеченной проблемы привело к идее создания механизмов, помогающих системе автономно находить и обрабатывать ошибки, а также перенастраиваться с учетом изменяющихся условий. Таким образом, появилось понятие самоуправления (*Self-Management*) программного обеспечения, включающее в себя реализацию нескольких "само-" функций (механизмов), таких как самоадаптация, самозащита, самолечение и т. д. В настоящее время эти механизмы находят все большее применение, и, как следствие, проводятся исследования по методикам их реализации [2].

Отечественные аппаратно-программные комплексы со сложной архитектурой, работающие под управлением оперативных систем (ОС) реального времени, также нуждаются в средствах самоуправления, позволяющих эффективно решать поставленные перед такими комплексами задачи.

Целью данной работы являются краткий обзор существующей классификации механизмов самоуправления в программном обеспечении [3] и описание разработанных при участии авторов средств самолечения и самовосстановления систем в стандартах ARINC и POSIX на примере отечественной ОС реального времени "Багет" [4].

Классификация механизмов самоуправления

Под самоуправлением (*Self-Management*) [3] будем понимать способность системы автоматически (без участия администратора) и динамически (по мере появления необходимости во время функционирования) менять свое поведение, а также характеристики в целях улучшения функциональных возможностей и степени надежности системы. Механизмы самоуправления в программном обеспечении можно классифицировать в соответствии с их назначением, а именно — адаптация, контроль и поддержание требований к режиму его функционирования.

К первому из упомянутых классов относятся механизмы, связанные с восстановлением и решением внутренних задач, которые возникают в программной системе. В этом классе можно выделить механизмы, меняющие архитектуру системы в целом (самонастройка) и механизмы, действующие на отдельные ее компоненты (самоорганизация). Во втором классе выделяют механизмы контроля внутренних изменений (самолечение) системы и механизмы контроля ее внешних изменений (самоадаптация). Третий класс включает механизмы поддержания требований безопасности программной системы (самозащита) и качества ее работы (самооптимизация).

Под самоадаптацией (*Self-Adaptiveness*) понимается способность программной системы динамически реагировать на изменения в среде окружения и менять свое поведение для поддержания требуемого качества предоставляемого сервиса, улучшения функциональных характеристик и результатов работы системы. Роберт Ладдага [5] определяет самоадаптацию как способность программного обеспечения оценивать и менять свое поведение в случае, если оценка показывает, что требуемые результаты его работы не достигаются или возможны лучшие результаты. Система, обладающая этим свойством, должна поддерживать описание целей, которых необходимо достичь (например, время отклика), и содержать базу альтернативных действий и алгоритмов. Она также должна быть способна оценивать качество работы определенных фрагментов программного кода и иметь возможность динамически заменять их на альтернативные.

Базовым механизмом самоадаптации является цикл обратной связи: позитивные результаты утверждаются и усиливаются, негативные вызывают отмену изменений. Такой цикл обычно включает следующие четыре фазы: сбор информации; анализ; планирование действий; их выполнение. С каждой фазой связаны определенные задачи, такие как оценка достоверности собранной информации, необходимость хранения прошлых результатов, приоритеты самоадаптации по всем циклам, в каждом отдельном цикле, а также ряд других задач [6].

Самонастраивающаяся (*Self-Configuring*) система имеет механизмы модификации взаимодействий между своими компонентами. Самонастраиваемость обозначает способность системы автоматически реагировать на динамические изменения структуры системы, такие как добавление нового компонента или замена одного компонента другим, удаление некоторых модулей или изменение взаимодействия с новыми. Установка, конфигурация и слияние сложных систем – процедуры ресурсозатратные, кроме того, при их выполнении велика вероятность ошибки. Автоматическая перенастройка системы проводится с использованием высокоуровневых правил, описывающих, что система должна делать, а не то, как она должна это делать. Когда добавляется новый компонент, система должна проверить состояние программного окружения и попытаться использовать его функциональные возможности с тем, чтобы добиться максимального эффекта.

В Техасском университете (США) проводились исследования по использованию нейроэволюционных методов перераспределения ресурсов для повышения производительности процессора [7] и эксперимент по созданию распределенной самонастраивающейся сети, в котором было показано отсутствие идеальной конфигурации, оптимальной во всех случаях, и оценен потенциал улучшения ее производительности за счет реконфигурирования [8].

Самолечение подразумевает наличие способности у системы находить, исследовать и реагировать на ошибки в работе. Компоненты или системы, обладающие механизмом самолечения, должны уметь исследовать системные ошибки и, учитывая наложенные извне ограничения, применять подходящие в данной ситуации исправления. Для автоматического нахождения ошибок система должна обладать знанием о своем ожидаемом поведении и способностью оценивать корректность фактического поведения.

Примером реализации самолечения может служить разработанная в Массачусетском Технологическом Институте (США) система восстановления постоянных структур данных [9]. Имея некоторый набор правил, которым должны удовлетворять структуры, система проводит поиск их нарушения, порождает схему из допустимых наборов состояний компонентов в виде дизъюнктивной нормальной формы и ищет наименее затратный способ привести систему в одно из допустимых состояний, применяя восстанавливающие действия к отдельным конъюнкциям. В качестве другого примера можно рассмотреть программные средства восстановления серверных приложений ASSURE [10], автоматически создающие точки восстановления и, при необходимости, откатывающие подконтрольную систему в подходящую точку.

Под самооптимизацией понимается способность системы следить за ресурсами и автономно оптимизировать их использование, перераспределяя эти ресурсы для поддержки требуемого уровня качества работы системы. Свойством самооптимизации обладает, в частности, инструментальный комплекс Truffle [11], применяемый для реализации подчиненных языков в Java. Система, реализующая язык, создает интерпретатор абстрактного синтаксического дерева (*Abstract Syntax Tree*, AST), который позволяет переписывать дерево во время его интерпретации, используя данные профилирования и, таким образом, специфицируя дерево. Когда дерево достигнет стабильного состояния, оно компилируется в оптимизированный машинный код.

В самоорганизуемой системе компоненты способны настраивать взаимодействие между собой, учитывая ограничения, наложенные дизайном системы. Как следствие, добавление новых компонентов не нарушает свойства архитектуры системы, поскольку обеспечивается выполнение ограничений. В качестве примера можно рассмотреть группы роботов, обладающих групповым интеллектом, которые разрабатываются в рамках проекта Килобот (The Kilobot Project) [12]. Эта система состоит из большого числа простых одинаковых роботов, взаимодействующих между собой для выполнения какой-либо задачи. При добавлении нового компонента (робота), должно обеспечиваться его встраивание в общую структуру, возможно, путем изменения схемы взаимодействия между роботами.

Системы этого типа способны отслеживать возможные внешние атаки, например, неавторизованный вход, вирусы, спам или атаки на "отказ-в-обслуживании",

а также принимать необходимые меры для усиления безопасности.

Средствами самозащиты обладает, например, программный комплекс Jade [3], использующий особенности архитектуры для создания механизмов самозащиты в приложениях java 2 Enterprise Edition (J2EE). Его механизмы автоматически приводят в действие брандмауэры на компонентах и настраивают их в соответствии с заданной J2EE архитектурой, что позволяет отслеживать часть нелегальных действий и принимать адекватные контрмеры.

Самолечение в ARINC-системах

ARINC-спецификации разрабатывались для работы в авиационной бортовой сети реального времени. Компьютерная система в этой модели разделяется на несколько уровней — модуль (вся система), раздел (отдельная операционная система в модуле), процесс (отдельный процесс в конкретном разделе). На рис. 1 представлен пример организации модуля, состоящего из четырех разделов с различными операционными системами. Для реализации требуемых функций и удовлетворения временных ограничений в каждом разделе должна предоставляться возможность работы нескольких процессов единовременно. Операционная система должна предоставлять сервисы для управления и поддержки условий работы всех процессов в разделе.

В ARINC-системах функции поддержки самолечения, общие для всех разделов, собраны в так называемый монитор здоровья (*Health Monitor*, HM) [13]. Монитор здоровья предназначен для отслеживания ошибок в прикладных программах, системных ошибок и ошибок аппаратуры, а также для их устранения (или минимизации ущерба от этих ошибок). Ошибки могут возникать на уровне процесса, раздела или модуля. Модульные ошибки влияют на все разделы внутри этого модуля (ошибки настройки модуля при запуске, ошибки выполнения системных функций, сбой питания и т. д.). Ошибки раздела влияют только на один раздел (ошибки настройки или инициализации раздела, ошибки управления процессами и т. д.). Соответственно, ошибки процессов влияют на несколько потоков в процессе или на весь процесс (ошибки выполнения процессов, ошибка выполнения приложения, замеченная им, и т. д.).

Уровень ошибки определяется характером ошибки и состоянием системы. Ошибки, найденные на одном уровне, могут быть для обработки подняты на более высокий уровень. Реакция на ошибку зависит от ее уровня. На ошибки модуля или раздела система реагирует, руководствуясь специальными таблицами, которые задаются при создании системы. Ошибки процессов обрабатываются специальным процессом с наивысшим приоритетом, который именуется процессом обработки ошибок. Ошибки, происходящие в процессе обработки ошибок, считаются ошибками уровня раздела.

Для ошибок уровня модуля возможными восстановительными действиями являются останов или перезапуск модуля. Аналогичные действия допускаются для ошибок уровня раздела: останов раздела, либо его рестарт. Для ошибок уровня процесса процесс обработки ошибок может выполнить одно из следующих действий:

- записать информацию об ошибке, но не предпринимать восстановительных действий;

Рис. 1. Пример организации модуля, включающего разделы с разными операционными системами

- подтвердить наличие ошибки некоторое число раз, прежде чем предпринять какие-либо действия;
- остановить процесс с ошибкой и перезапустить его;
- остановить процесс и запустить другой процесс;
- перезапустить весь раздел;
- остановить весь раздел.

Базовыми функциями монитора здоровья являются следующие функции:

- REPORT_APPLICATION_MESSAGE, предназначеная для записи информации о произошедшей ошибке и передачи сообщений о ней в процесс обработки;
- CREATE_ERROR_HANDLER, создающая для текущего раздела процесс обработки ошибок;
- GET_ERROR_STATUS, позволяющая получить информацию о произошедшей ошибке в процессе обработки;
- RAISE_APPLICATION_ERROR, передающая управление процессу обработки ошибок при обнаружении программой ошибки.

На листинге 1 (см. приложение) представлен пример системы с обработкой ошибок. Главная функция создает рабочий поток, начинающийся с testCodeFunction, а также запускает процесс обработки ошибок с функцией testErrorFunction. В рабочем потоке либо происходит численная ошибка, либо процесс обработки ошибки вызывается искусственно. Процесс обработки ошибки получает информацию о произошедшей ошибке, распечатывает ее и завершает работу.

Самолечение в POSIX-системах

В 2001 г. департамент стандартов IEEE включил в очередную редакцию POSIX рекомендации по трассировке программно реализуемых процессов в целях унификации интерфейса средств сбора и анализа данных об их выполнении [13]. Именно эти функциональные возможности POSIX-систем предлагается использовать при реализации средств самолечения программного обеспечения.

Трассировка в стандарте POSIX-2001. Стандарт фиксирует минимальные функциональные возможности средств трассировки, которые должна предоставлять POSIX-совместимая ОС. Под трассировкой в стандарте POSIX-2001 понимается порождение, накопление и анализ данных о событиях, имевших место при выполнении пользовательского приложения.

В трактовке стандарта POSIX-2001 в трассировке логически участвуют три процесса, которые физически могут совпадать между собой — трассируемый (целевой), трассирующий (управляющий трассировкой) и анализирующий данные трассировки.

Сведения о действиях, проводимых при выполнении приложения, фиксируются в виде объектов данных, называемых событиями трассировки (или, для краткости, просто событиями). События и данные, необходимые для их интерпретации, записываются в потоки трассировки.

Трассируемый процесс должен быть специальным образом организован: реализующая его программа должна содержать точки трассировки — действия, способные генерировать события трассировки. Для каждого трассируемого процесса должен быть открыт по крайней мере один поток трассировки. Процесс, создавший поток трассировки, называется *трассирующим* (управляющим трассировкой). *Анализирующим* называется процесс, извлекающий события трассировки из общего потока в целях получения информации о поведении трассируемого приложения.

События трассировки подразделяют на пользовательские и системные. Пользовательские события порождаются при вызове функции posix_trace_event(). Системные события генерируются в ответ на действия приложения или ОС. Стандартом предусмотрен механизм фильтрации, позволяющий отменять генерацию событий определенных типов, сокращая таким образом объем трассировочных данных.

Одно из главных требований к системе трассировки — минимизация накладных расходов. Исходя из этого требования, потоки трассировки, как правило, хранят в оперативной памяти. Для получения стабильной копии потока в целях последующего анализа, стандарт предусматривает его сохранение в журнале трассировки в долговременной памяти. И для потоков, и для журналов трассировки правила обработки ситуации заполнения могут сводиться к записи новых событий поверх самых старых или к приостановке трассировки. Для потоков

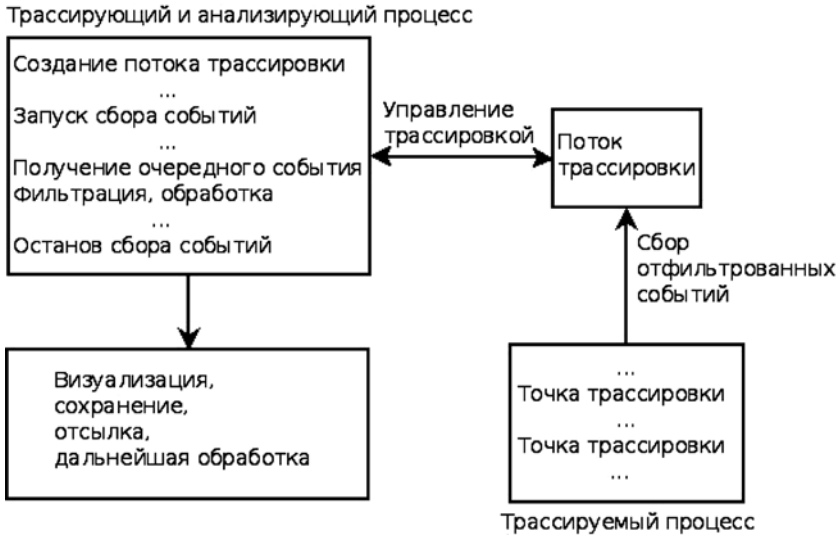


Рис. 2. Анализ трассы в ходе сбора событий

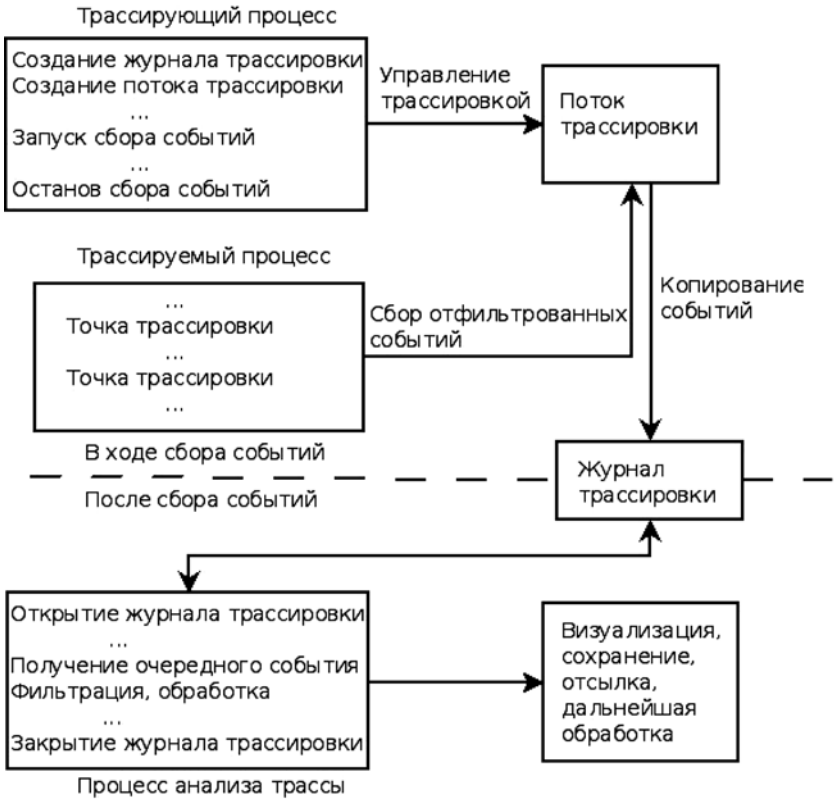


Рис. 3. Анализ трассы после сбора событий

возможен сброс в журнал с последующей очисткой, а для журналов — потенциально неограниченное расширение.

Стандарт определяет два способа анализа трассы — в ходе сбора событий (*online analysis*) и после него (*offline analysis*). В первом случае анализирующий процесс читает данные из потока событий, во втором случае — из журнала. Оба способа представлены соответственно на рис. 2 и 3.

Ниже приведены основные требования к программному интерфейсу трассировки POSIX-систем.

- Должна предоставляться возможность параллельной трассировки системных и пользовательских событий.
- Должна предоставляться возможность трассировать события, относящиеся к конкретному процессу, а также системные события, не относящиеся к какому-либо процессу.
- Должна предоставляться возможность управления трассировкой некоторого процесса, как потоками управления этого процесса, так и внешними процессами.
- Должна предоставляться возможность управления трассировкой некоторого потока управления другими потоками управления данного процесса.
- В одно и то же время можно трассировать только один процесс, при этом запрещена трассировка сущностей "бóльших", чем процесс, например, группы процессов.
- Каждый вновь порождаемый поток трассируемого процесса по умолчанию также является трассируемым.
- Независимо разрабатываемый код может подвергаться трассировке без дополнительных настроек и без конфликтов.
- Для доступа к потоку событий должен использоваться стандартный интерфейс прикладного программирования.
- Формат потока и журнала трассировки не определяется.
- Должна быть возможность поддержки нескольких потоков трассировки в системе.

Всю необходимую дополнительную информацию можно получить из работы [3].

Можно отметить несколько слабых мест этого стандарта. Во-первых, нет возможности отлаживать группы процессов, что является серьезным ограничением для применения POSIX-2001 при мониторинге состояния распределенных приложений. Во-вторых, в стандарте не отражен механизм группировки потоков управления внутри одного процесса (и вообще, "минимальной" единицей применения трассировки является именно процесс, а не отдельный поток). В-третьих, не прописаны механизмы и интерфейсы группировки событий (есть только две группы — пользовательские события и системные события).

Тем не менее при разработке средств мониторинга и самолечения целесообразно использовать интерфейс POSIX-2001 в качестве базового с последующим его расширением.

Реализация механизма самолечения в POSIX-системах. Система называется *самостабилизирующейся*, если из любой конфигурации она за конечное время приходит в безопасную конфигурацию, а из безопасной всегда переходит в безопасную. Для того чтобы аппаратно-программная система была самостабилизирующейся, необходимо, чтобы этим свойством обладали аппаратура, операционная система и программные приложения. Ниже предложена методика реализации самостабилизирующейся программной системы, основанная на анализе потока событий специально выделенным управляющим потоком.

Для реализации механизма самостабилизации (самолечения) разработчик программной системы должен описать так называемые *сигнатуры правильного поведения системы*. Это последовательности пользовательских и системных событий, выполнение которых свидетельствует о корректном выполнении системой заданных функций. Соответственно,

система должна быть надлежащим образом настроена, т. е. в критические участки кода разработчик должен добавить вызовы функции генерации пользовательских событий, а также включить протоколирование необходимых системных событий.

Далее, при разработке системы необходимо зарезервировать определенные ресурсы (время выполнения и память) для отдельного потока управления или процесса, который будет выполнять функции монитора здоровья целевой системы. Монитор здоровья следит за соответствием возникающих событий сигнатурам правильного поведения системы и в случае обнаруженного расхождения проводит действия по стабилизации системы, в простейшем случае — перезапуск нескольких (или всех) потоков управления или процессов.

На листингах 2 и 3 (см. приложение) представлен пример POSIX-процесса, содержащего три потока. Два из них — это сервер и клиент, вазаимодействующие при помощи обмена сообщениями по UDP-протоколу, третий поток контролирует происходящее в них путем анализа событий, генерируемых этими потоками и при необходимости перезапускает сервер и клиент. Клиент и сервер обмениваются сообщениями и при получении и отправке фиксируют соответствующие события в протоколе трассировки.

Использование UDP-протокола обеспечивает необходимую скорость работы системы, однако не дает гарантии успешной доставки всех сообщений. Для реализации механизма самостабилизации в данной системе контролирующий поток проверяет приходящие в протокол события. Если их последовательность не соответствует сигнатуре корректного поведения системы, осуществляет ее перезапуск.

Программа серверного потока представлена на листинге 2 (см. приложение). Программа клиента аналогична программе сервера с той разницей, что вначале проводится отправка сообщения, а затем делается попытка получить ответ от сервера.

Контролирующий поток, программа которого представлена на листинге 3 (см. приложение), создает потоки клиента и сервера и начинает обработку приходящих событий.

Заключение

Рассмотрены основные типы самоуправления программного обеспечения и некоторые примеры их применения. Применение комбинации этих методов на практике улучшает интерактивность, гибкость и стойкость программных систем, а также может являться решением задачи создания полностью автономной системы.

Рассмотрены средства стандартов POSIX и ARINC, позволяющие встраивать механизмы самолечения в программное обеспечение. Такие механизмы положены в основу самолечения и используются в отечественной операционной системе реального времени "Багет".

Стандарт ARINC предусматривает средства обнаружения и обработки ошибок на разных уровнях системы, в то время как средства трассировки в POSIX обеспечивают более гибкий контроль за выполнением программы путем анализа потока событиях.

Важными направлениями дальнейших исследований на обсуждаемом направлении являются: адаптация существующих методов для распределенных систем; ускорение реакции системы с помощью сохранения информации об уже происходивших событиях; создание систем прогнозирования ошибок по получаемым в реальном времени данным в целях их предотвращения.

Список литературы

1. **Horn P.** Autonomic computing: IBM's perspective on the state of information technology, IBM, October 2001.

2. **Bailey C., Montrieux L., De Lemos R., Yu Y., Wermelinger M.** Run-Time Generation, Transformation, and Verification of Access Control Models for Self-Protection // Proc. of 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. 2—3 June 2014. Hyderabad, India. URL: http://oro.open.ac.uk/39748/

3. **Tosi D.** Research Perspectives in Self-Healing Systems, Report of the University of Milano-Bieocca, 2004. http://www.cms.livjm.ac.uk/taleb/Lectures/cmssem003/Slides/lta.2004.06.pdf

4. **Годунов А. Н.** Операционная система реального времени Багет 3.0 // Программные продукты и системы. 2010. № 4. С. 15—19

5. **Laddaga R.** Creating robust software through self-adaptation// IEEE Intelligent Systems, 1999. № 14(3). P. 26—29.

6. **Igel C., Toussaint M.** Neutrality and Self-Adaptation. URL: http://ipvs.informatik.uni-stuttgart.de/mlr/marc/publications/03-igel-NatComp.pdf

7. **Gomez F., Burger D., Miikkulainen R.** A neuroevolution method for dynamic resource allocation on a chip multiprocessor// In Proceedings of the INNS-IEEE International Joint Conference on Neural Networks. IEEE. 2001. P. 2355—2361.

8. **Wildstrom J., Stone P., Witchel E., Mooney R.J., Dahlin M.** The Second International Conference on Autonomic Computing, June 2005. URL: http://www.cs.utexas.edu/~ml/papers/tuning-icac-05.pdf

9. **Demsky B., Rinard M.** Programming Languages Research Group URL: http://demsky.eecs.uci.edu/publications/sms03.pdf

10. **Sidiroglou S., Laadan O., Perez C.R., Viennot N., Nieh J., Keromytis A.D.** ASSURE: Automatic Software Self-healing Using REscue points. URL: http://people.csail.mit.edu/stelios/papers/assure_asplos.pdf

11. **Wimmer C., Würthinger T.** Truffle: A Self-Optimizing Runtime System, URL: http://www.christianwimmer.at/Publications/Wimmer12b/

12. **The Kilobot** project. URL: http://www.eecs.harvard.edu/ssr/projects/progSA/kilobot.html

13. **The Open** Group Base Specifications Issue 6. IEEE Std 1003.1, 2001.

Листинг 1. Пример системы с обработкой ошибок

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#include <string.h>
#include <sys/types.h>
#include <arinc/buffer.h>
#include <arinc/bbrd.h>
#include <arinc/event.h>
#include <arinc/partition.h>
#include <arinc/process.h>
#include <arinc/semaphore.h>
#include <arinc/sport.h>
#include <arinc/qport.h>
#include <arinc/error.h>

#define PROCESS_STACK_SIZE       4 * 4096          //Размер стека
#define PROCESS_PRIORITY    6                      // Приоритет тестового потока

PROCESS_ID_TYPE ProcId;                            //Идентификатор потока
#define MAXIMUM_LENGTH        64
static char *ErrorMessage = "Something Happened!";
/* Функция обработки ошибки */
int testErrorFunction(){
   RETURN_CODE_TYPE outReturnCode;       //Код возврата
   ERROR_STATUS_TYPE errorStatus;

   GET_ERROR_STATUS(&errorStatus, &outReturnCode);      //Получение информации об ошибке
   if (outReturnCode!= NO_ERROR)
   {

      printf("HandlerProc: false call!!! Return code %d\n", outReturnCode);
      return -1;
   }

   printf("ERROR CODE %d\n FAILED PROCESS ID %d\n", errorStatus.ERROR_CODE,
errorStatus.FAILED_PROCESS_ID);
   return 0;
}

/* Функция тестового потока */
int testCodeFunction(){

   RETURN_CODE_TYPE outReturnCode;

   srand (1);
   int random = rand() % 2;
   int res = 2 / random;                // Программная ошибка

                                        //Искусственный вызов
   RAISE_APPLICATION_ERROR(APPLICATION_ERROR,      // Тип ошибки
                     ErrorMessage,                 // Сообщение об ошибке
                     MAXIMUM_LENGTH,               // Его длина
                     &outReturnCode);
   if (outReturnCode!= NO_ERROR)
```

```
    {
      printf("TestProc: can't call Handler!!! Return code %d\n", outReturnCode);
      return -1;
    }


    return res;
}
int main()
{

    RETURN_CODE_TYPE ReturnCode;          //Код возврата

    printf("Test partition init started\n");
    /* Инициализация структуры атрибутов тестового потока */

    PROCESS_ATTRIBUTE_TYPE InAttribute = {
      "TestProcess",                                  // Имя потока управления
      (SYSTEM_ADDRESS_TYPE)&testCodeFunction,         // Адрес точки входа потока
      PROCESS_STACK_SIZE,                             // Размер стека потока
      PROCESS_PRIORITY,                               // Начальный приоритет потока
      INFINITE_TIME_VALUE,                            // Период потока
      INFINITE_TIME_VALUE,                            // Лимит времени работы потока
      SOFT                                            // Действия в случае превышения
                                                      // лимита времени
    };

    /* Порождение тестового процесса */
    CREATE_PROCESS (&InAttribute, &ProcId, &ReturnCode);
    if (ReturnCode!= NO_ERROR)
    {
      printf("TestProc: CREATE_PROCESS failed!!! Return code %d\n", ReturnCode);
      return -1;
    }
    CREATE_ERROR_HANDLER((SYSTEM_ADDRESS_TYPE)&testErrorFunction,
    PROCESS_STACK_SIZE,  &ReturnCode);
    if (ReturnCode!= NO_ERROR){
      printf("HandlerProc: CREATE_PROCESS failed!!! Return code %d\n", ReturnCode);
      return -1;
    }
    printf("Test partition init finished.\n");
    START (ProcId, &ReturnCode);              //Запуск рабочего процесса
    if (ReturnCode!= NO_ERROR)
    {
      printf("TestProc: START failed!!! Return code %d\n", ReturnCode);
      return -1;
    }

    SET_PARTITION_MODE (NORMAL, &ReturnCode);     //Установка режима работы процесса
    if (ReturnCode!= NO_ERROR)
    {

    printf ("TestProc: setting mode to NORMAL error %d\n", ReturnCode);
    return -1;
    }

    return 0;
}
```

Листинг 2. Пример самостабилизирующегося POSIX-процесса. Программа серверного потока

```
while (1){
  sleep(10);       /* Искусственная задержка */
  bytesRead = recvfrom(sock, recvData, 1024, 0, (struct sockaddr *)&clientAddr, &addrLen);
  /* Попытка получить сообщение */
  if(bytesRead == -1){
    printf("Server receiving error!!!\n");
    close(sock);
    return 7;
  }
   recvData[bytesRead] = '\0';
    posix_trace_event(serverRecEvent, recvData, strlen(recvData));   /* Отсылка в протокол информации о
том, что сервер успешно получил сообщение */
    posix_trace_event(serverSendEvent, answer, strlen(answer));       /* Отправка в протокол информации о
посылке сообщения */
    outCode = sendto(sock, answer, strlen(answer), 0, (struct sockaddr *)&clientAddr, sizeof(struct sockaddr));
/*Попытка отправить сообщение */
  if(outCode == -1){
    printf("Server sending error!!!\n");
    close(sock);
    return 8;
  }
  sched_yield();
}
```

Листинг 3. Пример самостабилизирующегося POSIX-процесса. Программа контролирующего потока

```
int state = 0;
int limit = 10;
while (limit > 0){
  curTime = time (NULL);
  const struct timespec timeout = {curTime, 500000};
  if(posix_trace_timedgetnext_event (testTrace, &event, recvData, 1024,
       &bytesRead, &outCode, &timeout)!= 0 || outCode!= 0)
  {   /*Попытка взять следующее событие из протокола */
    printf("Can't read events!!!\n");
    return 5;
  }
  if(event.posix_event_id == clientSendEvent && state == 0){
    printf("Client sent message\n»);
    state++;     /*Состояние, показывающее, какое событие ожидается дальше */
    /*Время с получения последнего ожидаемого события */
    importantTime = time (NULL);
  }
  else if(event.posix_event_id == serverRecEvent && state == 1){
    printf("Server received message\n");
    state++;
    importantTime = time (NULL);
  }
  else if(event.posix_event_id == serverSendEvent && state == 2){
    printf("Server sent message\n");
    state++;
    importantTime = time (NULL);
  }
  else if(event.posix_event_id == clientRecEvent && state == 3){
    printf("Client received message\n");
    state = 0;
```

```
        importantTime = time (NULL);
    }
    curTime = time(NULL);
    dif = difftime(curTime, importantTime);
    if(dif > 15)
    {    /*Если ожидаемое событие не получено
            в течение 15 секунд, рестарт системы */
        printf("Restarting system\n");
        outCode = pthread_create(&clientThread, &attributes, client, 0);
        if(0!= outCode){
            printf("Can't create new client thread!!! Error code: %d\n", outCode);
            return 6;
        }
        outCode = pthread_create(&serverThread, &attributes, server, 0);
        if(0!= outCode){
            printf("Can't create new server thread!!! Error code: %d\n", outCode);
            return 7;
        }
        importantTime = time(NULL);
        state = 0;
        limit --;
    }
    sched_yield();
}
```

A. A. Bombin, Engineer, **V. A. Galatenko,** Head of Department, **K. A. Kostiukhin**, Senior Researcher, e-mail: kost@niisi.msk.ru, Scientific Research Institute for System Analysis of the Russian Academy of Sciences (SRISA), Moscow

About Software Self-Healing in ARINC- and POSIX-compliant Systems

Attempts to solve the problem of complex systems management led to the idea of establishing mechanisms to help target system autonomously detect and handle errors, and also re-tune to reflect changing conditions. Thus, the new term "self-management software" was originated. This term includes the implementation of several "self-" functions (mechanisms), such as self-adaptation, self-defense, self-healing, and so on. Currently, these mechanisms are increasingly being used, and there are some researches on methods to implement them. This article discusses the existing mechanisms in the complex software systems self-management and describes self-healing tools of ARINC- and POSIX-compliant systems (such as the national real-time OS "Baget") developed with the participation of the authors.

Keywords: *self-healing, self-management, POSIX, ARINC*

References

1. **Horn P.** Autonomic computing: IBM's perspective on the state of information technology, IBM, October 2001.

2. **Bailey C., Montrieux L., De Lemos R., Yu Y., Wermelinger M.** Run-time generation, transformation, and verification of access control models for self-protection. *Proc. of 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2—3 June 2014. Hyderabad, India, available at: http://oro.open.ac.uk/39748/

3. **Tosi D.** Research Perspectives in Self-Healing Systems, Report of the University of Milano-Bieocca, 2004, available at:: http://www.cms.livjm.ac.uk/taleb/Lectures/cmssem003/Slides/lta.2004.06.pdf

4. **Godunov A. N.** *Programmnye produkty i sistemy*, 2010. no. 4, pp. 15—19 (in Russian).

5. **Laddaga R.** Creating robust software through self-adaptation. *IEEE Intelligent Systems,* 1999. no. 14(3), pp. 26—29.

6. **Igel C., Toussaint M.** Neutrality and Self-Adaptation, available at: http://ipvs.informatik.uni-stuttgart.de/mlr/marc/publications/03-igel-NatComp.pdf

7. **Gomez F., Burger D., Miikkulainen R.** A neuroevolution method for dynamic resource allocation on a chip multiprocessor. *In Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*. IEEE, 2001, pp. 2355—2361.

8. **Wildstrom J., Stone P., Witchel E., Mooney R. J., Dahlin M.** *The Second International Conference on Autonomic Computing,* June 2005, available at: http://www.cs.utexas.edu/~ml/papers/tuning-icac-05.pdf

9. **Demsky B., Rinard M.** Programming Languages Research Group, available at: http://demsky.eecs.uci.edu/publications/sms03.pdf

10. **Sidiroglou S., Laadan O., Perez C. R., Viennot N., Nieh J., Keromytis A. D.** ASSURE: Automatic Software Self-healing Using REscue points, available at: http://people.csail.mit.edu/stelios/papers/assure_asplos.pdf

11. **Wimmer C., Würthinger T.** Truffle: A Self-Optimizing Runtime System, available at: http://www.christianwimmer.at/Publications/Wimmer12b/

12. **The Kilobot** project, available at: http://www.eecs.harvard.edu/ssr/projects/progSA/kilobot.html

13. **The Open** Group Base Specifications Issue 6. IEEE Std 1003.1, 2001.

УДК 004.052.32

И. А. Пожилов, науч. сотр., e-mail: ilyapoz@gmail.com, **А. С. Семенов**, нач. сектора, e-mail: alxdr.semenov@gmail.com, **Д. В. Макагон**, нач. отдела, e-mail: makagond@nicevt.ru, ОАО "Научно-исследовательский центр электронной вычислительной техники", Москва

Алгоритм определения связности сети с топологией "многомерный тор" с отказами для детерминированной маршрутизации

Рассмотрена задача определения множества доступных путей в сети с топологией "многомерный тор" при наличии отказавших каналов связи между узлами. Предложен универсальный алгоритм ее решения для произвольной маршрутизации. Исследован показатель вероятности потери связности для неминимальной маршрутизации, метода First Step/Last Step и полной маршрутизации. Представлены оценки эффективности метода First Step/Last Step для обеспечения связности.

Ключевые слова: *отказоустойчивость, коммуникационные сети, многомерный тор, связность, детерминированная маршрутизация, маршрутизация с порядком направлений*

Введение

Современные суперкомпьютерные системы включают в себя большую совокупность элементов: процессоры, модули памяти, сетевые устройства, устройства ввода-вывода, связанные в единый программно-аппаратный комплекс. Каждый из этих элементов независимо от другого обладает высоким уровнем надежности, однако с ростом числа элементов система в целом обнаруживает тенденцию к появлению неисправностей.

В маршрутизаторе сети "Ангара" [1, 2] реализована пакетная передача данных по нескольким виртуальным подсетям, что позволяет логически разделить потоки данных и поддержать одновременно несколько различных алгоритмов маршрутизации. Основная задача маршрутизации — доставка пакетов корректным адресатам без создания в сети тупиковых ситуаций (например, взаимных блокировок *deadlocks*). Различные алгоритмы маршрутизации имеют свои преимущества: одни минимизируют длину проходимых пакетами путей, другие обеспечивают более равномерную загрузку сети в целом, третьи поддерживают "размножение" пакетов (например, при широковещательной рассылке). В сети "Ангара" поддерживается бездедлоковая детерминированная маршрутизация, основанная на правилах "пузырька" (*Bubble flow control* [3]) и "порядка направлений" (*Direction ordered routing*, DOR [4, 5]). Данная маршрутизация позволяет избежать взаимных блокировок вследствие циклических зависимостей пакетов в кольцах и между кольцами нескольких измерений, а также гарантирует сохранение порядка передачи пакетов между любыми двумя адресатами. С помощью дополнительных виртуальных каналов в сети "Ангара" поддерживается адаптивная маршрутизация, обеспечивающая более равномерную загрузку сети и коллективные операции по дереву, наложенному на многомерный тор. Аппаратно поддерживается обход отказавших узлов с помощью отклонения от кратчайших путей благодаря алгоритму *First Step/Last Step* — нестандартного первого и последнего шага [5].

Одной из задач обеспечения отказоустойчивости является определение множества доступных путей в коммуникационной сети в обход отказавших каналов связи (задача определения связности). В данной работе рассматривается задача определения связности для детерминированной маршрутизации по порядку направлений.

В разд. 1 приведены необходимые формальные определения. В разд. 2 представлена постановка задачи определения связности в общем виде. В разд. 3 рассмотрены два алгоритма решения задачи определения связности, приведены оценки их эффективности и области применимости. В разд. 4 дано формальное описание алгоритмов маршрутизации, которые исследованы в разд. 5.

1. Определения

В настоящем разделе введем некоторые формальные определения, которые будем использовать далее.

Рассмотрим вычислительную сеть с топологией "многомерный тор". Пусть размерности тора имеют

значения (d_1, d_2, ..., d_n). Общее число узлов сети обозначим N_{nodes}. В этой сети узел u имеет координаты $u = (u_1, u_2, ..., u_n)$, где $0 \leqslant u_j < d_j$. Множество всех узлов сети обозначим $\mathcal{N}$.

Соседними в рамках тороидальной топологии будут узлы $u = (u_1, u_2, ..., u_n)$ и $\tilde{u} = (u_1, ..., (u_j \pm 1) \bmod d_j, ..., u_n)$ для любого индекса $1 \leqslant j \leqslant n$.

Легко заметить, что каждый узел имеет $2n$ соседей (в случае $d_j > 2$). Будем считать, что каждый из этих соседей находится в одном из *направлений* от узла u.

Множество направлений обозначим

$$\mathcal{D} = \{\tilde{u} - u | \tilde{u},\ u \in \mathcal{N},\ \tilde{u} \text{ — соседний узел к } u\}$$

и пронумеруем их числами $\overline{1,\ 2n}$

$$\mathcal{D} = \left\{\Delta_j\right\}_{\overline{j=1,\ 2n}}.$$

Направления с номерами 1, ..., n будем называть *положительными*:

$$\tilde{u} = (u_1, ..., (u_j + 1) \bmod d_j, ..., u_n),$$

$$\Delta_j = \left(0, ..., \underbrace{1}_{\text{позиция } j}, ..., 0\right) \in \mathcal{D},$$

если $1 \leqslant j \leqslant n$.

Направления $n + 1$, ..., $2n$ будем называть *отрицательными*:

$$\tilde{u} = (u_1, ..., (u_{j-n} - 1) \bmod d_{j-n}, ..., u_n),$$

$$\Delta_j = \left(0, ..., \underbrace{-1}_{\text{позиция } j-n}, ..., 0\right) \in \mathcal{D},$$

если $n + 1 \leqslant j \leqslant 2n$.

В такой формулировке выражение "узел u находится в направлении D от узла v" записывается как $u = v + D$.

На множестве направлений $\mathcal{D}$ введем порядок в соответствии с указанной нумерацией: $\Delta_i < \Delta_j$, если $i < j$.

Введем операцию $s(d)$: $\Delta_j \to \Delta_{j+1}$, $j = \overline{1,\ 2n-1}$, которая позволяет получить следующее направление в последовательности.

Множество положительных и отрицательных направлений обозначим соответственно $\mathcal{D}_+$ и $\mathcal{D}_-$. Индекс вектора координат, который изменяется в данном направлении, обозначим $|D| \in \overline{1,\ n}$ [1].

Определение 1. *Путем* P в сети назовем последовательность вида u^0, D_1, u^1, D_2, ..., D_N, u^N, где u^j — узел вычислительной сети; D_j — направление, связывающее вершину u^{j-1} с вершиной u^j.

Множество всех путей обозначим $\mathcal{P}$.

Иногда будем опускать указание промежуточных вершин u^j, так как они могут быть получены из направлений соответствующих переходов: $u^j = u^{j-1} + D_j$. Таким образом,

$$u^0,\ D_1,\ u^1,\ D_2,\ ...,\ D_N,\ u^N = u^0,\ D_1,\ D_2,\ ...,\ D_N,\ u^N.$$

Среди алгоритмов маршрутизации для многомерных торов можно выделить класс алгоритмов, соблюдающих *правило порядка направлений*: маршрут между любой парой узлов включает движения в направлениях в определенном заранее заданном порядке. Эти алгоритмы обладают свойством отсутствия взаимных блокировок при любом числе одновременных запросов на передачу данных по сети [5].

Заметим, что правило порядка направлений для детерминированной маршрутизации в данном случае формулируется как $D_{j-1} \leqslant D_j$, $j = \overline{2, N}$. Чтобы задать путь, удовлетворяющий правилу порядка направлений, достаточно задать стартовую вершину и число шагов в каждом из направлений, т. е. набор v_0, s_1, s_2, ..., s_{2n}, где $v_0 \in \mathcal{N}$ — стартовый узел; s_1, s_2, ..., $s_{2n} \geqslant 0$ — число шагов в направлениях Δ_1, Δ_2, ..., Δ_{2n}.

Этот набор задает путь

$$v_0,\ \underbrace{\Delta_1, ..., \Delta_1}_{s_1 \text{ раз}},\ \underbrace{\Delta_2, ..., \Delta_2}_{s_2 \text{ раз}},\ ...,\ \underbrace{\Delta_{2n}, ..., \Delta_{2n}}_{s_{2n} \text{ раз}},\ u \in \mathcal{P}.$$

Обозначим $\mathcal{P}_D(u^0, u^N)$ множество путей, удовлетворяющих правилу порядка направлений.

2. Постановка задачи определения связности

Определение 2. Каналом связи сети будем называть пару (u, D), где $u \in \mathcal{N}$, $D \in \mathcal{D}$. Множество всех каналов связи обозначим $\varepsilon := \mathcal{N} \times \mathcal{D}$.

Среди всех возможных каналов связи определим *множество отказавших каналов* $F \subset \varepsilon$.

Так как канал связи между двумя узлами функционирует согласно двустороннему протоколу, то разумно предположить, что каналы связи между соседними узлами выходят из строя одновременно, т. е. множество F будет включать в себя отказавшие каналы связи попарно. В соответствии с этим обстоятельством будем считать число отказавших каналов как $N_F = |F|/2$.

Определение 3. Путь $P = u^0$, D_1, ..., D_N, u^N назовем *допустимым*, если ни один из переходов u^{j-1}, D_j, u^j не осуществляется по одному из отказавших каналов, т. е.

$$\forall j \in \overline{1, N} \Rightarrow \left(u^{j-1}, D_j\right) \notin F.$$

Множество допустимых путей обозначим $\mathcal{G}_F(u^0, u^N)$.

Пусть для каждой (упорядоченной) пары узлов (u, v) известно множество путей, которые соединяют их согласно *маршрутизации*: R:$(u, v) \to 2^{P_D(u,v)}$, где $2^{P_D(u,v)}$ множество всех подмножеств множества допустимых путей $P_D(u, v)$.

Определение 4. Пару узлов (u, v) назовем *связанными*, если существует допустимый путь из узла u в узел v согласно правилам маршрутизации:

[1] Математически $|\Delta_j| \equiv j \bmod n$.

$$R(u, v) \cap \mathcal{G}_F(u, v) \neq \varnothing.$$

Определение 5. *Задача определения связности* заключается в нахождении множеств

$$R(u, v) \cap \mathcal{G}_F(u, v),$$

т. е. в определении связности каждой пары узлов и нахождении допустимых путей между ними.

3. Алгоритмы решения задачи определения связности

Рассмотрим два алгоритма решения задачи определения связности. Первый, переборный алгоритм, решает эту задачу в общем виде для любой маршрутизации $R(u, v)$, удовлетворяющей правилу порядка направлений. Второй алгоритм решает задачу определения связности для *полной маршрутизации*, т. е. маршрутизации, включающей все пути, удовлетворяющие правилу порядка направлений.

Переборный алгоритм

Пусть маршрутизация $R(u, v)$ задана в виде множества наборов $v_0, s_1, s_2, ..., s_{2n}$. Каждый набор задает путь между парой вершин, удовлетворяющий правилу порядка направлений, согласно определению из разд. 1. Переборный алгоритм заключается в проверке принадлежности каждого из отказавших каналов связи путям, задаваемым множеством наборов $v_0, s_1, s_2, ..., s_{2n}$.

Приведем асимптотическую оценку вычислительной сложности алгоритма при росте N_{nodes}. Допустим, что число различных путей между двумя вершинами ограничено константой N_{paths}:

$$\forall u, v \Rightarrow \left|R(u,v)\right| \leqslant N_{paths}.$$

Необходимо проверить все пути между всеми парами вершин на наличие отказавших каналов в них. Для проверки одного пути между двумя узлами необходимо проверить попадание отказавшего канала связи в один из отрезков пути длиной $s_1, s_2, ..., s_{2n}$.

Проверка попадания отказавшего канала в отрезок выполняется за $O(n)$ операций сравнения. Чтобы проверить попадание всех отказавших каналов, необходимо $O(n|F|)$ операций. Таким образом, сложность алгоритма оценивается как

$$T = O\left(N_{nodes}^2 N_{paths} n|F|\right) = O\left(N_{nodes}^2 N_{paths}|F|\right).$$

Здесь предполагаем, что $n = O(1) \ll N_{nodes}$, т. е. число измерений тора ограничено некоторой небольшой константой и мало́ по сравнению с числом узлов.

Для некоторых алгоритмов маршрутизации число путей между парой вершин $N_{paths} = O(n)$, т. е. $N_{paths} \ll \ll N_{nodes}$, откуда для этих алгоритмов получаем

$$T = O\left(N_{nodes}^2 |F|\right). \quad (1)$$

Однако не все маршрутизации обладают этим свойством. Если число путей увеличивается с ростом числа узлов, то необходим другой алгоритм решения задачи определения связности.

Алгоритм для полной маршрутизации

В случае *полной маршрутизации* число путей между двумя вершинами будет увеличиваться с ростом числа узлов, что приведет к изменению оценки (1). В этом подразделе рассмотрим алгоритм, решающий задачу определения связности в случае, если маршрутизация $R(u, v)$ включает в себя *все* пути, удовлетворяющие правилу порядка направлений:

$$R(u, v) = \mathcal{P}_D(u, v).$$

Решение задачи определения связности в этом случае сведем к решению задачи о существовании пути в некотором ориентированном графе $G(V, E)$. Построим граф G.

Множество вершин графа G имеет следующий вид:

$$V := \mathcal{N} \times \mathcal{D}.$$

Здесь вершины (u, d_0) соответствуют узлам u, путь до которых включает движение только по направлениям $\Delta_1 \leqslant d \leqslant d_0$.

Множество дуг включает дуги двух видов:

$$E := A \cup B.$$

Здесь A — множество дуг, отвечающих каналам связи вдоль определенного направления: $A := \{((u, d), (u + d, d))|\ u \in \mathcal{N}, d \in \mathcal{D}, (u, u + d) \notin F\}$, где F — множество отказавших каналов связи. B — множество *тривиальных дуг*, отвечающих переходу к следующему направлению без изменения текущей вершины:

$$B := \left\{\left((u,d),\left(u,s(d)\right)\right)\middle|u \in \mathcal{N},\ \ d \in \overline{\Delta_1, \Delta_{2n-1}}\right\}.$$

Корректность сведения задачи определения связности сети к задаче поиска пути в графе основывается на следующем утверждении.

Утверждение. Путь из узла $u^0 \in \mathcal{N}$ в узел $v^0 \in \mathcal{N}$, удовлетворяющий правилу порядка направлений и не содержащий отказавших узлов, существует в том и только в том случае, если в графе G существует путь из вершины (u^0, Δ_1) в вершину (v_0, Δ_{2n}).

Доказательство. Для доказательства приведем взаимно-однозначное соответствие между множеством путей в сети согласно определению 1 и множеством путей в графе G.

Рассмотрим путь

$$P = u^0, D_1, u^1, D_2, ..., D_N, u^N = v^0,$$

где $D_{j-1} \leqslant D_j,\ j = \overline{2, N}$.

Построим соответствующий путь в графе G:

$$\left(u^0, \Delta_1\right),\ ...,\ \left(u^0, D_1\right), \left(u^1, D_1\right),\ ...,\ \left(u^1, D_2\right), \left(u^2, D_2\right),$$
$$...,$$
$$\left(u^{N-1}, D_N\right), \left(u^N, D_N\right),\ ...,\ \left(u^N, \Delta_{2n}\right).$$

Аналогичным образом можно по данному пути в графе G построить путь в сети согласно определению 1. Это соответствие показывает, что существование пути в сети из u^0 в v^0 равносильно существованию пути в графе G из (u^0, Δ_1) в (v^0, Δ_{2n}).

Заметим, что в указанном построении одному каналу связи сети соответствует единственное ребро графа G. Это обстоятельство позволяет свести задачу определения связности в сети с порядком направлений к задаче поиска пути в графе.

Для решения задачи определения связности с помощью графа G применим метод поиска вширь: для каждой из вершин (u^0, Δ_1) определим множество достижимых из нее вершин вида (v, Δ_{2n}). При этом соответствующие узлы v будут достижимы из u^0 согласно доказанному утверждению.

Оценим трудоемкость указанного алгоритма. Число вершин графа G равно $|V| = |\mathcal{N}|2n = N_{nodes}2n$. Число ребер $|E| \leqslant 2|V|$, так как $|A| = |V|$, $|B| \leqslant |V|$. Алгоритм поиска вширь имеет сложность $O(V + E) = O(nN_{nodes})$. Однако с учетом того, что требуется N_{nodes} запусков поиска вширь, общая сложность решения задачи определения связности $T = O\left(nN_{nodes}^2\right)$.

Если применить предположение, которое было сделано для первого алгоритма, а именно $n \ll N_{nodes}$, то можно окончательно записать:

$$T = O\left(N_{nodes}^2\right).$$

Отсюда следует, что вычислительная сложность для полной маршрутизации практически аналогична оценке (1). Остановимся на этом факте подробнее. Напомним, что при построении оценки (1) было сделано предположение о малом числе возможных путей между парой вершин. В случае полной маршрутизации это предположение не выполняется, поэтому решение задачи определения связности первым алгоритмом в данном случае имело бы сложность порядка $O\left(N_{nodes}^3\right)$. Однако путем сведения задачи к задаче определения связности на графе удалось применить алгоритм поиска вширь, что и позволило получить более выгодную оценку сложности.

4. Исследуемые алгоритмы маршрутизации

В задаче определения связности важную роль играет отображение $R(u, v)$, показывающее множество путей, которые данный алгоритм маршрутизации позволяет построить. Если имеются маршрутизации R_1 и R_2, такие что $R_1(u, v) \subset R_2(u, v)$, то маршрутизация R_2 позволяет, очевидно, строить путь в обход большего числа отказов.

В настоящем разделе рассмотрим два алгоритма маршрутизации $R(u, v)$, а именно — *dirbit*-маршрутизацию и метод *First Step/Last Step*. С помощью построенного алгоритма проведено исследование показателя, связанного с отказоустойчивостью рассматриваемой сети: вероятность потери связности при случайном выборе множества отказавших каналов связи.

Маршрутизация *dirbit*

Маршрутизация с использованием битов направлений предложена в работе [4]. Приведем здесь формальное описание этого алгоритма.

Пусть узел-отправитель имеет координаты $s = (s_1, ..., s_n)$, узел-получатель $t = (t_1, ..., t_n)$. *Dirbit*-маршрутизация является *покоординатной*: маршрут следования выравнивает координаты узлов в определенном порядке. Пусть задано подмножество направлений $D \subseteq \mathcal{D}$, причем выполнено $d \in D \Rightarrow -d \notin D$, т. е. D не включает в себя противоположных направлений.

Тогда путь P из s в t будет выглядеть следующим образом (здесь использована сокращенная запись пути, указаны не все промежуточные вершины):

$$P(D) = u^0 = s,$$

$$D_1, D_1, \ ..., \ D_1, u^1 = \left(s_1, s_2, \ ..., \ t_{|D_1|}, \ ..., \ s_n\right),$$

$$D_2, D_2, \ ..., \ D_2, u^2 = \left(s_1, \ ..., \ t_{|D_1|}, \ ..., \ t_{|D_2|}, \ ..., \ t_n\right),$$

$$...$$

$$D_{N-1}, \ ..., \ D_{n-1}, u^{N-1} = \left(t_1, t_2, \ ..., \ s_{|D_N|}, \ ..., \ t_N\right),$$

$$D_N, D_N, \ ..., \ D_N, u^N = t,$$

где $D_i \in D$, $D_{i-1} < D_i$. Откуда следует, что критерий существования этого пути

$$\left\{|d| \mid d \in D\right\} \supseteq \left\{i \mid s_i \neq t_i\right\},$$

т. е. множество D должно включать в себя направления, соответствующие отличающимся координатам узлов отправки и назначения.

Число возможных *dirbit*-путей из s в t представляется в виде

$$N_{paths}(s,t) = 2^{\left|\{i|s_i \neq t_i\}\right|} \leqslant 2^n.$$

В этом выражении для каждого индекса i, различающихся координат в s и t возможно составление двух маршрутов: один будет включать движение в направлении Δ_i, другой — в противоположном направлении. Отсюда видно, что число путей $N_{paths} \ll N_{nodes}$, а значит, построенный переборный алгоритм будет эффективно применим к этой маршрутизации.

Метод *First Step/Last Step*

Метод *First Step/Last Step* был предложен в работе [5] как механизм обхода отказавших узлов, а также для маршрутизации в необычных конфигурациях сети.

Добавление нестандартного первого шага и нестандартного последнего шага позволяет расширить множество возможных путей маршрутизации. Если $R_{0(u, v)}$ — множество путей, полученных при помощи некоторого другого алгоритма (например, *dirbit*-маршрутизации), то добавляя нестандартный первый и последний шаги (FS и LS), можно получить маршрутизацию $R_{FL}(u, v)$ в следующем виде:

$$\tilde{R}_{FL}(u, v) = R_0(u, v) \cup \{u, FS, p, LS, v \mid FS \in \mathcal{D},$$
$$LS \in \mathcal{D},\ p \in R_0(u + FS, v - LS)\}.$$

Такое описание означает: один шаг из u в направлении FS, затем произвольный путь из $u + FS$ в $v - LS$, затем один шаг из $v - LS$ в v. Под $u + FS$ понимается узел, соседний к u в направлении FS, под $v - LS$ понимается узел, соседний в направлении, *противоположном* LS.

Из представленной выше маршрутизации необходимо исключить пути, не удовлетворяющие правилу порядка направлений:

$$R_{FL}(u, v) = \tilde{R}_{FL}(u, v) \cap \mathcal{P}_D.$$

Далее будет рассмотрена задача определения связности для маршрутизаций R, R_0 и R_{FL}, где R — полная маршрутизация; R_0 — *dirbit*-маршрутизация.

5. Оценка отказоустойчивости маршрутизации

Пусть F — некоторое множество отказавших каналов связи, а Ω — множество всевозможных комбинаций отказавших каналов связи. На этом множестве рассмотрим отображение

$$\xi : \Omega \to \{0, 1\},$$

$$\xi(F) = \begin{cases} 1, & \text{если } \exists u, v \Rightarrow R(u, v) \cap \mathcal{G}_F(u, v) = \varnothing, \\ 0 & \text{в противном случае.} \end{cases}$$

Пусть известно, что в сети присутствуют N_F отказавших каналов связи. Можем рассматривать множество Ω, как множество элементарных исходов конечного вероятностного пространства. Тогда $\xi(F)$ можно считать случайной величиной, определенной на этом пространстве.

Определим вероятность потери связности для заданной маршрутизации R:

$$P_F(N_F) = \mathbb{P}(\xi(F) = 1 \mid |F| = 2N_F) = \\ = \mathbb{E}(\xi \mid |F| = 2N_F) = a,$$

где $\mathbb{P}$ и $\mathbb{E}$ — вероятность и математическое ожидание соответственно. Таким образом, P_F — вероятность появления хотя бы одной пары узлов, между которыми не существует допустимого пути.

Для нахождения зависимости $P_F(N_F)$ применим метод Монте-Карло: будем генерировать множества F заданной мощности N_F и определять наличие путей между всеми парами вершин:

$$P_F(N_F) \approx \frac{\left|\left\{\exists u, v \Rightarrow R(u, v) \cap \mathcal{G}(u, v) = \varnothing \mid F \in \mathcal{F}\right\}\right|}{N_{iter}} = \overline{X},$$

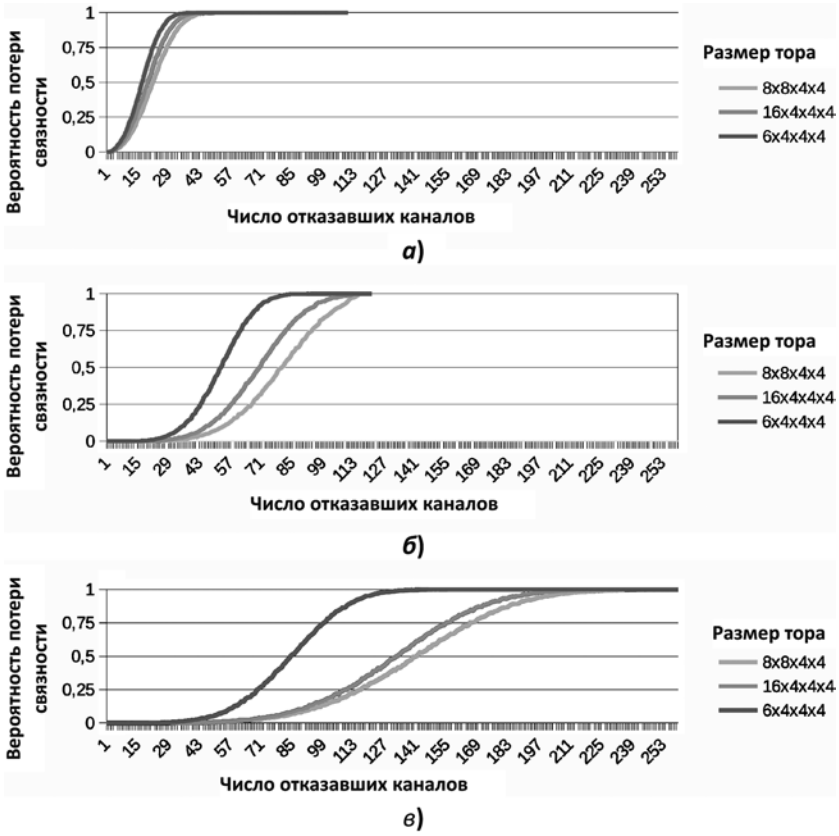


Рис. 1. Вероятность потери связности на примере 4D-торов:
а — *dirbit*-маршрутизация; *б* — метод *First Step/Last Step*; *в* — полная маршрутизация

где $\mathcal{F}$ — набор случайно сгенерированных множеств отказавших каналов связи; X — выборка из случайной величины ξ объемом N_{iter},

$$|\mathcal{F}| = N_{iter},$$

$$|F| = 2N_F,\ F \in \mathcal{F}.$$

Оценивать отказоустойчивость алгоритмов маршрутизации будем на основе вероятности потери связности.

Доверительный интервал $\left(\overline{X} - \delta, \overline{X} + \delta\right)$ покрывает неизвестную вероятность $P_F(N_F)$ с заданным уровнем доверия $\alpha = 0{,}999$ при $d \cong 0{,}026$, $N_{iter} = 1000$.

На рис. 1 и 2 представлены графики зависимости $P_F(N_F)$, построенные методом Монте-Карло для $N_{iter} = 1000$. Анализируя эти зависимости, можно сформулировать следующие выводы.

- Полная маршрутизация задает теоретический барьер для задачи определения связности. Она предполагает все возможные пути между парой узлов, удовлетворяющие правилу порядка направлений. Таким образом, увеличение отказоустойчивости за границы полной маршрутизации возможно только при помощи принципиально отличных методов маршрутизации, например *Turn model routing* [6].

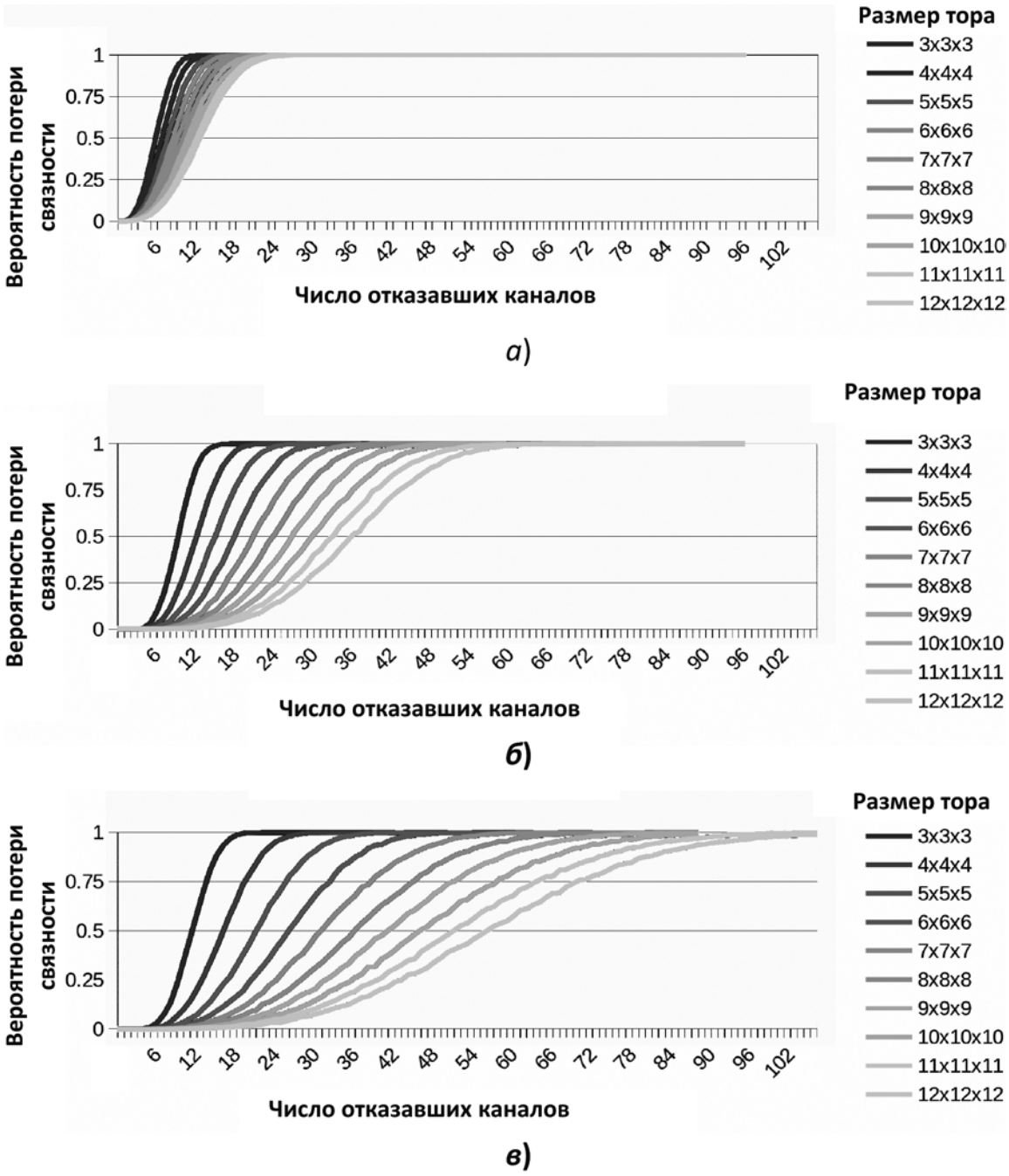


Рис. 2. Вероятность потери связности на примере равносторонних 3D-торов:
а — *dirbit*-маршрутизация; *б* — метод *First Step/Last Step*; *в* — полная маршрутизация

- Даже при малом числе отказов наблюдается появление ненулевой вероятности потери связности. Для *dirbit*-маршрутизации этот факт объясняется тем, что два отказавших канала в одном кольце тора приводят к отсутствию связности между некоторыми узлами в этом кольце.
- Маршрутизация с применением нестандартных шагов позволяет заметно снизить вероятность потери связности, так как вводит дополнительные пути. На рис. 1 видно, что для тора размером 8×8×4×4 вероятность 1/2 для *dirbit*-маршрутизации достигается при числе отказов каналов связи 22, а для маршрутизации с использованием метода *First Step/Last Step* — при числе отказов 81. Примерно такое же соотношение верно для всех остальных значений вероятности. Для торов меньших размеров видно,

что эффективность метода *First Step/Last Step* не столь высока: одинаковый уровень вероятности достигается на значительно более близких количествах отказавших каналов.

Заключение

Построен алгоритм решения задачи определения связности, т. е. выяснения возможности доставки сообщений по коммуникационной сети в обход отказавших каналов связи для детерминированной маршрутизации в сети с топологией "многомерный тор". В ходе исследований вероятности потери связности для некоторых конфигураций сети показана эффективность метода *First Step/Last Step*. Этот метод позволяет увеличить число отказов каналов связи при сохранении прежней вероятности потери связности. С помощью алгоритма для решения задачи определения связности в случае полной маршрутизации удается сравнить отказоустойчивость *dirbit*-маршрутизации и метода *First Step/Last Step* с теоретически максимальным результатом, который возможен для маршрутизации, удовлетворяющей правилу порядка направлений. По результатам исследований показано, что потенциал отказоустойчивости такого способа маршрутизации выше того, который достигается при помощи метода *First Step/Last Step*.

Список литературы

1. **Корж А. А., Макагон Д. В., Жабин И. А., Сыромятников Е. Л.** Отечественная коммуникационная сеть 3D-тор с поддержкой глобально адресуемой памяти для суперкомпьютеров транспетафлопсного уровня производительности // Параллельные вычислительные технологии (ПаВТ'2010): Труды междунар. конф. Уфа, 29 марта — 2 апреля 2010. Челябинск: Издательский центр ЮУрГУ, 2010. С. 227—237.
2. **Жабин И., Макагон Д. Симонов А. и др.** Кристалл для Ангары // Суперкомпьютеры. 2013. Т. зима-2013. С. 46—49.
3. **Puente V., Beivide R., Gregorio J. A. et al.** Adaptive bubble router: a design to improve performance in torus networks // Parallel Processing, 1999. Proceedings. 1999 International Conference on. P. 58—67.
4. **Adiga N. R., Blumrich M. A., Chen D. et al.** Blue Gene/L torus interconnection network // IBM Journal of Research and Development. 2005. Vol. 49, N. 2.3. P. 265—276.
5. **Scott S. L., Thorson G. M**. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus // HOT Interconnects IV. Proceeding. 1996. P. 147—155.
6. **Glass C., Ni L.** The turn model for adaptive routing // Computer Architecture, 1992. Proceedings, The 19th Annual International Symposium on. 1992. P. 278—287.

I. A. Pozhilov, Researcher, e-mail: ilyapoz@gmail.com, **A. S. Semenov,** Head of Sector, e-mail: alxdr.Semenov@gmail.com,
D. V. Makagon, Head of Department, e-mail: makagond@nicevt.ru,
Joint-stock company "Scientific Research Centre for Electronic Computer Technology", Moscow

Connectivity Problem Solution for Direction Ordered Deterministic Routing in nD Torus

Connectivity problem for nD torus with link failures is addressed. An algorithm is proposed to solve the problem in the general case. Connectivity loss probability for given number of link failures examined for non-minimal direction ordered routing, First Step/Last Step method and the complete routing algorithms. First Step/Last Step is shown to be adequate for maintaining connectivity.

Keywords: *fault tolerance, communication networks, multidimensional torus, connectivity, deterministic routing, direction ordered routing*

References

1. **Korzh A. A., Makagon D. V., Zhabin I. A., Syromjatnikov E. L.** *Parallel'nye vychislitel'nye tehnologii (PaVT'2010): Trudy mezhdunar. konf.,* 2010, Cheljabinsk: Izdatel'skij centr JuUrGU, pp. 227—237 (in Russian).
2. **Zhabin I., Makagon D., Simonov A., Syromyatnikov E., Frolov A., Sherbak A.** *Superkomp'jutery*, 2013, vol. zima-2013, pp. 46—49 (in Russian).
3. **Puente V., Beivide R., Gregorio J. A., Prellezo J. M., Duato J., Izu C.** Adaptive bubble router: a design to improve performance in torus networks. *Parallel Processing, 1999. Proceedings.* 1999 International Conference on, 1999, pp. 58—67.
4. **Adiga N. R., Blumrich M. A., Chen D., Coteus P., Gara A., Giampapa M., Heidelberger P., Singh S., Steinmacher-Burow B. D., Takken T., Tsao M., Vranas P.** Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 2005, vol. 49, no. 2.3, pp. 265—276.
5. **Scott S. L., Thorson G. M.** The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. *HOT Interconnects IV. Proceedings*, 1996, pp. 147—155.
6. **Glass C., Ni L.** The turn model for adaptive routing. *Computer Architecture, 1992. Proceedings,* The 19th Annual International Symposium on. 1992, pp. 278—287.

УДК 004.5

Я. А. Туровский, канд. мед. наук, зав. лаб., e-mail: yaroslav_turovsk@mail.ru,
С. Д. Кургалин, д-р физ.-мат. наук, зав. каф., **А. А. Вахтин,** канд. физ.-мат. наук, доц.,
В. А. Белобродский, аспирант, Воронежский государственный университет

Человеко-машинный интерфейс, учитывающий функциональное напряжение человека

Предложен вариант интерфейса "человек-компьютер", основанный на регистрации биомедицинского сигнала вариабельности сердечного ритма (изменения частоты сердечных сокращений во времени), изменяющегося при функциональном напряжении человека. Этот сигнал анализируется как во временной, так и в частотной областях, с использованием непрерывного вейвлет-преобразования. Результаты анализа используются при формировании команд интерфейса "человек-компьютер" для управления работой операционных систем или программных приложений.

Ключевые слова: *интерфейс человек-компьютер, биомедицинские сигналы, цифровая обработка сигналов, управление, вейвлет-преобразование, функциональное напряжение, вариабельность сердечного ритма*

Введение

Активное развитие в последние десятилетия как информационно-коммуникационных технологий, так и микроэлектронной техники привело к значительному расширению возможностей существующих интерфейсов "человек-компьютер". Кроме традиционных, ставших уже классическими, интерфейсов, работающих на основе клавиатуры, джойстика или мыши, появился ряд программно-аппаратных решений [1—4], использующих альтернативные подходы к формированию команд пользователя, подаваемых компьютеру. К таковым можно отнести интерфейсы, использующие разнообразные биомедицинские сигналы: окулодвигательные (возникающие при изменении направления взгляда за счет изменения положения глазного яблока), электромиографические (получаемые при активности мышц), нейрокомпьютерные (получаемые из электроэнцефалографов или магнитно-резонансных томографов), стабилографические (возникающие при изменении положения тела человека в пространстве) и др. Однако приходится констатировать, что значительная их часть не нашла широкого применения. Причина такого положения дел заключается в том, что каждый из этих интерфейсов обладает одним или несколькими недостатками из следующего списка: небольшая скорость передачи команд от пользователя компьютеру; относительно низкая точность работы; высокая стоимость; наличие достаточно строгих требований к процедурам обучения классификаторов команд, используемых в интерфейсах, а также рядом других недостатков. Отметим при этом, что очень мало внимания разработчики уделяют интерфейсам, использующим для целей управления команды, генерируемые человеком непроизвольно и/или неосознанно. Существуют только единичные работы на эту тему [5, 6] на фоне значительного числа публикаций, относящихся к области конструирования других типов интерфейсов. Это свидетельствует об определенном отставании разработок этой группы интерфейсов от создания интерфейсов, использующих в управлении команды, которые осознанно вырабатываются человеком.

Рассматривая технологии человеко-машинных интерфейсов, альтернативных классическим, заметим, что имеется значительный объем данных, несущих информацию о состоянии человека, его эмоциональной сфере и о других характеристиках, известных как “невербальная активность". Сведения о подобной активности являются важным элементом в процессе общения людей. Однако они, как правило, не учитываются при организации процесса взаимодействия человека с компьютером. В существующих аппаратно-программных средствах, предназначенных для обеспечения человеко-машинного общения, в подавляющем большинстве случаев пока еще отсутствуют механизмы, оценивающие невербальную активность пользователя. Такой вид активности включает в себя и эмоциональную сферу человека, связанную с так называемым *функциональным напряжением.* Данное состояние находит отражение в маркерах активности вегетативной нервной системы. К таким маркерам можно отнести вариабельность сердечного ритма (ВСР) — изменение частоты сердечных сокращений (пульса) во времени;

кожно-гальваническую реакцию — изменение электрофизиологических параметров кожи (как правило, электрического сопротивления); параметры дыхательных движений — изменение частоты дыхания, объема выдыхаемого воздуха и др. Безусловным преимуществом при разработке методов генерации управляющих команд для компьютера, обусловленных функциональным напряжением, обладает сигнал ВСР, как наиболее изученный из указанных выше сигналов. Важно отметить, что для регистрации этого сигнала существуют широко известные на практике аппаратно-программные решения. К таковым можно отнести программно-аппаратные средства, анализирующие временные интервалы между *R*-зубцами электрокардиограммы (*RR*-интервалы). Другим решением являются фотоплетизмографы, где пик временной зависимости кровенаполнения сосуда (так называемой *пульсовой волны*) соответствует максимальному его значению, порождаемому сокращением сердца. Следовательно, с помощью значения интервала пик-пик фотоплетизмограммы можно определить, с определенной поправкой, значение частоты сердечных сокращений.

Полученные таким образом данные и будут после их обработки трансформироваться интерфейсом "человек-компьютер" в управляющие команды. В контексте настоящей работы такие интерфейсы будем называть *интерфейсы, зависящие от функционального напряжения человека*.

Целью исследования, результаты которого представлены в настоящей работе, является разработка варианта интерфейса "человек-компьютер", основанного на регистрации биомедицинского сигнала вариабельности сердечного ритма, изменяющегося при функциональном напряжении человека, а также демонстрация возможности его применения для формирования команд управления работой операционных систем или программных приложений.

Характеристики интерфейса, зависящего от функционального напряжения человека

Рассмотрим элементы кривой ВСР (т. е. зависимость частоты пульса от времени), получаемой в виде временного ряда, состоящего из значений частот сердечных сокращений. Целью обработки кривой ВСР является извлечение из нее информации о функциональном напряжении человека для формирования команд человеко-машинного интерфейса.

Традиционно при анализе сигналов ВСР используют как статистические, так и спектральные [7—9] методы оценивания. Следует отметить, что несмотря на значительное число работ, посвященных интерпретации показателей ВСР, эти методы обработки ВСР не лишены существенных недостатков. Очевидно, например, что разные временные последовательности *RR*-интервалов кардиограмм могут дать одни и те же статистические показатели ВСР. Это не лучшим образом скажется на их содержательной интерпретации. Следует учитывать, что на кривой ВСР присутствуют волны различных периодов, связываемые с разными контурами регуляции частоты сердечных сокращений.

Спектральный анализ, основой которого традиционно является использование преобразования Фурье [8, 9], имеет ряд ограничений. Эти ограничения связаны с трудностями временной локализации тех или иных нестационарных процессов, наличие которых и позволяет формировать управляющие команды для человеко-машинного интерфейса.

Для выявления паттернов ВСР, на основе которых будут формироваться управляющие команды, авторами настоящей работы был разработан алгоритм, использующий непрерывное вейвлет-преобразование:

$$W(a,b) = 1/\sqrt{a}\sum_{t=0}^{t_{\max}} f(t)\,\psi\left(\frac{t-b}{a}\right),$$

где $f(t)$ — анализируемые данные, зависящие от дискретного времени t, выражаемого в отсчетах аналого-цифрового преобразователя (АЦП); ψ — вейвлет; a и b — параметры масштаба и времени вейвлет-преобразования соответственно; a, b — целые числа, $a > 0$; $t_{\max}$ — максимальное время регистрации сигнала (в отсчетах).

После расчета матрицы квадратов коэффициентов вейвлет-преобразования $W^2(a, b)$ получается набор локальных спектров. По их экстремумам и строятся, по аналогии с анализом электроэнцефалограмм [10, 11], цепочки локальных максимумов (ЦЛМ) и минимумов (ЦЛМин). Данные структуры матрицы $W^2(a, b)$ несут важную информацию об особенностях регуляции ВСР и, как следствие, о функциональном напряжении конкретного человека. При этом частотные диапазоны наиболее высокоамплитудных компонентов сигнала, которые трактуются как выражение активности тех или иных управляющих контуров организма, определяются динамически как разница значений локальных минимумов одного спектра.

Определим величину U, с помощью которой можно описать активность регуляторных контуров ВСР в динамически сформированных частотных диапазонах:

$$U = \frac{1}{N}\sum_{l=l_{\min}}^{l_{\max}}\sum_{k=k_{\min}}^{k_{\max}}\left(W^2(a_l, b_k)\right),$$

где l — индекс, нумерующий масштабы вейвлет-преобразования; k — индекс, характеризующий положение вейвлета на оси дискретного времени; $k_{\min}$ и $k_{\max}$ устанавливают временной отрезок для построения локальных спектров; $N = (l_{\max} - l_{\min})(k_{\max} - k_{\min})$.

Величина U определяет мощность сигнала в тех или иных частотных физиологически значимых диапазонах. Данные диапазоны связаны, в свою очередь, с контурами регуляции ВСР. Мощность сигнала в этих частотных диапазонах отражает в том числе и функциональное напряжение человека. Следовательно, один и тот же контур регуляции ВСР может иметь частотный диапазон больший, чем его общепринятые значения, захватывая частотные области,

которые относятся к активности иных центров регуляции. Возможно и обратное: контур регуляции может сужать свой частотный диапазон в ходе реакций ВСР на экзогенные и эндогенные стимулы.

Таким образом, используя полученные как временные, так и частотные особенности волновой структуры ВСР, удалось определить набор параметров кривой ВСР, который соответствует процессам, происходящим при изменении функционального напряжения человека.

Проанализируем данные ВСР, чтобы оценить степень функционального напряжения пользователя. После их обработки и превращения в команды они должны изменить характер функционирования установленной на компьютере операционной системы или какого-то иного программного обеспечения, работающего под ее управлением [6].

Для выявления частотно-временных особенностей ВСР, несущих информацию о функциональном напряжении, будем проводить анализ не только единичных *RR*-интервалов электрокардиограмм, но и их последовательностей: RR_1, RR_2, ..., RR_i, ... (или, в случае использования пульсовых волн и фотоплетизмограмм, временных отрезков пик-пик).

На рис. 1, *а* представлено изменение динамики ВСР при решении человеком в уме арифметической задачи: исходное "состояние 1" (до начала решения задачи) — слева от левого вертикального маркера; "состояние 2" (между двумя вертикальными маркерами) — состояние функционального напряжения человека в процессе решения задачи; "состояние 3" (справа от правого вертикального маркера) — возвращение в исходное состояние после окончания решения.

По данным, представленным на рис. 1, *а*, построена скаттерграмма ВСР (рис. 2), где "состояние 1" — круглые маркеры, "состояние 2" — черные квадратные маркеры, "состояние 3" — серые квадратные маркеры. Нетрудно заметить, что отсутствие анализа временных последовательностей ВСР может привести к тому, что *RR*-интервалы, соответствующие разным состояниям организма (выделенная овальная область на рис. 2), будут находиться весьма близко друг к другу. Это обстоятельство существенно затруднит оценку паттернов активности ВСР и, соответственно, снизит информативность с точки зрения формирования команд человеко-машинному интерфейсу.

В свете отмеченных особенностей более целесообразным, чем величина *U,* будет оценка спектральных компонентов ВСР на основе анализа структуры цепочек локальных максимумов и минимумов матрицы квадратов коэффициентов вейвлет-преобразования (рис. 1, *б*). Данные цепочки получены путем непрерывного вейвлет-преобразования в частотно-временном пространстве.

Пусть имеется набор локальных спектров, формирующих цепочку локальных максимумов длительностью Δb. Каждый локальный максимум цепочки ограничен со стороны высоких и низких частот локальными минимумами, что формирует в простран-

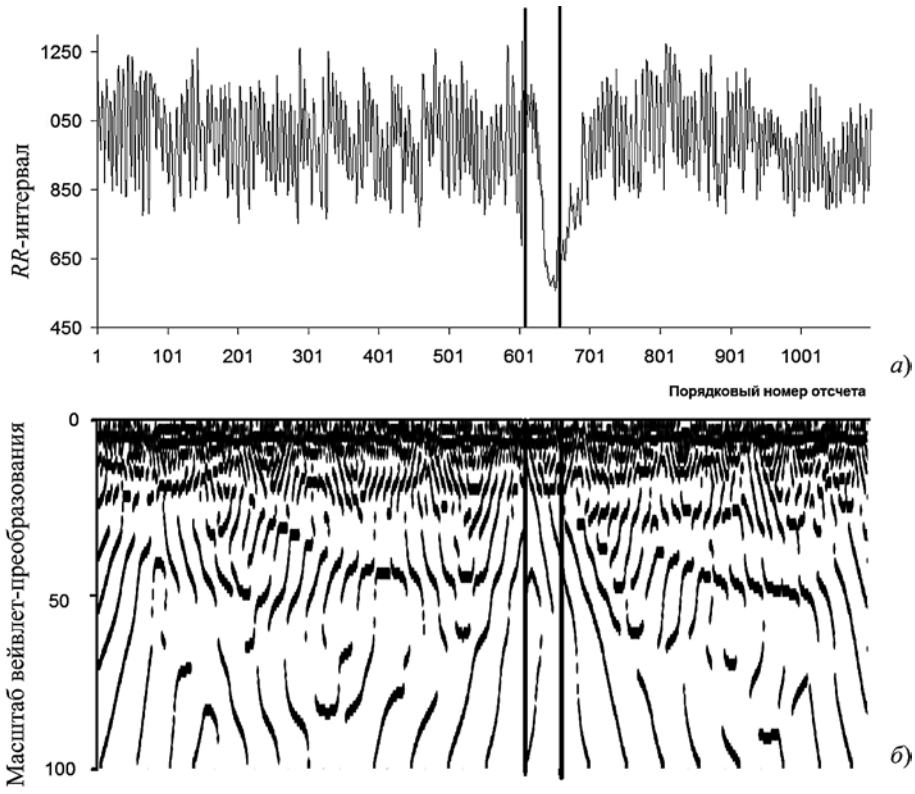


Рис. 1. Кривая вариабельности сердечного ритма (*а*) и цепочки локальных максимумов на плоскости W^2 (*a*, *b*) (*б*)

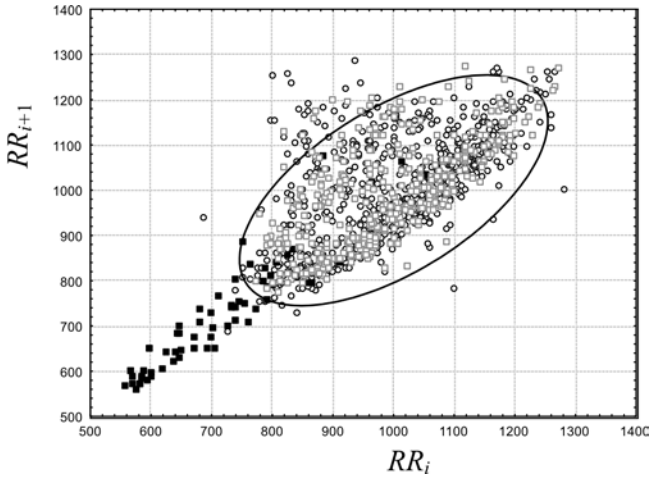


Рис. 2. Скаттерграмма ВСР, построенная по данным, представленным на рис. 1, *а*

стве (a, b) область $W_i^2(\Delta a,\Delta b)$. Рассмотрим область $W_i^2(\Delta a,\Delta b)$ как множество точек на поверхности (a, b). Все исследуемое частотно-временное пространство, при этом содержащее паттерн, который отражает функциональное напряжение и представляется несколькими ЦЛМ, можно определить как

$$W^2(\Delta a,\Delta b) = W_1^2(\Delta a_1,\Delta b_1) \cup w_2^2(\Delta a_2,\Delta b_2) \cup \ \ldots$$
$$\ldots \cup W_m^2(\Delta a_m,\Delta b_m),$$

где m — последний по времени на отрезке Δb элемент множества $W^2(\Delta a,\ \Delta b)$; $\Delta a = \Delta a_1 \cup \Delta a_2 \cup \ldots \cup \Delta a_m$ и $\Delta b = \Delta b_1 \cup \Delta b_2 \cup \ldots \cup \Delta b_m$; Δa — частотный диапазон области $W^2(\Delta a,\Delta b)$; Δb — временной диапазон этой же области.

Полученные ЦЛМ и ЦЛМин формируют области $W_i^2(\Delta a,\Delta b)$, при этом $W_1^2(\Delta a_1,\Delta b_1) \cap \ldots \cap W_m^2(\Delta a_m,\Delta b_m) = \varnothing$.

Таким образом, частотно-временное пространство, отражающее функциональное напряжение человека, по данным ВСР будет разбито на ряд подпространств (или областей). Каждое из этих подпространств несет информацию о контурах регуляции, участвующих в формировании кривой ВСР.

Формализуем представленные выше соображения. Пусть имеется вектор **V**, отражающий текущие значения параметров биомедицинских сигналов, регистрируемых от пользователя при его работе с интерфейсом. Применительно к описываемому интерфейсу это будут параметры *RR*-интервалов электрокардиограммы (или интервалов пик-пик на пульсограмме, фотоплетизмограмме) и тем или иным способом рассчитанные на их основе показатели функционального напряжения и активности ВНС. Получение команд k для работы данного интерфейса можно описать с помощью следующих логических выражений:

$$\begin{cases} \left(\mathbf{V}_i \in \left([x_1,x_2],[y_1,y_2],[z_1,z_2]\right)\right) \Rightarrow k_i, \\ \left(\mathbf{V}_{i+1} \in \left([x_3,x_4],[y_3,y_4],[[z_3,z_4]\right)\right) \Rightarrow k_{i+1} \\ \ldots\ldots\ldots\ldots \\ \left(\begin{matrix} \mathbf{V}_{i+d} \in \left(\left[x_{2d+1},x_{2(d+1)}\right],\left[y_{2d+1},y_{2(d+1)}\right]\right), \\ \left[z_{2d+1},z_{2(d+1)}\right] \end{matrix}\right) \Rightarrow k_{i+d}, \end{cases}$$

где $k_i,\ldots,\ k_{i+d}$ — управляющие команды, например, команды операционной системе на функционирование в определенном режиме, на изменение параметров работы программного обеспечения компьютера или графических интерфейсов программ; d — число команд, которые нужно классифицировать; x_i, y_i, z_i — координаты вектора $\mathbf{V}_i$ в пространстве, обеспечивающем его разделение с заданной вероятностью с другими векторами $\mathbf{V}_j$ при $j \neq i$, при этом координаты векторов $\mathbf{V}_i$ и $\mathbf{V}_j$ не совпадают.

В ходе обучения интерфейса передаче команд, отражающих состояния функционального напряжения и вегетативной нервной системы конкретного пользователя, устанавливается соответствие между вектором $\mathbf{V}_i$ и командами k_i для операционной системы и других программ. Иными словами, появление вектора $\mathbf{V}_i$ трактуется как нахождение пользователя в определенном состоянии функционального напряжения. Этот факт означает, что необходимо применить команду k_i для изменения параметров работы того программного обеспечения, которым управляет интерфейс "человек-компьютер". Например, это может быть изменение приоритета программы, с которой в данный момент работает пользователь, цветовой гаммы графического интерфейса программы и т.д. Аналогично, наличие вектора $\mathbf{V}_j$ свидетельствует о состоянии пользователя (не обязательно произвольном), при котором нужно применить другую команду k_j для управления этим же программным обеспечением. После того как обучение интерфейса выполнено, т. е. изменение состояния пользователя привело к изменению параметров функционирования программ в желаемом направлении, интерфейс начинает работать в режиме передачи команд от пользователя к программному обеспечению. При этом изменение функций данного программного обеспечения и входит в задачу интерфейса.

Определение феноменов ВСР для формирования команд интерфейса, зависящего от функционального напряжения человека

Определим параметры интерфейса, зависящего от функционального напряжения человека. Будем использовать такой вариант обработки данных ВСР, при котором анализируется вейвлетная плотность мощности [12—17]. Отметим, что это аналог спектральной плотности мощности для преобразования Фурье, которая рассчитывается в динамически из-

меняющихся частотных диапазонах исследуемого сигнала. При этом, согласно частотным подходам к анализу ВСР, будем рассматривать группу векторов, содержащих данные только о двух частотных диапазонах ВСР — *HF* и *LF*, отражающих различные контуры регуляции частоты сердечных сокращений. Очевидно, что для полноценной обработки кривой ВСР необходимо осуществить накопление определенного числа *RR*-интервалов кардиограммы, содержащих те или иные изменения волновой структуры ВСР. При этом лимитирующим фактором эпохи анализа будет число осцилляций са́мого низкочастотного компонента сигнала ВСР. Таким образом, при использовании вейвлета Морле потребуется не менее 120 кардиоциклов для того чтобы можно было составить представление о высокочастотных элементах в *VLF*-диапазоне — одном из самых медленных частотных диапазонов ВСР. Очевидно, что подобная задержка во времени существенно снижает ценность информации, передаваемой по данному типу интерфейса, делая его функционирование малопригодным для практического использования.

Применяя результат обучения человеко-машинного интерфейса, нетрудно определить особенности пространства (*HF*, *LF*), отражающие функциональное напряжение, состояние покоя или иные состояния человека, которые могут использоваться для управления компьютером. Например, для команды k_i это может быть набор векторов $\mathbf{V}_1\left(\Delta_1^i HF, \Delta_1^i LF\right)$, $\mathbf{V}_2\left(\Delta_2^i HF, \Delta_2^i LF\right)$, ..., $\mathbf{V}_m\left(\Delta_m^i HF, \Delta_m^i LF\right)$, где $\Delta_j^i HF$ — область динамически определенного *HF*-диапазона ВСР. Это именно тот диапазон, в котором находился вектор $\mathbf{V}_i$ при нахождении пользователя в состоянии, соответствующем команде k_i, при $j = 1, 2, ..., m$; $\Delta_j^i LF$ — область динамически определенного *LF*-диапазона ВСР при тех же условиях.

Введем логическую функцию $S_i(t_i)$, отражающую попадание интервала $RR_i(t_i)$, т. е. *i*-го *RR*-интервала в момент времени t_i, в "коридор" между нижними значениями $RR_i^n(t_i)$ и верхними значениями $RR_i^v(t_i)$ в тот же момент времени:

$$S_i(t_i) = \begin{cases} 1, RR_i(t_i) \in [RR_i^n(t_i), RR_i^v(t_i)] \\ 0, RR_i(t_i) \notin [RR_i^n(t_i), RR_i^v(t_i)] \end{cases}. \quad (1)$$

Присутствие или отсутствие сигнала в таком коридоре значений определяется логической функцией *Z*. Она учитывает превышение числа *RR*-интервалов, находящихся в указанном коридоре, а именно значения *P*, задаваемого исходя из целей пользователя и являющегося его индивидуальной особенностью. Логическую функцию *Z* можно описаь следующей формулой:

$$Z = \begin{cases} 0, \frac{1}{m}\sum_{i=1}^{m} S_i \leqslant P \\ 1, \frac{1}{m}\sum_{i=1}^{m} S_i > P \end{cases}, \quad (2)$$

где *m* — отрезок времени, на котором анализировалась функция S_i.

При этом определено, что паттерн ВСР сохраняется достаточно длительное время (до одной минуты), что позволяет рассматривать его в качестве индивидуально-воспроизводимой команды для использования в человеко-машинном интерфейсе. Следовательно, передача данных о состоянии пользователя в компьютер, обеспечивающая изменение работы программного обеспечения, наиболее информативна при изменении структуры вариабельности сердечного ритма. Это изменение отражает динамику адаптационных процессов пользователя (в том числе как к эмоциональной, так и к умственной нагрузке, сопровождающей работу человека на компьютере).

Схема архитектуры предлагаемого интерфейса с учетом того, что его реализация может быть как локальной, так и связанной с передачей данных в сеть, представлена на рис. 3.

Как видно на рис. 3, при работе человеко-машинного интерфейса данные о функциональном напряжении пользователя в виде паттернов частоты сердечных сокращений регистрируются как аналоговые сигналы и в виде электрокардиограммы, и в виде фотоплетизмограммы. Оцифрованный сигнал попадает в компьютер пользователя как по проводному, так и по беспроводному (Bluetooth) каналам. Полученные цифровые ряды обрабатываются либо с использованием статистических методов, либо с использованием спектральных оценок по методу Фурье, либо с применением вейвлет-преобразования. Учитывая достаточно высокую ресурсоемкость спектральных методов оценивания, для ускорения расчетов можно применять процессоры видеокарт (авторы использовали технологию CUDA [14]). На основе полученных результатов на следующем этапе проводится оценка функционального напряжения пользователя. В зависимости от конфигурации человеко-машинного интерфейса данные о функциональном напряжении используются в дальнейшем в двух направлениях. Первое направление — для изменения параметров работы программного обеспечения, не относящегося к человеко-машинному интерфейсу. К числу таких параметров относятся, например, параметры графических интерфейсов различных программ. К первому же направлению можно отнести и данные, которые используются для изменения приоритетов программ в операционных системах семейства Windows, цвета текста в Интернет-чате (chat, средство обмена сообщениями по компьютерной сети в режиме реального времени) и т. п. Второе направление — передача данных о функциональном напряжении пользователя в сеть с тем чтобы изменять параметры визуализации его текущего состояния на компьютерах иных пользователей данной сети.

Тестирование созданного человеко-машинного интерфейса показало, что в зависимости от частоты сердечных сокращений время генерации одной команды составляет от 2...5 кардиоциклов (за 1,5...4 с) (при оценке функционального напряжения со-

Рис. 3. Схема архитектуры человеко-машинного интерфейса, учитывающего функциональное напряжение пользователя

гласно формулам (1), (2)), до 60...120 кардиоциклов (за 50...80 с) в случае использования вейвлет-преобразования и анализа ЦЛМ и ЦЛМин.

Заметим, что информация, передаваемая по этому типу интерфейса, принципиально отличается от данных, используемых в текстовых и графических интерфейсах, доминирующих в настоящее время. Основная причина такого отличия в том, что команды, которые передаются по функционально-зависимому интерфейсу, несут в себе сведения о функциональном напряжении пользователя. Как следствие, они позволяют корректировать работу компьютера в направлении снижения этого напряжения.

Заключение

Предложено архитектурное решение, на основе которого разработан вариант функционально-зависимого человеко-машинного интерфейса, основанного на регистрации, анализе и практическом использовании биомедицинских сигналов. Такие сигналы характеризуют функциональное напряжение человека, включающего в себя и эмоциональный компонент его поведения. Результаты обработки как временных, так и частотных особенностей волновой структуры ВСР, обеспечили определение набора параметров, который соответствует процессам, происходящим при изменении функционального напряжения человека. Данные параметры могут использоваться для формирования команд в рамках разработанного человеко-машинного интерфейса. Показано, что сигнал ВСР после соответствующей обработки может быть использован для формирования команд управления программным обеспечением (операционной системой и программными приложениями), что существенно расширяет функциональные возможности человеко-машинного взаимодействия.

Список литературы

1. **Kinect** for Windows. URL: http://www.microsoft.com/en-us/kinectforwindows/
2. **Кубряк О. В., Гроховский С. С.** Практическая стабилометрия. Статические двигательно-когнитивные тесты с биологической обратной связью по опорной реакции. М.: Маска, 2012. 88 с.
3. **Eye** tracking products. URL: http://www.tobii.com/en/eye-tracking-research/global/products/
4. **Lotte F., Congedo M., L'ecuyer A. et al.** A review of classification algorithms for EEG-based brain−computer interfaces // Journ. Neural Eng. 2007. V. 4. P. R1—R13.
5. **Kaplan A. Ya., Lim J. J., Jin K. S. et al.** Unconscious operant conditioning in the paradigm of brain-computer interface based on color perception // Intern. Journ. Neuroscience. 2005. Vol. 115. P. 781—802.
6. **Туровский Я. А., Максимов А. В., Кургалин С. Д.** Способ оптимизации управления компьютером. Патент RU 2485572 от 10 мая 2012 г.
7. **Баевский Р. М., Баринова А. П., Барсукова Ж. В.** Возрастные особенности сердечного ритма у лиц с разной степенью адаптации к условиям окружающей среды // Физиология человека. 1985. Т. 11. № 2. С. 208—212.
8. **Heart** rate variability. Standarts of measurement, physiological interpretation and clinical use // Circulation. 1996. Vol. 93. P. 1043—1065.
9. **Баевский Р. М., Иванов Г. Г., Чирейкин Л. В. и др.** Анализ вариабельности сердечного ритма при использовании различных элекрокардиографических систем (методические рекомендации) // Вестник аритмологии. 2001. № 24. С. 65—87.
10. **Туровский Я. А., Кургалин С. Д., Максимов А. В., Семенов А. Г.** Анализ электроэнцефалограмм на основе исследования изменяющейся во времени структуры локальных

максимумов матрицы вейвлет-коэффициентов // Вестник Воронежского гос. ун-та. Сер. Системный анализ и информационные технологии. 2012. № 2. С. 69—73.

11. **Туровский Я. А., Кургалин С. Д., Вахтин А. А**. Обработка сигнала электроэнцефалограммы на основе анализа частотных зависимостей и вейвлет-преобразования // Биомедицинская радиоэлектроника. 2012. № 12. С. 39—45.

12. **Кургалин С. Д., Туровский Я. А., Максимов А. В., Насер Н.** Вейвлет-анализ энцефалограмм // Информационные технологии в проектировании и производстве. 2010. № 1. С. 89—95.

13. **Туровский Я. А., Кургалин С. Д., Семенов А. Г.** Динамика цепочек локальных максимумов спектров электроэнцефалограмм человека // Биофизика. 2014. Т. 59. Вып. 1. С. 185—190.

14. **Туровский Я. А., Кургалин С. Д., Максимов А. В.** Выбор анализирующих вейвлетов для системы с параллельной обработкой биомедицинских данных // Вестник Воронежского гос. ун-та. Сер. Системный анализ и информационные технологии. 2011. № 2. С. 74—79.

15. **Туровский Я. А., Кургалин С. Д., Максимов А. В.** Моделирование процесса выделения частотных локальных минимумов в сигналах электроэнцефалограмм // Вестник Тамбовского гос. технического ун-та. 2012. Т. 18. № 4. С. 827—833.

16. **Туровский Я. А., Кургалин С. Д., Вахтин А. А., Максимов А. В.** Фактор времени при реализации непрерывного вейвлет-преобразования для анализа сигналов ЭЭГ // Информационные технологии в проектировании и производстве. 2012. № 2. С. 61—66.

17. **Туровский Я. А., Кургалин С. Д., Семенов А. Г.** Моделирование выделения и анализа цепочек локальных максимумов вейвлет-спектров на примере сигналов с известными свойствами // Системы управления и информационные технологии. 2013. Т. 52, № 2. С. 24—29.

Y. A. Turovsky, Associate Professor, Head of Laboratory, e-mail: yaroslav_turovsk@mail.ru,
S. D. Kurgalin, Head of the Department, **A. A. Vahtin**, Associate Professor,
V. A. Belobrodsky, Postgraduate Student, Voronezh State University

Human-Machine Interface with Evaluation of Human Functional Pressure

The paper offers an architectural solution, which was used to design a version of functionally dependent human-machine interface, based on the detection, analysis and practical application of biomedical signals. Such signals characterize the functional pressure experienced by a human, which also includes the emotional component of human behavior. The results of processing both time and frequency features of the wave structure of heart rate variability provide a certain set of parameters, which corresponds to processes occurring when functional pressure of a human changes. These parameters can be used for forming commands in the frames of a developed "human-machine" interface. It is shown that the HRV signal, after an appropriate processing, can be used to form control commands for software (operating system and software applications). Such commands may be commands to change color settings of graphic interface, priorities of the program operation in the operating system, changes in multimedia applications operation. The designed interface greatly expands the functionality of human-machine interaction.

Keywords: *human-computer interface, biomedical signal, digital signal processing, managing, wavelet-transformation, stress, heart rate variability*

References

1. **Kinect** for Windows, available at: http://www.microsoft.com/en-us/kinectforwindows/

2. **Kubrjak O. V., Grohovskij S. S.** *Prakticheskaja stabilometrija. Staticheskie dvigatel'no-kognitivnye testy s biologicheskoj obratnoj svjaz'ju po opornoj reakcii* (Practical stabilometrics. Static motor-cognitive tests with biofeedback on supporting reaction). Moscow: Maska, 2012. 88 p. (in Russian).

3. **Eye** tracking products. available at: http://www.tobii.com/en/eye-tracking-research/global/products/

4. **Lotte F., Congedo M., L'ecuyer A., Lamarche F., Arnaldi B.** A review of classification algorithms for EEG-based brain−computer interfaces. *Journ. Neural Eng*, 2007, vol. 4, pp. R1−R13.

5. **Kaplan A. Ya., Lim J. J., Jin K. S., Park B. W., Byeon J. G., Tarasova S. U.** Unconscious operant conditioning in the paradigm of brain-computer interface based on color perception *Intern. Journ. Neuroscience*, 2005, vol. 115, pp. 781—802.

6. **Turovskij Ja. A., Maksimov A. V., Kurgalin S. D**. *Patent RU 2485572* 2012 (in Russian).

7. **Baevskij R. M., Barinova A. P., Barsukova Zh. V.** *Fiziologija cheloveka,* 1985, vol. 11, no. 2, pp. 208—212 (in Russian).

8. **Heart** rate variability. Standarts of measurement, physiological interpretation and clinical use. *Circulation*, 1996, vol. 93, pp. 1043—1065.

9. **Baevskij R. M., Ivanov G. G., Chireikin L. V., Gavrilushkin A. P., Dovgalevsky P. Ja., Kukushkin Iu. A., Mironova T. F., Prilutsky D. A., Semenov Iu. N., Fedorov V. F., Fleishman A. N., Medvedev M. M.** *Vestnik aritmologii,* 2001, no. 24, pp. 65—87 (in Russian).

10. **Turovskij Ja. A., Kurgalin S. D., Maksimov A. V., Semjonov A. G**. *Vestnik Voronezhskogo gos. un-ta. Ser. Sistemnyj analiz i informacionnye tehnologii,* 2012, no. 2, pp. 69—73 (in Russian).

11. **Turovskij Ja. A., Kurgalin S. D., Vahtin A. A.** *Biomedicinskaja radiojelektronika,* 2012. no. 12, pp. 39—45 (in Russian).

12. **Kurgalin S. D., Turovskij Ja. A., Maksimov A. V., Naser N.** *Informacionnye tehnologii v proektirovanii i proizvodstve,* 2010, no. 1, pp. 89—95 (in Russian).

13. **Turovskij Ja. A., Kurgalin S. D., Semjonov A. G.** *Biofizika,* 2014, vol. 59, no. 1, pp. 185—190 (in Russian).

14. **Turovskij Ja. A., Kurgalin S. D., Maksimov A. V.** *Vestnik Voronezhskogo gos. un-ta. Ser. Sistemnyj analiz i informacionnye tehnologii,* 2011, no. 2, pp. 74—79 (in Russian).

15. **Turovskij Ja. A., Kurgalin S. D., Maksimov A. V.** *Vestnik Tambovskogo gos. tehnicheskogo un-ta,* 2012, vol. 18, no. 4, pp. 827—833 (in Russian).

16. **Turovskij Ja. A., Kurgalin S. D., Vahtin A. A., Maksimov A. V.** *Informacionnye tehnologii v proektirovanii i proizvodstve,* 2012, no. 2, pp. 61—66 (in Russian).

17. **Turovskij Ja. A., Kurgalin S. D., Semjonov A. G.** *Sistemy upravlenija i informacionnye tehnologii,* 2013, vol. 52, no. 2, pp. 24—29 (in Russian).

УДК 004.023

П. В. Матренин, аспирант, e-mail: pavel.matrenin@gmail.com, Новосибирский государственный технический университет

Описание и реализация алгоритмов роевого интеллекта с использованием системного подхода

Приведены результаты анализа алгоритмов роевого интеллекта как особого класса алгоритмов оптимизации. Отмечены принципиальные отличия алгоритмов роевого интеллекта от других стохастических алгоритмов оптимизации. Представлены UML-диаграммы, иллюстрирующие предложенный автором единый подход к описанию алгоритмов роевого интеллекта. Продемонстрированы возможности применения такого подхода для описания алгоритма поиска косяком рыб и алгоритма роя пчел.

Ключевые слова: *алгоритм роя пчел, поиск косяком рыб, глобальная оптимизация, стохастические методы, роевой интеллект, системный подход*

Введение

Для решения задач глобальной оптимизации часто применяют так называемые популяционные алгоритмы. К их числу относят эволюционные алгоритмы; алгоритмы, использующие концепцию роевого интеллекта; алгоритмы, основанные на иных механизмах живой и неживой природы [1]. В работе [2] было показано, что стохастическая природа этих алгоритмов, их многообразие и использование аналогий с природными механизмами приводят к ряду трудностей, связанных с классификацией, терминологической поддержкой и описанием алгоритмов. Для разрешения этих трудностей было предложено использовать системное описание на примере алгоритмов роевого интеллекта. Настоящая статья развивает тему работы [2] с теоретических позиций, отмечая особенности алгоритмов роевого интеллекта и их отличия от других популяционных алгоритмов. С практической точки зрения эти особенности и отличия позволяют сформулировать рекомендации по разработке программ, использующих алгоритмы роевого интеллекта. Приводятся полученные на основе предложенного подхода описания алгоритма роя пчел и алгоритма поиска косяком рыб.

Концептуальные положения

Концептуальные положения модели роевого интеллекта основаны на описании децентрализованных систем, состоящих из единообразных элементов (агентов), косвенно взаимодействующих друг с другом и с окружающей средой для достижения предопределенной цели [3]. Именно это определение лежит в основе алгоритмов роевого интеллекта. Для дальнейшего анализа алгоритмов роевого интеллекта необходимо перечислить основные обозначения, введенные в работе [2]:

$f(\mathbf{X})$ — скалярная целевая функция (фитнесс-функция), для которой требуется найти экстремальное значение;

$\mathbf{X}$ — вектор варьируемых параметров;

D — область допустимых значений $\mathbf{X}$, $D \subset R^{|\mathbf{X}|}$; $R^{|\mathbf{X}|}$ — пространство поиска решений;

$G(\mathbf{X})$ — функция, задающая ограничения на $\mathbf{X}$;

$|S|$ — число агентов роя;

$\mathbf{S} = \{s_1, s_2, ..., s_{|S|}\}$ — множество всех агентов роя;

$\mathbf{X}_{ij}$ — вектор варьируемых параметров i-го агента на j-й итерации алгоритма;

$\mathbf{X}^{opt}$ — наилучшее значение вектора варьируемых параметров;

f^{opt} — наилучшее значение целевой функции;

$\varphi(\mathbf{X}) = f(\mathbf{X})$ — значение фитнесс-функции в положении $\mathbf{X}$.

В рамках упомянутого анализа алгоритмов роевого интеллекта рассматривается задача максимизации, которая легко сводится к задаче минимизации:

$$f^{opt} = f(\mathbf{X}^{opt}) = \max_{\mathbf{X} \in D} f(\mathbf{X}).$$

Особенности алгоритмов роевого интеллекта

Во многих исследованиях не выделяют алгоритмы роевого интеллекта в особую группу. Эти алгоритмы смешиваются с эволюционными алгоритмами, такими как генетический алгоритм [1, 4—6] или с алгоритмом имитации отжига [4, 6]. В работе [1] эволюционные алгоритмы и алгоритмы роевого интеллекта объединяют под общим названием "популяционные алгоритмы". В работе [5] используется термин "*Swarm-based optimization algorithms* (SOAs)". Таким

образом, авторы относят и генетический алгоритм к алгоритмам роевого интеллекта, хотя концепции эволюционного отбора и роевого интеллекта имеют совершенно разные основы, как на уровне природных механизмов, так и на уровне математических моделей.

Действительно, алгоритмы роевого интеллекта и эволюционные алгоритмы относятся к алгоритмам популяционным. В некоторых работах утверждается, что алгоритмы роевого интеллекта, эволюционные алгоритмы и алгоритм имитации отжига имеют одну основу, а именно конечные цепи Маркова [4, 7]. Однако алгоритмы роевого интеллекта, в частности, наиболее известные — алгоритм роя части и муравьиный алгоритм — появились не в результате изучения цепей Маркова или их применения для решения задач оптимизации. Они стали результатом моделирования поведения птиц в стае (алгоритм роя частиц [8]) и изучения принципов поведения муравьев в природе (алгоритм колонии муравьев или муравьиный алгоритм [6]). В работе [9] предложенная модификация алгоритма роя частиц обосновывается с использованием теоретической концепции роевого интеллекта, а не на основе математических моделей алгоритма. Уже после распространения алгоритмов роевого интеллекта к ним были применены различные математические модели, в том числе и цепи Маркова. Использование цепей Маркова позволяет доказать сходимость рассматриваемых алгоритмов к глобальному оптимуму только теоретически, при устремлении времени работы алгоритма к бесконечности. Однако такой подход не объясняет показанную в многочисленных экспериментах высокую эффективность алгоритмов роевого интеллекта для решения практических задач с ограничениями по времени.

Различные принципы работы алгоритмов роевого интеллекта и эволюционных алгоритмов требуют их разделения. Эксперт в разработке алгоритмов С. Скиена указывает на недостаток эволюционных алгоритмов, опираясь на их биологические основы: "Операции пересечения и мутации обычно не используют структуры данных, специфичных для данной задачи, вследствие чего большинство переходов дает низкокачественные решения... В этом случае аналогия с эволюцией (которой для осуществления важных изменений требуются миллионы лет) оказывается уместной" [10]. В работе [11] приводится обзор недостатков эволюционных алгоритмов с поиском причин в самой идее применения принципов эволюции в решении задач оптимизации: "Неспособность формирования сложных структур и есть основная проблема эволюционных вычислений". Автор статьи [11] В. Э. Карпов противопоставляет эволюционным вычислениям "гипотезу простоты", высказанную М. Л. Цетлиным, но только как идею: "Следствием ее является идея о том, что совместное функционирование простых "маленьких зверушек" в сложной среде способно обеспечить устойчивое существование всего коллектива, который можно рассматривать как некий "сверхорганизм" [11]. Карпов В. Э. опускает тот факт, что высказанная им идея совпадает с концепцией роевого интеллекта. Вместе с тем очевидно, что в его работе неявно дается разделение эволюционных алгоритмов (эволюционных вычислений) и роевого интеллекта ("гипотеза простоты").

Алгоритмы роевого интеллекта принципиально отличаются от эволюционных, поскольку не требуют создания на каждом шаге новых популяций путем отбора и скрещивания агентов предыдущей популяции, а используют коллективные децентрализованные перемещения агентов одной популяции, без процедур отбора, уничтожения старых и порождения новых агентов. Способ определения, относится ли тот или иной популяционный алгоритм к роевому интеллекту, предложен в работе [2]: в формулах, задающих поведение (перемещения, миграцию) агентов роя, должен присутствовать объект, обеспечивающий косвенный обмен информацией между агентами.

С позиции практического применения можно также отметить важное отличие концепции роевого интеллекта от концепции эволюционных вычислений. На основе роевого интеллекта можно не только решать задачи оптимизации, но и управлять группами роботов. Это направление называется роевой робототехникой и активно развивается в последние годы [12—15]. Очевидно, что на основании принципов эволюционного отбора нельзя децентрализованно управлять группами роботов. Для этого потребовалось бы реализовать централизованный отбор роботов и создавать новых роботов в процессе их функционирования.

Новый подход к описанию и реализации алгоритмов роевого интеллекта

На основании проведенного анализа автором предлагается единая схема обобщенного описания алгоритмов роевого интеллекта, предполагающая представление алгоритма в следующем виде [2]:

$$SI = \{\mathbf{S}, \mathbf{M}, A, \mathbf{P}, I, O\}, \quad (1)$$

где **S** — множество агентов роя; **M** — объект для обмена опытом между агентами **S**; A — правила работы роевого алгоритма; **P** — параметры, используемые в правилах A; I и O — порты (входы и выход) роевого алгоритма, посредством которых он взаимодействует с окружающей средой и управляющей системой.

Любой алгоритм роевого интеллекта включает в себя следующие этапы, определяемые правилами A:

1) инициализация начальных состояний агентов **S**;

2) вычисление фитнесс-функции для каждого агента;

3) миграция (перемещение) агентов **S** с учетом правил A, вектора параметров **P** и объекта обмена опытом между агентами **M**.

Данная схема может быть применена не только для описания алгоритмов роевого интеллекта, но и для их программной реализации. Схему можно рассматривать как класс абстрактного роевого алго-

ритма, а каждый конкретный алгоритм (роя частиц, колонии муравьев, роя пчел, косяка рыб и т. п.) как вариант реализации абстрактного алгоритма. Такой подход соответствует широко распространенному шаблону проектирования "Абстрактный суперкласс", который гарантирует общее поведение концептуально связанных классов. Общая логика связанных классов реализуется в суперклассе, варианты поведения каждого отдельного алгоритма задаются в реализации методов с одинаковой сигнатурой, как это показано на UML-диаграмме классов на рис. 1.

Классы PSO (*Particle Swarm Optimization*), ACO (*Ant Colony Optimization*), ABCO (*Artificial Bee Colony Optimization*), FSS (*Fish School Search*) представлены как примеры классов, реализующих различные конкретные алгоритмы роевого интеллекта. Для упрощения диаграммы детали описания классов опущены. В соответствии с формулой (1) роевой алгоритм использует множество агентов **S** (agents), вектор параметров **P** (parameters), правила работы *A*, включающие инициализацию (initialization), вычисление целевой функции (getPostion, setFitness) и миграцию (movement) агентов роя. Для вычисления целевой функции используются средства обмена данными с моделью решаемой задачи (указанные в формуле (1) порты *I* и *O)*. Передача позиции агента на текущей итерации в блок вычисления целевой функции выполняется методом getPostion, а метод setFitness используется для сохранения вычисленного значения. Как правило, лучшее из найденных решений и соответствующее ему значение целевой функции хранятся отдельно (bestSolution и bestFitness соответственно). Другие указанные на рис. 1 методы позволяют выбрать наилучшее из найденных решений и работать с параметрами алгоритма. Реализация методов initialization и movement обладает спецификой для каждого алгоритма роевого интеллекта, прочие методы могут иметь одинаковую реализацию, заданную в классе Swarm.

На рис. 1 показано, что каждый агент роя является объектом класса Unit, содержащего текущую позицию агента (вектор координат в пространстве поиска) и значение целевой функции в этой позиции. В целом нет необходимости использовать отдельный класс для реализации агентов. Можно реализовать набор агентов с помощью структур или матриц для повышения скорости работы программы и экономии памяти. В данном случае использован именно класс для абстрагирования от деталей реализации. Заметим, что это относится и к самой структуре данных для хранения агентов, когда необязательно применять массив или вектор. Можно использовать связный список, а для алгоритма роя пчел имеет смысл применить такую структуру данных, как пирамида. В таком алгоритме на каждой итерации требуется определять часть агентов с наилучшим значением фитнесс-функции.

Схема взаимодействия обобщенного алгоритма роевого интеллекта с внешней средой представлена на рис. 2 с помощью UML-диаграммы взаимодействия.

Роевой алгоритм реализован объектом swarm1, модель решаемой задачи оптимизации — объектом foo.

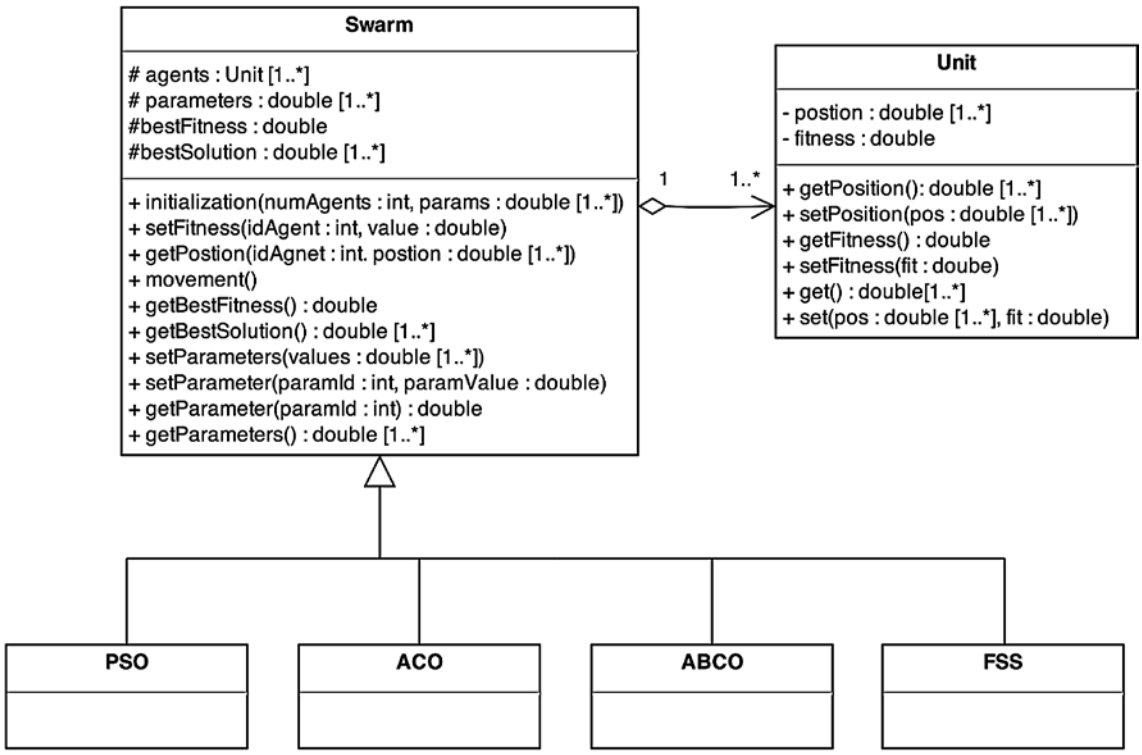


Рис. 1. Диаграмма классов, показывающая единый подход к алгоритмам роевого интеллекта

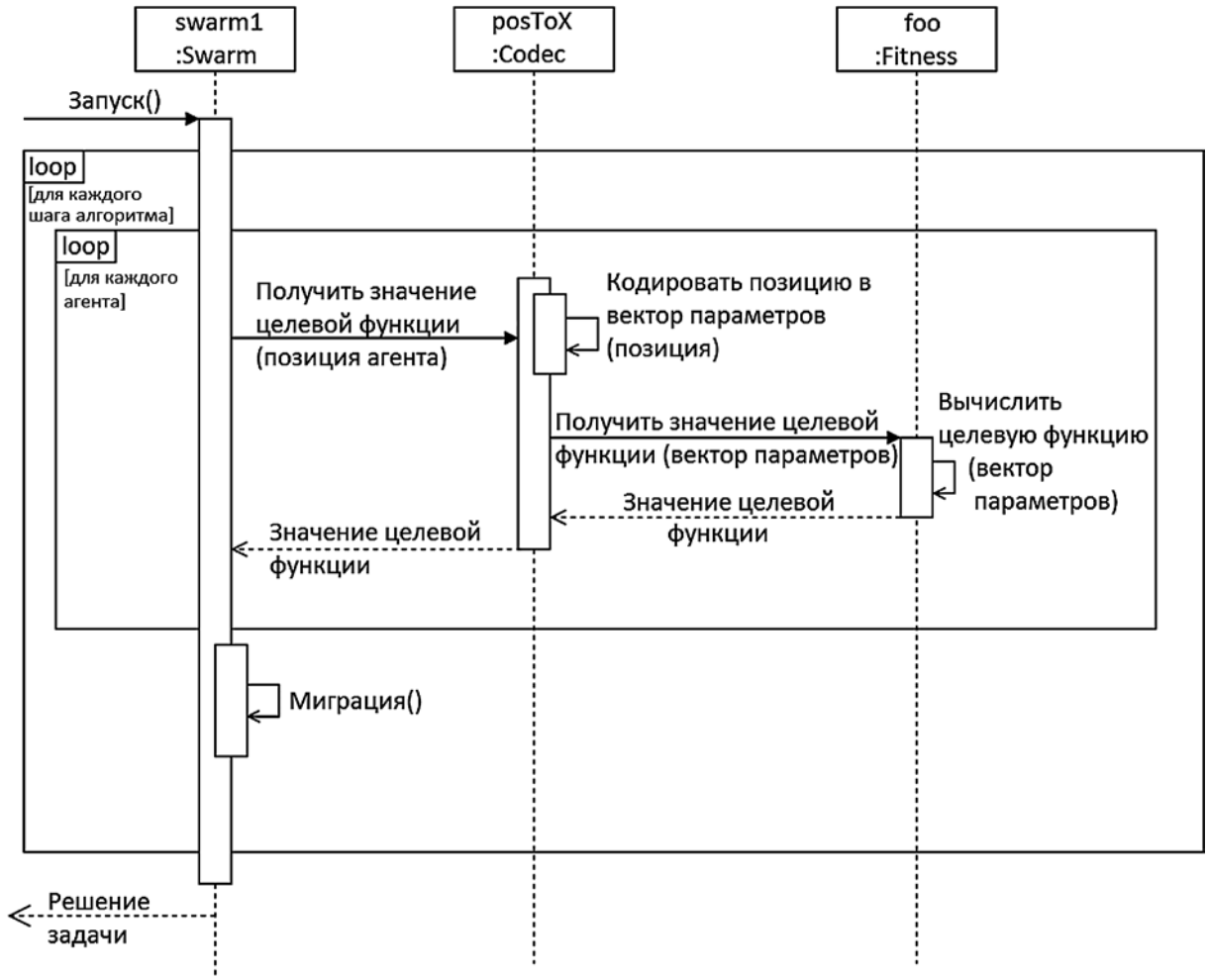


Рис. 2. UML-диаграмма взаимодействия работы обобщенного роевого алгоритма

Объект posToX выполняет преобразование позиций агента в вектор аргументов целевой функции. Такое преобразование бывает необходимо для комбинаторных задач, например, задач на графах. На каждом шаге алгоритма каждый агент определяет значение целевой функции в своей текущей позиции. Для этого позиция при необходимости преобразуется в вектор аргументов целевой функции, после вычисления которой полученное значение возвращается агенту. После определения значений целевой функции для всех агентов происходит из миграция (перемещение). После окончания работы алгоритма возвращается наилучшее их найденных решение задачи. В частных случаях возможны изменения представленной схемы. Например, миграция может выполняться каждым агентом сразу же после определения значения целевой функции в его позиции.

Такая реализация отличается большой гибкостью, поскольку она не зависит от решаемой задачи оптимизации и деталей реализации алгоритма роевого интеллекта. В результате повышается скорость реализации алгоритмов для решения новых задач и появляется возможность быстро добавлять в систему новые алгоритмы. Кроме того, представляется возможность использовать единые механизмы для настройки и адаптации различных алгоритмов роевого интеллекта. Ниже приведены примеры описаний алгоритма поиска косяком рыб и алгоритма роя пчел, выполненных согласно формуле (1).

Алгоритм поиска косяком рыб

Алгоритм поиска косяком рыб (*Fish School Search*, FSS) создан на основании моделирования движения косяка рыб и представлен Б. Фило и Л. Нето в 2008 г. [1]. В косяке рыбы двигаются в одном направлении с близкими скоростями, поддерживая согласованное движение и приблизительно одинаковое расстояние между соседями. Такой способ коллективного движения помогает эффективнее перемещаться на большие расстояния, находить пищу и защищаться от хищников. Как и для алгоритма роя частиц, агенты (рыбы) перемещаются в пространстве поиска решений, но правила перемещения и средство косвенного обмена информацией значительно отличаются.

Каждый агент выполняет несколько видов движений: во-первых, движения на основании только своего собственного опыта, во-вторых, на основании опыта всего косяка. Второй вид движения разделяется на две фазы, описанные ниже. В отличие от алгоритма роя частиц, косяк рыб "помнит" только результаты предыдущей итерации, а не наилучшие найденные результаты за весь процесс. Для применения алгоритма требуется, чтобы целевая функция была неотрицательна на всем пространстве поиска: $f(\mathbf{X}) \geqslant 0$, $\mathbf{X} \in D$.

Согласно формуле (1) для получения обобщенного описания алгоритм поиска косяком рыб необходимо представить в виде

$$FSS = \{\mathbf{S}, \mathbf{M}, A, \mathbf{P}, I, O\}.$$

Затем требуется поочередно описать каждый элемент *FSS* и действия на отдельных этапах работы алгоритма: инициализацию, вычисление фитнесс-функций и миграцию.

1. Множество агентов (рыб) $\mathbf{S} = \{s_1, s_2, \ldots, s_{|S|}\}$, $|S|$ — число агентов. На j-й итерации i-й агент характеризуется состоянием $s_{ij} = \{\mathbf{X}_{ij}, \mathbf{V}_{ij}, w_{ij}\}$, где $\mathbf{X}_{ij} = \left\{x_{ij}^1, x_{ij}^2, \ldots, x_{ij}^l\right\}$ — вектор варьируемых параметров (положение агента); $\mathbf{V}_{ij} = \left\{v_{ij}^1, v_{ij}^2, \ldots, v_{ij}^l\right\}$ — вектор скоростей агента; l — размерность пространства поиска решений; w_{ij} — вес i-го агента на j-й итерации.

2. Средством косвенного обмена $\mathbf{M}$ является вектор из двух элементов. Первый из них является скаляром и определяет взвешенную сумму индивидуальных перемещений рыб, а второй является вектором длиной l и представляет взвешенный центр тяжести всего косяка, $\mathbf{M} = \{m^S, \mathbf{C}_j\}$.

3. Реализация правил A определяет механизм функционирования косяка рыб и согласно используемой схеме описания должна включать в себя инициализацию, вычисление фитнесс-функций и миграцию. Существуют различные вариации алгоритма поиска косяком рыб. Далее представлена схема алгоритма в соответствии с описанием [1].

3.1. Инициализация начальных положений ($j = 1$):

$$\mathbf{X}_{i1} = \text{rand}\left(G(\mathbf{X})\right),\ i = 1,\ \ldots,\ |S|,$$

где rand($G(\mathbf{X})$) — вектор равномерно распределенных случайных величин, отвечающих ограничениям на область поиска;

$$w_{i1} = \frac{w_{\max}}{2},\ \ i = 1,\ \ldots,\ |S|, \qquad (2)$$

где $w_{\max}$ — один из параметров алгоритма.

3.2. Вычисление фитнесс-функций каждого агента на текущей j-й итерации:

$$\varphi\left(\mathbf{X}_{ij}\right) = f\left(\mathbf{X}_{ij}\right),\ i = 1,\ \ldots,\ |S|.$$

3.3. Миграция, т. е. перемещения агентов в рамках одной итерации, включает несколько стадий, в данное описание вводится промежуточная между итерациями j и $j + 1$ итерация с индексом $j + 0{,}5$. Этот индекс введен для упрощения формул и показывает, что соответствующие значения скорости i-й частицы $\mathbf{V}_{ij+0,5}$, ее положения $\mathbf{X}_{ij+0,5}$ и инерции $w_{ij+0,5}$ являются промежуточными в расчетах.

3.3.1. Индивидуальные перемещения агентов между итерациями j и $j + 1$ выполняются независимо для всех агентов и состоят из трех шагов. На первом шаге задается случайное значение скорости:

$$\mathbf{V}_{ij+0,5} = rnd\,\mathbf{V}_{\max},\ i = 1,\ \ldots,\ |S|, \qquad (3)$$

где rnd — случайное число, равномерно распределенное на интервале [0,1]; $\mathbf{V}_{\max}$ — вектор максимальных скоростей по каждому измерению области поиска $\mathbf{V}_{\max} = \left\{v_{\max}^1, v_{\max}^2, \ldots, v_{\max}^l\right\}$. Вектор может быть заменен скалярной величиной $v_{\max}$ в случае равенства всех его элементов.

На втором шаге выполняется перемещение с найденной скоростью в пределах области допустимых значений:

$$\mathbf{X}_{ij+0,5} = \begin{cases} \mathbf{X}_{ij} + \mathbf{V}_{ij+0,5},\ G\left(\mathbf{X}_{ij} + \mathbf{V}_{ij+0,5}\right) = 1 \\ \mathbf{X}_{ij},\ G\left(\mathbf{X}_{ij} + \mathbf{V}_{ij+0,5}\right) = 0 \end{cases},\ i = 1,\ \ldots,\ |S|,$$

здесь $G(\mathbf{X})$ используется как предикат, который показывает, принадлежит ли $\mathbf{X}$ области допустимых значений D.

Последний третий шаг возвращает агента на предыдущую позицию, если значение целевой функции в новой позиции оказалось хуже:

$$\mathbf{X}_{ij+0,5} = \begin{cases} \mathbf{X}_{ij+0,5},\ \varphi\left(\mathbf{X}_{ij+0,5}\right) \geqslant \varphi\left(\mathbf{X}_{ij}\right) \\ \mathbf{X}_{ij},\ \varphi\left(\mathbf{X}_{ij+0,5}\right) < \varphi\left(\mathbf{X}_{ij}\right) \end{cases},\ i = 1,\ \ldots,\ |S|.$$

3.3.2. Инстинктивно-коллективное плавание выполняется всеми рыбами в одном направлении и с одинаковой скоростью. На этом этапе используется объект косвенного взаимодействия m_j^S:

$$m_j^S = \frac{\Sigma_i\left(\mathbf{V}_{ij+0,5}\left(\varphi\left(\mathbf{X}_{ij+0,5}\right) - \varphi\left(\mathbf{X}_{ij}\right)\right)\right)}{\Sigma_i\left(\varphi\left(\left(\mathbf{X}_{ij+0,5}\right) - \varphi\left(\mathbf{X}_{ij}\right)\right)\right)}.$$

Каждая рыба после этого перемещается на данное значение:

$$\mathbf{X}_{ij+0,5} = \mathbf{X}_{ij+0,5} + m_j^S,\ i = 1,\ \ldots,\ |S|.$$

Если какой-либо элемент $\mathbf{X}_{ij+0,5}$ выходит за границу области допустимых значений, то он заменяется на значение соответствующей границы.

3.3.3. Последний этап перемещений называется коллективно-волевым плаванием. Предварительно необходимо вычислить веса агентов, вес агента i на шаге j вычисляется по следующей формуле:

$$w_{ij+0,5} = w_{ij} + \frac{\varphi\left(\mathbf{X}_{ij+0,5}\right) - \varphi\left(\mathbf{X}_{ij}\right)}{\max\left(\varphi\left(\mathbf{X}_{ij+0,5}\right), \varphi\left(\mathbf{X}_{ij}\right)\right)},\ i = 1,\ \ldots,\ |S|,$$

с учетом ограничения

$$1 \leqslant w_{ij+0,5} \leqslant w_{\max}.$$

Если в результате индивидуального и инстинктивно-коллективного плавания положение всего роя (косяка) в целом стало лучше, то область поиска сужается для более тщательного исследования текущей занятой области. В противном случае эта область расширяется для поиска новых решений и выхода из потенциального локального экстремума. Используется так называемый центр тяжести роя:

$$C_j = \frac{\sum_i w_{ij+0,5}\mathbf{X}_{ij+0,5}}{\sum_i w_{ij+0,5}},\ i = 1,\ \ldots,\ |S|. \qquad (4)$$

Перемещения при коллективно-волевом плавании выполняются по следующему правилу:

$$\mathbf{X}_{ij+1} = \begin{cases} \mathbf{X}_{ij+0,5} + vol\left(\mathbf{X}_{ij+0,5} - C_j\right), ws_{j+0,5} > ws_{j-0,5} \\ \mathbf{X}_{ij+0,5} - vol\left(\mathbf{X}_{ij+0,5} - C_j\right), ws_{j+0,5} \leqslant ws_{j-0,5} \end{cases}$$
$$i = 1,\ \ldots,\ |S|,$$

где $ws_{j+0,5}$ — сумма весов всех агентов на текущей стадии (знаменатель в формуле (4)); $ws_{j-0,5}$ является аналогичной суммой на предыдущей итерации, а *vol* определяет значение шага перемещений и вычисляется как

$$vol = rnd \cdot vol_{\max}, \qquad (5)$$

где $vol_{\max}$ — максимально возможный размер шага; *rnd* — случайная величина, равномерно распределенная на интервале [0, 1].

Чтобы получить окончательные позиции агентов на итерации $j+1$, нужно учесть границы области допустимых значений, так же как на этапе 2.2.

4. Коэффициенты (параметры) алгоритма, используемые в формулах (2), (3), (5), образуют вектор $\mathbf{P} = \{v_{\max}, w_{\max}, vol_{\max}\}$. Эти коэффициенты определяют поведение роя, выбор их значений является особой задачей, которая решается с помощью различных методов адаптации.

5. Идентификаторы *I* и *O* — описанные выше вход и выход роя, они не зависят от реализации алгоритма роевого интеллекта.

Алгоритм роя пчел

Алгоритм роя пчел (*Artificial Bee Colony Algorithm* или *Bees Algorithm*) разработан группой авторов и опубликован в 2005 г. [5, 16]. Метод основан на симуляции поведения пчел при поиске нектара. Рой пчел отправляет несколько разведчиков в случайных направлениях для поиска нектара. Вернувшись, разведчики сообщают о найденных на поле участках с цветами, содержащими нектар, и на них вылетают остальные пчелы. При этом чем больше на участке нектара, тем больше пчел к нему устремляется. Однако при этом пчелы могут случайным образом отклоняться от выбранного направления. После возвращения всех пчел в улей вновь происходит обмен информацией и отправка пчел-разведчиков и пчел-рабочих. Фактически разведчики действуют по алгоритму случайного поиска.

Для перехода к формальному описанию алгоритма необходимо представить поле с цветами как пространство поиска решения, а количество нектара как критерий задачи оптимизации, т. е. целевую функцию. На каждом шаге работы алгоритма среди всех агентов выбирается n^b лучших по значению целевой функции. Среди прочих выбирается еще n^g лучших, так называемых выбранных или перспективных. В некоторых вариантах алгоритма требуется, чтобы расстояния между каждой парой позиций в объединенном множестве лучших и выбранных позиций не превышали определенного значения. Иными словами, если есть две близкие позиции, то худшая из них по значению целевой функции отбрасывается, вместо нее берется позиция другого агента, подходящая под условия.

Определенные таким образом позиции (множество лучших $\mathbf{N}^b$ и множество выбранных $\mathbf{N}^g$) запоминаются и на следующем шаге в окрестность каждой лучшей позиции высылается c^b пчел, а каждой выбранной — c^g пчел. Пчела, посылаемая в окрестность участка, попадает в случайную точку внутри окрестности, например, в двумерном пространстве окрестность позиции с центром в точке (*x, y*) представляет область ([$x - rx$; $x + rx$], [$y - ry$; $y + ry$]), где *rx* и *ry* являются параметрами алгоритма. Возможно использование одного коэффициента *rx* по всем измерениям или вектора **RX**. Пчелы-разведчики высылаются в случайные позиции по всему пространству поиска на каждой итерации. Согласно формуле (1) для получения обобщенного описания алгоритм роя пчел необходимо представить в следующем виде

$$ABCO = \{\mathbf{S},\ \mathbf{M},\ A,\ \mathbf{P},\ I,\ O\}$$

и описать каждый элемент.

1. Множество агентов (пчел) $\mathbf{S} = \{s_1, s_2, \ldots, s_{|S|}\}$, $|S|$ — число агентов. На *j*-й итерации *i*-й агент характеризуется состоянием $s_{ij} = \{\mathbf{X}_{ij}\}$, где $\mathbf{X}_{ij} = \left\{x_{ij}^1, x_{ij}^2, \ldots, x_{ij}^l\right\}$ — вектор варьируемых параметров (положение агента); *l* — размерность пространства поиска решений.

2. Средством косвенного обмена **M** является список лучших и перспективных позиций, найденных на *j*-й итерации. Нужно отметить, что в отличие от эволюционных алгоритмов, отбираются не агенты, а только соответствующие позиции в пространстве поиска, $\mathbf{M} = \left\{\mathbf{N}_{ij}^b, \mathbf{N}_{kj}^g\right\}$, $i = 1, \ldots, n^b$, $k = 1, \ldots, n^g$.

3. Алгоритм, согласно (1) реализующий правила *A,* описывает механизмы функционирования роя пчел, в общем виде он может быть записан следующим образом.

3.1. Инициализация начальных положений ($j = 1$) выполняется только для пчел-разведчиков:

$$\mathbf{X}_i = \text{rand}(G(\mathbf{X}),\ \ i = 1,\ ...,\ n^s,$$

где n^s — число пчел-разведчиков.

3.2. Вычисление фитнесс-функций каждого агента на текущей j-й итерации:

$$\varphi(\mathbf{X}_{ij}) = f(\mathbf{X}_{ij}),\ \ i = 1,\ ...,\ |S|.$$

3.3. Миграция агентов. После формирования списков лучших и перспективных найденных на $(j-1)$-й итерации $\left(\mathbf{N}_{ij}^b, \mathbf{N}_{kj}^g\right)$, в окрестности позиций отправляются пчелы-рабочие. В окрестность каждой лучшей позиции отправляется c^b пчел, в окрестность каждой перспективной — c^g пчел. В некоторых вариантах алгоритма число отправляющихся в окрестность участка пчел зависит от его качества с точки зрения целевой функции, но в данном описании это не используется. Таким образом, позиции всех пчел-рабочих определяются следующим образом:

$$\mathbf{X}_{(i-1)c^b+kj} = \mathbf{N}_{ij-1}^b + \mathbf{Rnd} \cdot rad, \quad i = 1,\ ...,\ n^b, k = 1,\ ...,\ c^b; \tag{6}$$

$$\mathbf{X}_{n^b c^b+(i-1)c^b+kj} = N_{ij-1}^g + \mathbf{Rnd} \cdot rad, \quad i = 1,\ ...,\ n^g, k = 1,\ ...,\ c^g, \tag{7}$$

где **Rnd** — вектор, состоящий из l равномерно распределенных на диапазоне (−1, 1). случайных величин.

Пчелы-разведчики в этом случае отправляются в случайные позиции, координаты которых являются случайными величинами, равномерно распределенными на всем допустимом диапазоне значений:

$$\mathbf{X}_{n^b c^b+n^g c^g+ij} = \text{rand}\left(G\left(\mathbf{X}\right)\right),\ i = 1,\ ...,\ n^s.$$

4. Коэффициенты (параметры) алгоритма, используемые в данном описании и формулах (6) и (7), образуют вектор коэффициентов алгоритма $\mathbf{P} = \{n^s,\ n^b,\ n^g,\ c^b,\ c^g,\ rad,\ rx\}$ — коэффициенты алгоритма, который, согласно формуле (1) использует правила *A*. Коэффициент *rad* определяет рассеяние агентов при отправлении на лучшие и перспективные позиции, коэффициент *rx* задает минимально возможные расстояния между этими позициями. Значение выражения $n^s + n^b c^b + n^g c^g$ равно общему число агентов роя $|S|$. От выбора параметров алгоритма значительно зависит качество получаемых решений, поэтому для повышения эффективности алгоритма необходимо адаптировать параметры.

5. Идентификаторы *I* и *O* — описанные выше вход и выход роя, они не зависят от реализации алгоритма роевого интеллекта.

Заключение

Алгоритмы роевого интеллекта рекомендуется выделить в обособленную группу среди популяционных алгоритмов, поскольку они имеют общие базовые принципы работы, основанные на коллективных децентрализованных перемещениях агентов и косвенном обмене информации, в отличие от принципов отбора, по которым работают эволюционные алгоритмы.

Предложенное единообразное описание алгоритмов роевого интеллекта при их практической реализации обеспечивает стандартизацию, повышение гибкости и переносимости создаваемого программного обеспечения, повышение скорости разработки.

В рамках системного подхода в статье даны описания алгоритма поиска косяком рыб и алгоритма роя пчел, аналогичные описаниям алгоритма роя частиц и колонии муравьев, которые были представлены в работе [2].

Список литературы

1. **Карпенко А. П.** Популяционные алгоритмы глобальной оптимизации. Обзор новых и малоизвестных алгоритмов // Информационные технологии. Приложение. 2012. № 7. 32 с.

2. **Матренин П. В., Секаев В. Г.** Системное описание алгоритмов роевого интеллекта // Программная инженерия. 2013. № 12. С. 39—45.

3. **Beni G., Wang J.** Swarm Intelligence in Cellular Robotic Systems // Robots and Biological Systems: Towards a New Bionics? 1993. Vol. 102. P. 703—712.

4. **Зайцев И. Д.** Многоагентные системы в моделировании социально-экономических отношений: исследование поведения и верификация свойств с помощью цепей Маркова: дис. ... канд. техн. наук. Новосибирск, 2014. 142 с.

5. **Pham D. T., Ghanbarzadeh A., Koc E. et al.** The Bees Algorithm — A Novel Tool for Complex Optimisation Problems [Электронный ресурс]. Technical Note. Manufacturing Engineering Centre. Cardiff University. UK. 2005. URL: https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/Pham06%20-%20The%20Bee%20Algorithm.pdf.

6. **Dorigo M., Birattari M., Stutzle T.** Ant Colony Optimization. Artificial Ants as a Computational Intelligence Technique. IRIDIA. Technical Report Series. Technical Report No. TR/IRIDIA/2006-023. Brussels, Belgium. 2006. 14 p.

7. **Кочетов Ю. А.** Вероятностные методы локального поиска для задач дискретной оптимизации // Дискретная математика и ее приложения: Сборник лекций молодежных и научных школ по дискретной математике и ее приложениям. М.: Изд-во центра прикладных исследований при механико-математическом факультете МГУ, 2000. С. 87—117.

8. **Kennedy J., Eberhart R.** Particle Swarm Optimization. [Электронный ресурс] // Proc. of IEEE International Conference on Neural Network. Piscataway, NJ. 27 November — 01 December 1995. P. 1942—1948. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=488968.

9. **Pedersen M., Chippereld A.** Simplifying Particle Swarm Optimization // Applied Soft Computing. 2010. Vol. 10, Is. 2. P. 618—628.

10. **Скиена С.** Алгоритмы. Руководство по разработке. 2-е изд.: пер. с англ. СПб.: БХВ-Петербург, 2013. 720 с.

11 **Карпов В. Э.** Методологические проблемы эволюционных вычислений // Искусственный интеллект и принятие решений. 2012. № 4. С. 95—102.

12. **Карпов В. Э.** Управление в статических роях. Постановка задачи // Сб. научных трудов VII Междунар. науч.-практ. конф. "Интегрированные модели и мягкие вычисления в искусственном интеллекте". Коломна. 20—22 мая 2013. В трех томах. Т. 2. М.: Физматлит, 2013. С. 730—739.

13. **Иванов Д. Я.** Использование принципов роевого интеллекта для управления целенаправленным поведением массово-применяемых микроботов в экстремальных условиях // Известия высших учебных заведений. Машиностроение. 2011. № 9. С. 70—78.

14. **Beni G.** From Swarm Intelligence to Swarm Robotics // Swarm Robotics. Lecture Notes in Computer Science. 2005. Vol. 3342. P. 1—9.

15. **Sahin E.** Swarm robotics: From sources of inspiration to domains of application // Swarm Robotics. Lecture Notes in Computer Science. 2005. Vol. 3342. P. 10—20.

16. **Karaboga D.** An idea based on honey bee swarm for numerical optimization [Электронный ресурс]. Technical report TR06. Erciyes University, Engineering Faculty, Computer Engineering Department. 2005. URL: http://mf.erciyes.edu.tr/abc/pub/tr06_2005.pdf.

P. V. Matrenin, Postgraduate Student, e-mail: pavel.matrenin@gmail.com, Novosibirsk State Technical University

Description and Implementation of Swarm Intelligence Algorithms Using the System Approach

The article results analysis algorithms Swarm intelligence as a special class of optimization algorithms. It shows fundamental differences between Swarm Intelligence algorithms and other stochastic optimization algorithms. UML-diagrams illustrating the unified approach proposed by author to swarm intelligence algorithms are given.

Swarm Intelligence algorithms should be considered as a special group among population-based optimization algorithms, since they share the same characteristic idea, based on the collective movements of decentralized agents and the indirect information exchange. This distinguishes Swarm Intelligence of evolutionary algorithms that mimics the process of natural selection.

Proposed unified description of Swarm Intelligence algorithms in case of their implementation provides standardization, enhance flexibility and portability of software and increasing the speed of development.

The scheme is applied to describe the Artificial Bee Colony Optimization and the Fish School Search. Analogous descriptions of the Particle Swarm Optimization and Ant Colony Optimization have been given in this journal, N. 12, 2013.

Keywords: *artificial bee colony optimization, fish school search, global optimization, probabilistic methods, Swarm Intelligence, system approach*

References

1. **Karpenko A. P.** *Informatsionnye tekhnologii*, 2012, no. 9, pp. 1—32 (in Russian).

2. **Matrenin P. V., Sekaev V. G.** *Programmnaya ingeneria*, 2013, no. 12, pp. 39—45 (in Russian).

3. **Beni G., Wang J.** Swarm Intelligence in Cellular Robotic Systems. *Robots and Biological Systems: Towards a New Bionics?* 1993, vol. 102, pp. 703—712.

4. **Zaitsev I. D.** *Mnogoagentnye sistemy v modelirovanii sotsial'no-ekonomicheskikh otnoshenii: issledovanie povedeniya i verifikatsiya svoistv s pomoshch'yu tsepei Markova* (Multi-agent Systems for Modeling Socio-economic relations: the Research of the Behavior and Verification of Properties Using Markov Chains), Candidate's thesis, Novosibirsk, 2014. 142 p. (in Russian).

5. **Pham D. T., Ghanbarzadeh A., Koc E., Otri S., Rahim S., Zaidi M.** The Bees Algorithm — A Novel Tool for Complex Optimisation Problems, available at: https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/Pham06%20-%20The%20Bee%20Algorithm.pdf, 2005.

6. **Dorigo M., Birattari M., Stutzle T.** Ant Colony Optimization. Artificial Ants as a Computational Intelligence Technique, IRIDIA. Technical Report Series, Technical Report no. TR/IRIDIA/2006-023, Brussels. 2006.

7. **Kochetov Yu. A.** *Diskretnaya matematika i ee prilozheniya: Sbornik lektsii molodezhnykh i nauchnykh shkol po diskretnoi matematike i ee prilozheniyam,* Moscow, 2000, pp. 87—117 (in Russian).

8. **Kennedy J., Eberhart R.** Particle Swarm Optimization. *Proc. of IEEE International Conference on Neural Network*, Piscataway, NJ, 1995, pp. 1942—1948, available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=488968.

9. **Pedersen M., Chippereld A.** Simplifying Particle Swarm Optimization. *Applied Soft Computing*, 2010, vol. 10, issue 2, pp. 618—628.

10. **Skiena S.** *Algoritmy. Rukovodstvo po razrabotke* (The Algorithm Design Manual), Saint Petersburg: BKhV-Peterburg, 2013. 720 p. (in Russian).

11. **Karpov V. E.** *Iskusstvennyi intellekt i prinyatie reshenii,* 2012, no. 4, pp. 95—102 (in Russian).

12. **Karpov V. E.** *Sb. nauch. trudov VII Mezhdunar. nauch.-prakt. konf. "Integrirovannye modeli i myagkie vychisleniya v iskusstvennom intellekte"*, Kolomna, 2013. Moscow: Fizmatlit, 2013, pp. 730—739 (in Russian).

13. **Ivanov D. Ya.** *Izvestiya vysshikh uchebnykh zavedenii. Mashinostroenie*, 2011, no. 9, pp. 70—78 (in Russian).

14. **Beni G.** From Swarm Intelligence to Swarm Robotics, *Swarm Robotics. Lecture Notes in Computer Science*, 2005, vol. 3342, pp. 1—9.

15. **Sahin E.** Swarm robotics: From sources of inspiration to domains of application. *Swarm Robotics. Lecture Notes in Computer Science*, 2005, vol. 3342, pp. 10—20.

16. **Karaboga D.** An idea based on honey bee swarm for numerical optimization, available at: http://mf.erciyes.edu.tr/abc/pub/tr06_2005.pdf, 2005.

УДК 629.73.02; 629.73.05/.06; 535.643

И. О. Жаринов, д-р техн. наук, доц., зав. каф., e-mail: igor_rabota@pisem.net, Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики (НИУ ИТМО), руководитель учебно-научного центра, Санкт-Петербургское ОКБ "Электроавтоматика" им. П. А. Ефимова,
О. О. Жаринов, канд. техн. наук, доц., Санкт-Петербургский государственный университет аэрокосмического приборостроения (ГУАП)

Теоретическая оценка функции плотности вероятности распределения координат цвета в системах бортовой индикации

Рассмотрена задача определения функции плотности вероятности распределения координат цвета изображения. Показано, что вид функции плотности вероятности зависит от программного кода компонентов красного, зеленого и синего цветов, используемых в цветовой палитре, и в частных случаях соответствует виду функции плотности вероятности равномерного распределения — трапецеидальному виду, треугольному виду или кусочно-параболическому виду. Приведены аналитические выражения функции плотности вероятности распределения для всех видов математических зависимостей, учитывающих принятую модель технологического разброса и коэффициенты профиля жидкокристаллической панели.

Ключевые слова: *координаты цвета, системы индикации, функция плотности вероятности, сумма равномерно распределенных случайных величин, распределение Ирвина-Холла*

Введение

Практический опыт [1—3] разработки и эксплуатации многофункциональных цветных индикаторов (МФЦИ), выполненных на базе жидкокристаллической (ЖК) панели иностранного производства, показал, что один и тот же программный код *RGB* (*R* — Red, *G* — Green, *B* — Blue), определяющий цвет индицируемого изображения, имеет различные *XYZ*-координаты цвета и (*x*, *y*)-координаты цветности на разных образцах МФЦИ.

Отличия в значениях *XYZ*-координат цвета и в значениях (*x*, *y*)-координат цветности обусловлены технологическим разбросом параметров изготовления ЖК панелей. Эксперименты с применением колориметра показывают, что цветовая разница (*x*, *y*)-координат для изображения, индицируемого на разных образцах МФЦИ с одним кодом *RGB*, достигает 0,01...0,03 ед., что чувствительно для наблюдателя и для контрольно-измерительного оборудования.

Таким образом, группа индикаторов МФЦИ, изготовленных по одной программной и конструкторской документации с применением серийно изготавливаемых ЖК панелей и установленных в ряд на приборной панели гражданского самолета (до 6 шт.), различается по значениям координат цвета (цветности) индицируемого изображения, что недопустимо [4].

Отечественные разработчики в рамках программы импортозамещения в авиационной промышленности в настоящее время предпринимают попытки [5, 6] воспроизводства на территории Российской Федерации технологии изготовления ЖК панелей. Основные направления развития исследований связаны с разработкой конструктивов ЖК панелей, обеспечивающих стойкость и устойчивость образцов к внешним воздействующим факторам, и с разработкой технологии создания экранов на базе жидких кристаллов или светодиодов.

В связи с этим актуальной является задача исследования законов распределения координат цвета (цветности) синтезируемого изображения с формированием требований к качеству производства, обеспечивающему приемлемый для авиационной отрасли уровень технологического разброса визуальных параметров серийно изготавливаемых экранов для авионики.

1. Цветовые пространства в авионике

Координаты цвета изображения, формируемого на экране ЖК панели бортового средства индика-

ции, связаны со значениями кодов *RGB* программного обеспечения МФЦИ преобразованием Грассмана [7]:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}, \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix}^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \quad (1)$$

где X, Y, Z — координаты цвета в системе XYZ цветового треугольника Максвелла; X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b — весовые коэффициенты цвета (профиль ЖК панели), определенные Международной комиссией по освещению (МКО); R, G, B — десятичные коды компонентов основных цветов (красный, зеленый и синий).

Коэффициенты профиля ЖК панели определяют треугольник цветового охвата на XY-плоскости [8]. Треугольник цветового охвата включает геометрическое место точек со всеми возможными (x, y)-координатами цветности воспроизводимого на данной ЖК панели изображения. Значения (x,y)-координат цветности определяются через XYZ-координаты цвета [9]:

$$x = \frac{X}{X+Y+Z}, \; y = \frac{Y}{X+Y+Z}. \quad (2)$$

2. Модель технологического разброса параметров ЖК панели

Технологический разброс параметров изготовления ЖК панели учитывается в модели:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r + \Delta\xi_{X_r} & X_g + \Delta\xi_{X_g} & X_b + \Delta\xi_{X_b} \\ Y_r + \Delta\xi_{Y_r} & Y_g + \Delta\xi_{Y_g} & Y_b + \Delta\xi_{Y_b} \\ Z_r + \Delta\xi_{Z_r} & Z_g + \Delta\xi_{Z_g} & Z_b + \Delta\xi_{Z_b} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \Rightarrow$$

$$\Rightarrow \begin{cases} X = R\left(X_r + \Delta\xi_{X_r}\right) + G\left(X_g + \Delta\xi_{X_g}\right) + B\left(X_b + \Delta\xi_{X_b}\right) \\ Y = R\left(Y_r + \Delta\xi_{Y_r}\right) + G\left(Y_g + \Delta\xi_{Y_g}\right) + B\left(Y_b + \Delta\xi_{Y_b}\right) \\ Z = R\left(Z_r + \Delta\xi_{Z_r}\right) + G\left(Z_g + \Delta\xi_{Z_g}\right) + B\left(Z_b + \Delta\xi_{Z_b}\right), \end{cases} \quad (3)$$

где $\xi_{X_r} \in [-X_r/2; +X_r/2]$, $\xi_{X_g} \in [-X_g/2; +X_g/2]$,

$\xi_{X_b} \in [-X_b/2; +X_b/2]$, $\xi_{Y_r} \in [-Y_r/2; +Y_r/2]$,

$\xi_{Y_g} \in [-Y_g/2; +Y_g/2]$, $\xi_{Y_b} \in [-Y_b/2; +Y_b/2]$,

$\xi_{Z_r} \in [-Z_r/2; +Z_r/2]$, $\xi_{Z_g} \in [-Z_g/2; +Z_g/2]$,

$\xi_{Z_b} \in [-Z_b/2; +Z_b/2]$ — равномерно распределенные случайные величины; Δ — параметр технологического разброса, обусловленного качеством производства. Модель учитывает равномерный разброс допусков на значения коэффициентов X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b профиля ЖК панели, возникающий в процессе ее изготовления с использованием полупроводниковых светосинтезирующих элементов.

3. Теоретическая оценка функции плотности вероятности распределения координат цвета

Как следует из анализа уравнений (1), (3), координаты цвета могут определяться в трех комбинациях состояний компонентов кодов *RGB*:

1. Воспроизводимый на экране ЖК панели цвет образован одним компонентом кода *RGB* при нулевых значениях двух других компонентов кода, т. е. $R \neq 0$, $G = 0$, $B = 0$, $R = 0$, $G \neq 0$, $B = 0$ или $R = 0$, $G = 0$, $B \neq 0$.

В этом случае распределение координат цвета подчиняется равномерному закону, границы которого определяются абсолютным значением ненулевого компонента кода *RGB*, соответствующим коэффициентом профиля ЖК панели и параметром технологического разброса Δ.

Выражения для оценки функций плотности вероятности $f_X(t), f_Y(t), f_Z(t)$ распределения соответственно координат X, Y, Z цвета приведены в табл. 1. При выходе переменной t за пределы указанных в табл. 1 интервалов, функция плотности вероятности $f(t)$ равна нулю.

2. Цвет образован двумя ненулевыми компонентами кода *RGB* при нулевом значении третьего компонента кода, т. е. $R \neq 0$, $G \neq 0$, $B = 0$, $R = 0$, $G \neq 0$, $B \neq 0$ или $R \neq 0$, $G = 0$, $B \neq 0$. В этом случае функция плотности вероятности $f(t)$ распределения координаты цвета представляет собой математическую функцию трапецеидального вида. В статистическом смысле функция $f(t)$ соответствует функции плотности вероятности для суммы двух равномерно распределенных случайных величин [10, 11], каждая из которых определена на своем интервале $[a_i; b_i]$, $b_i > a_i > 0$, $i = 1, 2$.

Функция $f(t)$ вычисляется методом математической свертки функций плотности вероятности. Например, для X-координаты $f_X(t) = f_{\xi_{X_r}}(t-s) \otimes f_{\xi_{Xg}}(s)$ при $R \neq 0, G \neq 0, B = 0$:

$$\begin{aligned} f_X(t) &= f_{\xi_{X_r}}(t-s) \otimes f_{\xi_{Xg}}(s) = \\ &= \int_{-\infty}^{+\infty} f_{\xi_{X_r}}(t-s) f_{\xi_{Xg}}(s)\,ds = \underbrace{\int_{-\infty}^{a_2} f_{\xi_{X_r}}(t-s) f_{\xi_{Xg}}(s)\,ds}_{0} + \\ &+ \int_{a_2}^{b_2} f_{\xi_{X_r}}(t-s) f_{\xi_{Xg}}(s)\,ds + \underbrace{\int_{b_2}^{+\infty} f_{\xi_{X_r}}(t-s) f_{\xi_{Xg}}(s)\,ds}_{0} = \\ &= \frac{1}{b_2 - a_2} \int_{a_2}^{b_2} f_{X_r}(t-s)\,ds = [v = t-s, dv = -ds] = \\ &= \frac{1}{b_2 - a_2} \int_{v-b_2}^{v-a_2} f_{\xi_{X_r}}(v)\,dv = \\ &= \begin{cases} 0, t < a_1 + a_2, b_1 + b_2 \leqslant t \\ \dfrac{t - a_1 - a_2}{(b_2 - a_2)(b_1 - a_1)}, a_1 + a_2 \leqslant t < b_1 + a_2 \\ \dfrac{1}{b_2 - a_2}, b_1 + a_2 \leqslant t < a_1 + b_2 \\ \dfrac{b_1 + b_2 - t}{(b_2 - a_2)(b_1 - a_1)}, a_1 + b_2 \leqslant t < b_1 + b_2. \end{cases} \end{aligned} \quad (4)$$

Таблица 1

Выражения для оценки функции плотности вероятности распределения координат цвета *X, Y, Z*

Компоненты кода *RGB*	Функция плотности вероятности координаты цвета		
	$f_X(t)$	$f_Y(t)$	$f_Z(t)$
$R \neq 0$, $G = 0$, $B = 0$	$\frac{1}{X_r R\Delta}$, $t \in [RX_r - RX_r\Delta / 2;\ \ RX_r + RX_r\Delta / 2]$	$\frac{1}{Y_r R\Delta}$, $t \in [RY_r - RY_r\Delta / 2;\ \ RY_r + RY_r\Delta / 2]$	$\frac{1}{Z_r R\Delta}$, $t \in [RZ_r - RZ_r\Delta / 2;\ \ RZ_r + RZ_r\Delta / 2]$
$R = 0$, $G \neq 0$, $B = 0$	$\frac{1}{X_g G\Delta}$, $t \in [GX_g - GX_g\Delta / 2;\ \ GX_g + GX_g\Delta / 2]$	$\frac{1}{Y_g G\Delta}$, $t \in [GY_g - GY_g\Delta / 2;\ \ GY_g + GY_g\Delta / 2]$	$\frac{1}{Z_g G\Delta}$, $t \in [GZ_g - GZ_g\Delta / 2;\ \ GZ_g + GZ_g\Delta / 2]$
$R = 0$, $G = 0$, $B \neq 0$	$\frac{1}{X_b B\Delta}$, $t \in [BX_b - BX_b\Delta / 2;\ \ BX_b + BX_b\Delta / 2]$	$\frac{1}{Y_b B\Delta}$, $t \in [BY_b - BY_b\Delta / 2;\ \ BY_b + BY_b\Delta / 2]$	$\frac{1}{Z_b B\Delta}$, $t \in [BZ_b - BZ_b\Delta / 2;\ \ BZ_b + BZ_b\Delta / 2]$

Система (4) справедлива для $b_2 - a_2 > b_1 - a_1 > 0$, в случае невыполнения неравенства потребуется переопределение индексов коэффициентов a_i, b_i. Можно показать, что в зависимости от соотношения длин интервалов $[a_1; b_1]$ и $[a_2; b_2]$ выражения функции плотности вероятности системы (4) будут различными. В табл. 2 приведены частные случаи всех возможных границ интервалов для оценки функции плотности вероятности распределения координат *X*, *Y* и *Z* цвета. Границы интервалов $[a_i; b_i]$, $b_i > a_i > 0$ по каждой координате цвета определены в соответствии с данными табл. 1.

Случаи $b_1 - a_1 = b_2 - a_2 > 0$ соответствуют одинаковой длине интервалов $[a_1; b_1]$ и $[a_2; b_2]$ в системе (4), при которой функция плотности вероятности $f(t)$ трапецеидального вида приводится к треугольному виду [12]. В математической статистике для случая $[a_1 = a_2 = 0; b_1 = b_2 = 1]$ функция $f(t)$ определяет треугольное распределение Симпсона, являющееся частным случаем распределения Ирвина-Холла:

$$f_n(t) = \frac{1}{2(n-1)!}\sum_{k=0}^{n}(-1)^k\binom{n}{k}(t-k)^{n-1}\operatorname{sign}(t-k)$$

для суммы n равномерно распределенных на интервале [0;1] случайных величин при $n = 2$.

3. Цвет образован комбинацией всех трех ненулевых кодов компонентов основных цветов, т. е. $R \neq 0$, $G \neq 0$, $B \neq 0$. В этом случае функция плотности вероятности $f(t)$ распределения координаты цвета представляет собой математическую функцию кусочно-параболического вида.

Функция $f(t)$ вычисляется методом математической свертки функций плотности вероятности — для *X*-координаты $f_X(t) = f_{\xi_{X_r}+\xi_{X_g}}(t-s) \otimes f_{\xi_{X_b}}(s)$ — функции плотности вероятности трапецеидального вида, например (4), определенной на интервале $[a_1 + a_2; b_1 + b_2]$, и функции плотности вероятности, например, $f_{\xi_{X_b}}(s)$,

Таблица 2

Соотношения параметров функции плотности вероятности распределения координат цвета

Компоненты кода *RGB*		Функция плотности вероятности распределения координаты цвета		
		$f_X(t)$	$f_Y(t)$	$f_Z(t)$
$R \neq 0$, $G \neq 0$, $B = 0$	a_1	$RX_r - RX_r\Delta / 2$	$RY_r - RY_r\Delta / 2$	$RZ_r - RZ_r\Delta / 2$
	b_1	$RX_r + RX_r\Delta / 2$	$RY_r + RY_r\Delta / 2$	$RZ_r + RZ_r\Delta / 2$
	a_2	$GX_g - GX_g\Delta / 2$	$GY_g - GY_g\Delta / 2$	$GZ_g - GZ_g\Delta / 2$
	b_2	$GX_g + GX_g\Delta / 2$	$GY_g + GY_g\Delta / 2$	$GZ_g + GZ_g\Delta / 2$
$R = 0$, $G \neq 0$, $B \neq 0$	a_1	$GX_g - GX_g\Delta / 2$	$GY_g - GY_g\Delta / 2$	$GZ_g - GZ_g\Delta / 2$
	b_1	$GX_g + GX_g\Delta / 2$	$GY_g + GY_g\Delta / 2$	$GZ_g + GZ_g\Delta / 2$
	a_2	$BX_b - BX_b\Delta / 2$	$BY_b - BY_b\Delta / 2$	$BZ_b - BZ_b\Delta / 2$
	b_2	$BX_b + BX_b\Delta / 2$	$BY_b + BY_b\Delta / 2$	$BZ_b + BZ_b\Delta / 2$
$R \neq 0$, $G = 0$, $B \neq 0$	a_1	$RX_r - RX_r\Delta / 2$	$RY_r - RY_r\Delta / 2$	$RZ_r - RZ_r\Delta / 2$
	b_1	$RX_r + RX_r\Delta / 2$	$RY_r + RY_r\Delta / 2$	$RZ_r + RZ_r\Delta / 2$
	a_2	$BX_b - BX_b\Delta / 2$	$BY_b - BY_b\Delta / 2$	$BZ_b - BZ_b\Delta / 2$
	b_2	$BX_b + BX_b\Delta / 2$	$BY_b + BY_b\Delta / 2$	$BZ_b + BZ_b\Delta / 2$

равномерно распределенной на интервале $[a_3;\ b_3]$ случайной величины (см. табл. 1).

В статистическом смысле функция $f(t)$ представляет собой функцию плотности вероятности распределения суммы трех равномерно распределенных случайных величин [13, 14], каждая из которых определена на своем интервале $[a_i;\ b_i]$, $b_i > a_i > 0$, $i = 1, 2, 3$, причем: $a_1 \neq a_2 \neq a_3$, $b_1 \neq b_2 \neq b_3$.

В общем виде кусочно-параболическая функция $f(t)$ плотности вероятности распределения одной координаты цвета (на примере X-координаты) имеет следующий вид:

$$f(t) = \begin{cases} f_1(t), a_1 + a_2 + a_3 \leqslant t < a_1 + a_2 + b_3 \\ f_2(t), a_1 + a_2 + b_3 \leqslant t < a_1 + b_2 + a_3 \\ f_3(t), a_1 + b_2 + a_3 \leqslant t < a_1 + b_2 + b_3 \\ f_4(t), a_1 + b_2 + b_3 \leqslant t < b_1 + a_2 + a_3 \\ f_5(t), b_1 + a_2 + a_3 \leqslant t < b_1 + a_2 + b_3 \\ f_6(t), b_1 + a_2 + b_3 \leqslant t < b_1 + b_2 + a_3 \\ f_7(t), b_1 + b_2 + a_3 \leqslant t < b_1 + b_2 + b_3 \\ 0, 0 < t < a_1 + a_2 + a_3, t \geqslant b_1 + b_2 + b_3 \end{cases}, \quad (5)$$

где

$$f_1(t) = \underbrace{\frac{1}{b_3 - a_3} \int\limits_{t-b_3}^{a_1+a_2} f_{\xi_{X_r}+\xi_{X_g}}(s)\,ds}_{0} + \frac{1}{b_3 - a_3} \int\limits_{a_1+a_2}^{t-a_3} f_{\xi_{X_r}+\xi_{X_g}}(s)\,ds = \frac{1}{b_3 - a_3} \int\limits_{a_1+a_2}^{t-a_3} \frac{s - (a_1 + a_2)}{(b_1 - a_1)(b_2 - a_2)}\,ds =$$

$$= \frac{\left(t - (a_1 + a_2 + a_3)\right)^2}{2(b_1 - a_1)(b_2 - a_2)(b_3 - a_3)},$$

$$f_2(t) = \frac{1}{b_3 - a_3} \int\limits_{t-b_3}^{t-a_3} \frac{s - (a_1 + a_2)}{(b_1 - a_1)(b_2 - a_2)}\,ds = \frac{(t - a_3)^2 + 2(a_1 + a_2)(a_3 - b_3) - (t - b_3)^2}{2(b_1 - a_1)(b_2 - a_2)(b_3 - a_3)},$$

$$f_3(t) = \frac{1}{b_3 - a_3} \int\limits_{t-b_3}^{a_1+b_2} \frac{s - (a_1 + a_2)}{(b_1 - a_1)(b_2 - a_2)}\,ds + \frac{1}{b_3 - a_3} \int\limits_{a_1+b_2}^{t-a_3} \frac{1}{b_1 - a_1}\,ds =$$

$$= \frac{2(b_2 - a_2)\left(t - (a_1 + b_2 + a_3)\right) + (a_1 + b_2)^2}{2(b_1 - a_1)(b_2 - a_2)(b_3 - a_3)} + \frac{2(a_1 + a_2)\left(t - (a_1 + b_2 + b_3)\right) - (t - b_3)^2}{2(b_1 - a_1)(b_2 - a_2)(b_3 - a_3)},$$

$$f_4(t) = \frac{1}{b_3 - a_3} \int\limits_{t-b_3}^{t-a_3} \frac{1}{b_1 - a_1}\,ds = \frac{s}{(b_3 - a_3)(b_1 - a_1)}\Bigg|_{t-b_3}^{t-a_3} = \frac{1}{b_1 - a_1},$$

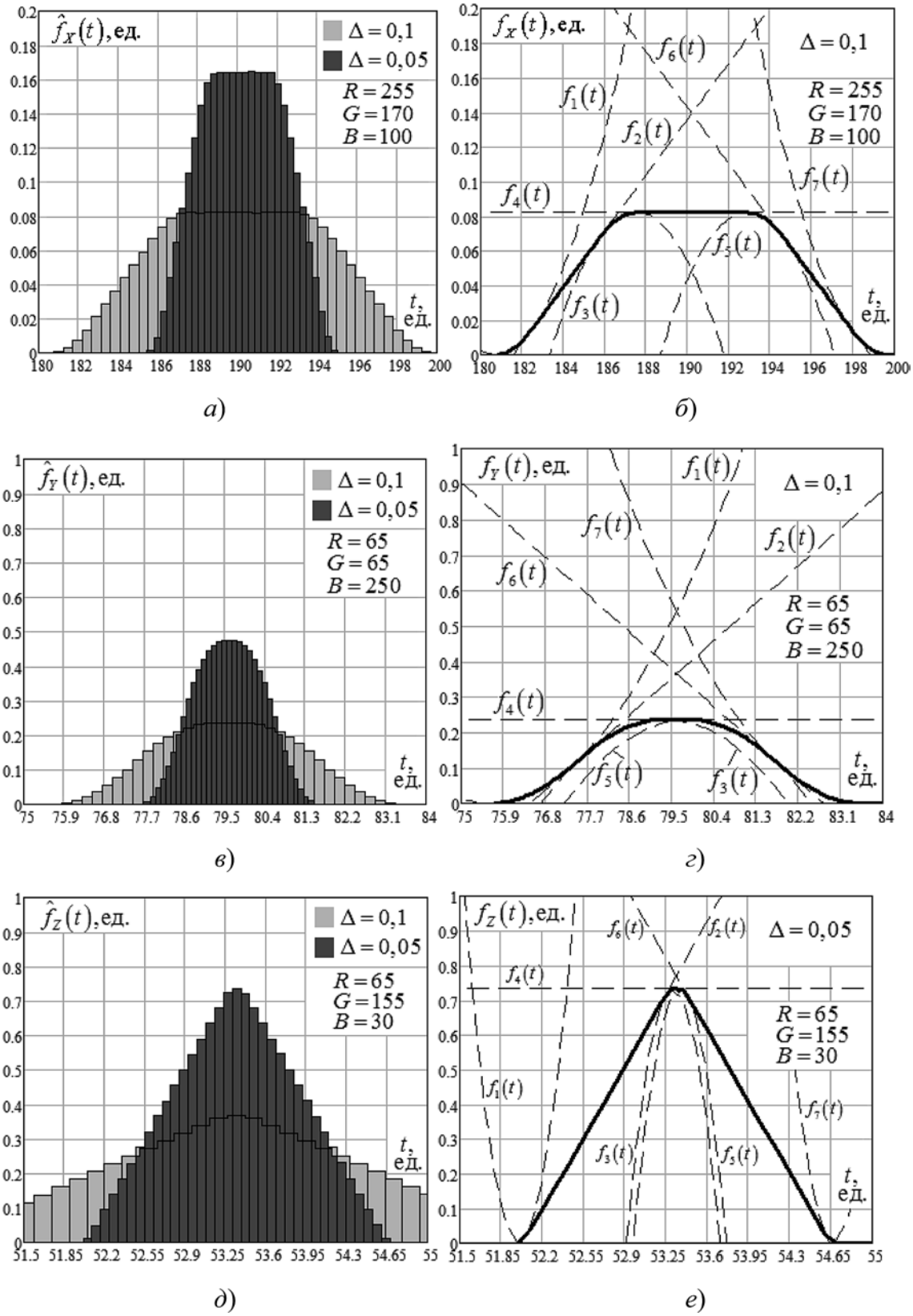
$$f_5(t) = \frac{1}{b_3 - a_3} \int\limits_{t-b_3}^{a_2+b_1} \frac{1}{b_1 - a_1}\,ds + \frac{1}{b_3 - a_3} \int\limits_{a_2+b_1}^{t-a_3} \frac{(b_1 + b_2) - s}{(b_1 - a_1)(b_2 - a_2)}\,ds =$$

$$= \frac{2(b_2 - a_2)\left((b_1 + a_2 + b_3) - t\right) + (a_2 + b_1)^2}{2(b_1 - a_1)(b_2 - a_2)(b_3 - a_3)} + \frac{2(b_1 + b_2)\left(t - (b_1 + a_2 + a_3)\right) - (t - a_3)^2}{2(b_1 - a_1)(b_2 - a_2)(b_3 - a_3)},$$

$$f_6(t) = \frac{1}{b_3 - a_3} \int\limits_{t-b_3}^{t-a_3} \frac{(b_1 + b_2) - s}{(b_1 - a_1)(b_2 - a_2)}\,ds = \frac{(t - b_3)^2 + 2(b_1 + b_2)(b_3 - a_3) - (t - a_3)^2}{2(b_1 - a_1)(b_2 - a_2)(b_3 - a_3)},$$

$$f_7(t) = \frac{1}{b_3 - a_3} \int\limits_{t-b_3}^{b_1+b_2} f_{\xi_{X_r}+\xi_{X_g}}(s)\,ds + \underbrace{\frac{1}{b_3 - a_3} \int\limits_{b_1+b_2}^{t-a_3} f_{\xi_{X_r}+\xi_{X_g}}(s)\,ds}_{0} =$$

$$= \frac{1}{b_3 - a_3} \int\limits_{t-b_3}^{b_1+b_2} \frac{(b_1 + b_2) - s}{(b_1 - a_1)(b_2 - a_2)}\,ds = \frac{\left(t - (b_1 + b_2 + b_3)\right)^2}{2(b_1 - a_1)(b_2 - a_2)(b_3 - a_3)}.$$



а) *б*) *в*) *г*) *д*) *е*)

Рис. 1. Оценки функций $\hat{f}_X(t)$ (*а*), $\hat{f}_Y(t)$ (*в*), $\hat{f}_Z(t)$ (*д*) плотности вероятности распределения *XYZ*-координат цвета и кусочно-параболические теоретические функции $f_X(t)$ (*б*), $f_Y(t)$ (*г*), $f_Z(t)$ (*е*). Сплошной линией на рис. *б*, *г*, *е* выделена огибающая теоретического распределения функций плотности вероятности из аппроксимирующих функций

Таблица 3

Выражения для вычисления коэффициентов функции плотности вероятности распределения координат цвета

Коэффициенты	Функция плотности вероятности распределения координаты цвета		
	$f_X(t)$	$f_Y(t)$	$f_Z(t)$
a_1	$RX_r - RX_r\Delta / 2$	$RY_r - RY_r\Delta / 2$	$RZ_r - RZ_r\Delta / 2$
b_1	$RX_r + RX_r\Delta / 2$	$RY_r + RY_r\Delta / 2$	$RZ_r + RZ_r\Delta / 2$
a_2	$GX_g - GX_g\Delta / 2$	$GY_g - GY_g\Delta / 2$	$GZ_g - GZ_g\Delta / 2$
b_2	$GX_g + GX_g\Delta / 2$	$GY_g + GY_g\Delta / 2$	$GZ_g + GZ_g\Delta / 2$
a_3	$BX_b - BX_b\Delta / 2$	$BY_b - BY_b\Delta / 2$	$BZ_b - BZ_b\Delta / 2$
b_3	$BX_b + BX_b\Delta / 2$	$BY_b + BY_b\Delta / 2$	$BZ_b + BZ_b\Delta / 2$

Значения коэффициентов a_i, b_i в выражениях системы (5) различны для функций плотности вероятности распределения X-координаты $f_X(t)$, Y-координаты $f_Y(t)$ и Z-координаты $f_Z(t)$. Точные выражения для вычисления коэффициентов a_i, b_i системы (5) приведены в табл. 3.

Важно заметить, что система (5) справедлива при условии:

$$b_1 - a_1 \geqslant b_2 - a_2 \geqslant b_3 - a_3 > 0. \qquad (6)$$

Для соблюдения условия (6) интервалы распределения могут быть предварительно ранжированы с переопределением индексов коэффициентов a_i, b_i.

На рис. 1. приведены оценки функций $\hat{f}_X(t)$, $\hat{f}_Y(t)$, $\hat{f}_Z(t)$ плотности вероятности распределения *XYZ*-координат цвета, полученные в результате моделирования, и кусочно-параболические теоретические функции $f_X(t), f_Y(t), f_Z(t)$ плотности вероятности распределения *XYZ*-координат цвета, построенные в соответствии с формулой (5), при ненулевых значениях компонентов кода *RGB*.

Анализ формул (4), (5) показывает, что $f(t)$ является функцией параметра технологического разброса Δ, коэффициентов профиля ЖК панели и значений компонентов кода *RGB*. Этот факт означает, что при одних ненулевых значениях компонентов кода *RGB* распределение $f(t)$ *XYZ*-координат цвета имеет вид трапецеидальной (треугольной) функции, при других, также ненулевых — вид кусочно-параболической функции.

4. Распределение координат цветности

Как следует из анализа системы (5), при вычислении теоретической функции плотности вероятности распределения координат X, Y, Z цвета получаются сложные математические функции кусочно-параболического вида. При этом абсолютные значения *XYZ*-координат цвета определяют только числитель выражения (2) и используются при расчете (x,y)-координат цветности.

Для случая ненулевых значений компонентов кода *RGB*, функция $f_{X+Y+Z}(t)$ плотности вероятности распределения модуля цвета, т. е. знаменателя выражения (2) — суммы $X + Y + Z$ значений координат цвета, с учетом технологического разброса параметров изготовления ЖК панели будет соответствовать функции плотности вероятности распределения суммы

$$\xi_{X_r} + \xi_{X_g} + \xi_{X_b} + \xi_{Y_r} + \xi_{Y_g} + \xi_{Y_b} + \xi_{Z_r} + \xi_{Z_g} + \xi_{Z_b}$$

равномерно распределенных случайных величин на неидентичных интервалах.

Нетрудно показать, что закон распределения такой суммы может быть представлен с достаточной степенью точности нормальным законом распределения [15], в функции плотности вероятности которого будут присутствовать компоненты сумм математических ожиданий и дисперсий всех девяти равномерно распределенных случайных величин:

$$f_{X+Y+Z}(t) \approx \frac{1}{\sqrt{2\pi D_{XYZ}}} \exp\left(-\frac{(t - M_{XYZ})^2}{2D_{XYZ}}\right),$$

где

$$M_{XYZ} = (X_r + Y_r + Z_r)R + (X_g + Y_g + Z_g)G + (X_b + Y_b + Z_b)B,$$

$$D_{XYZ} = \frac{\Delta^2\left(R^2\left(X_r^2 + Y_r^2 + Z_r^2\right) + G^2\left(X_g^2 + Y_g^2 + Z_g^2\right) + B^2\left(X_b^2 + Y_b^2 + Z_b^2\right)\right)}{12}.$$

Точное аналитическое выражение функции $f_{X+Y+Z}(t)$ плотности вероятности распределения модуля цвета в явном виде оказывается малопригодным для практического использования. Оно содержит более 500 подинтервалов определения $f_{X+Y+Z}(t)$, каждый участок которой аппроксимируется функцией полинома восьмого порядка, а само значение модуля цвета $X + Y + Z$ распределено на интервале (0;255).

Кроме того, анализ формулы (2) показывает, что при вычислении отношения координаты цвета и модуля цвета в числителе и в знаменателе (2) используются одни и те же значения *XYZ*. Как следствие, случайные величины $\xi_{X_r} + \xi_{X_g} + \xi_{X_b}$, $\xi_{Y_r} + \xi_{Y_g} + \xi_{Y_b}$, $\xi_{Z_r} + \xi_{Z_g} + \xi_{Z_b}$, $\xi_{X_r} + \xi_{X_g} + \xi_{X_b} + \xi_{Y_r} + \xi_{Y_g} + \xi_{Y_b} + \xi_{Z_r} + \xi_{Z_g} + \xi_{Z_b}$ не могут рассматриваться как независимые. Таким образом, вычисление теоретических функций плотности вероятности

$$f_x(t) = f_X(t) / f_{X+Y+Z}(t),\ f_y(t) = f_Y(t) / f_{X+Y+Z}(t)$$

потребует введения дополнительных компонент условных вероятностей, существенно усложняющих аналитические выкладки. В связи с этим оценка влияния параметра технологического разброса Δ на распределение (x, y)-координат цветности индицируемого изображения выполнена с использованием средств компьютерного моделирования.

Результаты моделирования приведены на рис. 2. Числовые значения координат цветности, среднего значения координат цветности (математические ожидания M_x, M_y) и гистограмм $p(x)$, $p(y)$ распределения (x, y)-координат цветности получены в среде моделирования Mathcad 15.0. Для генерации последовательности случайных величин использовался встроенный датчик среды.

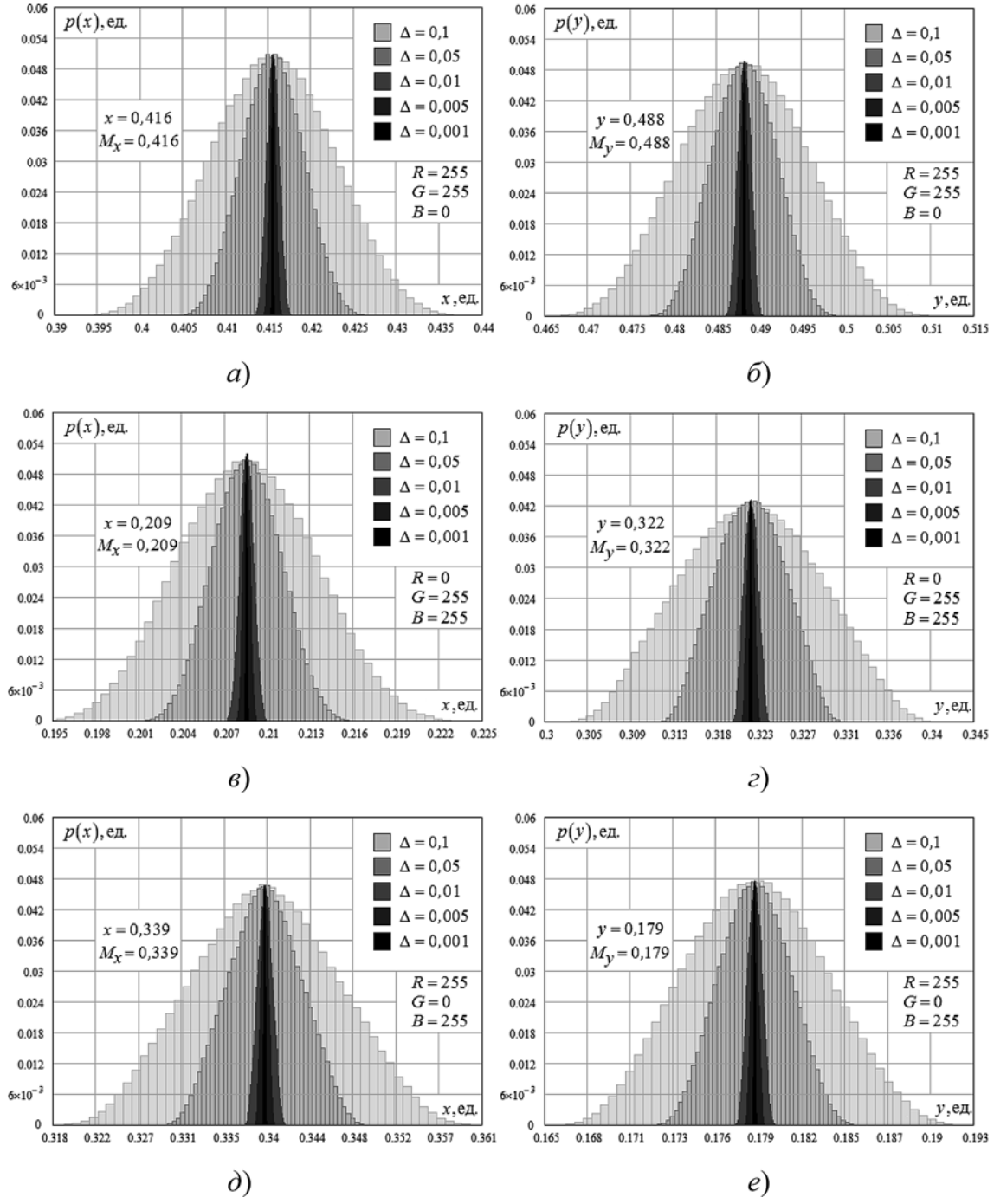


Рис. 2. Гистограммы распределения (x, y)-координат цветности:
а, *б* — желтый цвет; *в*, *г* — голубой цвет; *д*, *е* — пурпурный цвет

Таблица 4

Оценки дисперсий распределения (*x*, *y*)-координат цветности

Δ, ед.	Желтый цвет		Голубой цвет		Δ, ед.	Красный цвет		Зеленый цвет	
	D_x, ед.^{2}	D_y, ед.^{2}	D_x, ед.^{2}	D_y, ед.^{2}		D_x, ед.^{2}	D_y, ед.^{2}	D_x, ед.^{2}	D_y, ед.^{2}
0,1	$4,7 \cdot 10^{-5}$	$5,0 \cdot 10^{-5}$	$2,1 \cdot 10^{-5}$	$4,8 \cdot 10^{-5}$	0,1	$8,5 \cdot 10^{-5}$	$8,2 \cdot 10^{-5}$	$5,4 \cdot 10^{-5}$	$7,6 \cdot 10^{-5}$
0,05	$1,2 \cdot 10^{-5}$	$1,3 \cdot 10^{-5}$	$5,3 \cdot 10^{-6}$	$1,2 \cdot 10^{-5}$	0,05	$2,1 \cdot 10^{-5}$	$2,0 \cdot 10^{-5}$	$1,4 \cdot 10^{-5}$	$1,9 \cdot 10^{-5}$
0,01	$4,7 \cdot 10^{-7}$	$5,0 \cdot 10^{-7}$	$2,1 \cdot 10^{-7}$	$4,8 \cdot 10^{-7}$	0,01	$8,5 \cdot 10^{-7}$	$8,2 \cdot 10^{-7}$	$5,4 \cdot 10^{-7}$	$7,6 \cdot 10^{-7}$
0,005	$1,2 \cdot 10^{-7}$	$1,3 \cdot 10^{-7}$	$5,3 \cdot 10^{-8}$	$1,2 \cdot 10^{-7}$	0,005	$2,1 \cdot 10^{-7}$	$2,0 \cdot 10^{-7}$	$1,4 \cdot 10^{-7}$	$1,9 \cdot 10^{-7}$
0,001	$4,7 \cdot 10^{-9}$	$5,0 \cdot 10^{-9}$	$2,1 \cdot 10^{-9}$	$4,8 \cdot 10^{-9}$	0,001	$8,5 \cdot 10^{-9}$	$8,2 \cdot 10^{-9}$	$5,4 \cdot 10^{-9}$	$7,6 \cdot 10^{-9}$
Δ, ед.	Пурпурный цвет		Белый цвет		Δ, ед.	Синий цвет		Коричневый цвет	
	D_x, ед.^{2}	D_y, ед.^{2}	D_x, ед.^{2}	D_y, ед.^{2}		D_x, ед.^{2}	D_y, ед.^{2}	D_x, ед.^{2}	D_y, ед.^{2}
0,1	$4,9 \cdot 10^{-5}$	$1,9 \cdot 10^{-5}$	$2,7 \cdot 10^{-5}$	$3,2 \cdot 10^{-5}$	0,1	$2,5 \cdot 10^{-5}$	$6,0 \cdot 10^{-6}$	$5,8 \cdot 10^{-5}$	$5,4 \cdot 10^{-5}$
0,05	$1,2 \cdot 10^{-5}$	$4,8 \cdot 10^{-6}$	$6,7 \cdot 10^{-6}$	$8,0 \cdot 10^{-6}$	0,05	$6,3 \cdot 10^{-6}$	$1,5 \cdot 10^{-6}$	$1,4 \cdot 10^{-5}$	$1,3 \cdot 10^{-5}$
0,01	$4,9 \cdot 10^{-7}$	$1,9 \cdot 10^{-7}$	$2,7 \cdot 10^{-7}$	$3,2 \cdot 10^{-7}$	0,01	$2,5 \cdot 10^{-7}$	$6,0 \cdot 10^{-8}$	$5,8 \cdot 10^{-7}$	$5,4 \cdot 10^{-7}$
0,005	$1,2 \cdot 10^{-7}$	$4,8 \cdot 10^{-8}$	$6,7 \cdot 10^{-8}$	$8,0 \cdot 10^{-8}$	0,005	$6,3 \cdot 10^{-8}$	$1,5 \cdot 10^{-8}$	$1,4 \cdot 10^{-7}$	$1,3 \cdot 10^{-7}$
0,001	$4,9 \cdot 10^{-9}$	$1,9 \cdot 10^{-9}$	$2,7 \cdot 10^{-9}$	$3,2 \cdot 10^{-9}$	0,001	$2,5 \cdot 10^{-9}$	$6,0 \cdot 10^{-10}$	$5,8 \cdot 10^{-9}$	$5,4 \cdot 10^{-9}$

Моделирование проводилось на персональном компьютере ASUS K56CB-X0391H со следующими характеристиками: процессор Intel(R) Core(TM) i5-3337U, 4 ядра, тактовая частота 1,8 ГГц, оперативная память 6 Гбайт под управлением операционной системы Windows 8.1.

Коэффициенты профиля ЖК панели в выражениях (1), (3) в экспериментах приняты равными: $X_r = 0{,}478$; $X_g = 0{,}299$; $X_b = 0{,}175$; $Y_r = 0{,}263$, $Y_g = 0{,}650$; $Y_b = 0{,}081$; $Z_r = 0{,}020$; $Z_g = 0{,}160$; $Z_b = 0{,}908$. Объем выборки при моделировании каждой случайной величины ξ в системе уравнений (3) составил $8 \cdot 10^6$.

Одновременно вычислялись оценки дисперсий D_x, D_y распределения (x, y)-координат цветности при различных значениях параметра технологического разброса Δ. Оценки дисперсий для цветов, входящих в состав цветовой палитры средства бортовой индикации, приведены в табл. 4.

Заключение

Результаты экспериментов показывают, что приемлемой для практики является повторяемость в технологии изготовления ЖК панелей, при которой достигается значение $\Delta \leqslant 0{,}001$. В этом случае значение дисперсий D_x, D_y распределений координат цветности (цвета) не превосходит разрешающей способности восприятия наблюдателя и современных и перспективных инструментальных средств (колориметр), используемых при прямом измерении значений координат (x, y) [7]. По результатам замеров существующие сегодня технологии изготовления иностранных ЖК панелей обеспечивают разброс $\Delta \leqslant 0{,}05$.

Выражения в табл. 1—3, в системах (4), (5) приведены в аналитическом виде и могут быть использованы для определения функции плотности вероятности распределения координат цвета как для стандарта МКО 1931 г., так и для стандарта МКО 1964 г. с различными системами оценки баланса белого цвета: D-75, D-65, D-55, D-50 и др., в пределах каждой из которых компоненты профиля ЖК панели X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b базового преобразования (1), (2) являются различными.

Полученные в результате расчетов аналитические выражения для описания функций плотности вероятности распределения координат цвета выходят за рамки предметной области колориметрии. Математические выражения системы (5) расширяют границы применимости закона распределения Ирвина-Холла, где каждая равномерно распределенная случайная величина $\xi_i \in [0; 1]$, на общий случай суммирования равномерно распределенных случайных величин, при котором: $\xi_i \in [a_i; b_i]$, $a_1 \neq a_2 \neq a_3$, $b_1 \neq b_2 \neq b_3$, $b_i > a_i > 0$.

Список литературы

1. **Жаринов И. О., Жаринов О. О.** Бортовые средства отображения информации на плоских жидкокристаллических панелях: учеб. пособие. Информационно-управляющие системы, СПб.: ГУАП, 2005. 144 с.
2. **Жданов В.** Передовые технологии фирмы Sharp в изготовлении ЖК-дисплеев для различных применений // Современная электроника. 2006. № 1. С. 14—19.
3. **Сомарин А.** Модули ЖК-дисплеев для авионики // Компоненты и технологии. 2005. № 3. С. 56—62.
4. **Barber S. et al.** US Patent 7,417,641 B1: Aeronautical chart display apparatus and method, Aug. 26. 2008.

5. **Дятлов В. М.** Разработка технических решений по разработке торцевого модуля задней подсветки // Научно-технический вестник "Военная электроника и электротехника". Труды 22 ЦНИИИ Минобороны России, 2010. Вып. 62. С. 280—291.

6. **Дятлов В. М.** Разработка и исследование конструкции стеклопакета жидкокристаллического экрана // Научно-технический вестник "Военная электроника и электротехника". Труды 22 ЦНИИИ Минобороны России, 2010. Вып. 62. С. 270—279.

7. **Жаринов И. О., Жаринов О. О.** Исследование распределения оценки разрешающей способности преобразования Грассмана в системах кодирования цвета, применяемых в авионике // Программная инженерия. 2014. № 8. С. 40—47.

8. **Жаринов И. О., Жаринов О. О.** Исследование свойства равноконтрастности цветовых пространств, применяемых в авионике // Программная инженерия. 2014. № 11. С. 35—43.

9. **Костишин М. О., Жаринов И. О., Жаринов О. О.** Исследование визуальных характеристик средств отображения пилотажно-навигационных параметров и геоинформационных данных в авионике // Информационно-управляющие системы. 2014. № 4. С. 61—67.

10. **Bradley D. M., Gupta R. C.** On the distribution of the sum of *n* non-identically distributed uniform random variables // Annals of the Institute of Statistical Mathematics. 2002. Vol. 54, N. 3. P. 689—700.

11. **Sadooghi-Alvandi S. M., Nematollahi A. R., Habibi R.** On the distribution of the sum of independent uniform random variables // Statistical papers. 2009. Vol. 50, N. 1. P. 171—175.

12. **Gard M., Choudhary S., Kalla S. L.** On the sum of two triangular random variables // International journal of optimization: theory, methods and applications. 2009. Vol. 1, N. 3. P. 279—290.

13. **Buonocore A., Pirozzi E., Caputo L.** A note on the sum of uniform random variables // Statistics & Probability Letters. 2009. Vol. 79, N. 19. P. 2092—2097.

14. **Potuschak H., Muller W. G.** More on the distribution of the sum of uniform random variables // Statistical papers. 2009. Vol. 50, N. 1. P. 177—183.

15. **Murakami H.** A saddlepoint approximation to the distribution of the sum of independent non-identically uniform random variables // Statistica Neerlandica. 2014. Vol. 68, N. 4. P. 267—275.

I. O. Zharinov, Associate Professor, e-mail: igor_rabota@pisem.net, Chief of Department, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (University ITMO), Chief of Learning-Scientists Center, SPb Scientific Design Bureau “Electroavtomatica" n. a. P. A. Efimov,
O. O. Zharinov, Associate Professor, Saint-Petersburg State University of Aerospace Equipment

Theoretical Evaluation of the Probability Density Function of Color Coordinates in On-Board Indication Equipment

The problem of evaluation of the probability density function of color coordinates that are used to display an image on on-board indication equipment is considered. A statistical approach for characterization of color coordinates distribution is necessary because of existing technological variations of the LCD-panels parameters. It is shown that the probability density function of color coordinates depends upon values of RGB codes, used in the color palette, and in particular cases refers to: uniform distribution, trapezoid-like function, triangle function and piecewise parabolic function. The analytical expressions for the probability density function of color coordinates for each particular type of mathematical models of technological variations and values of the LCD-panel profile coefficients are presented. The theoretical expressions for probability density function of color coordinates are confirmed by mathematical simulation, and the histograms representing the distributions of chromaticity coordinates were obtained during simulation. The result of the research establishes some technical requirements for technological variations of the values of LCD-panel profile coefficients, in order to ensure repeatability of color coordinates in serial manufacturing of LCD-panels.

Keywords: *color coordinates, indication systems, probability density function, sum of uniformly distributed random values, Irvin-Hall distibution*

References

1. **Zharinov I. O., Zharinov O. O.** *Bortovye sredstva otobrazhenija informacii na ploskih zhidkokristallicheskih paneljah* (Onboard Display on Flat Liquid Crystal Panels), Saint-Petersburg, Informacionno-upravljajushhie sistemy, 2005, 144 p (in Russian).

2. **Zhdanov V.** *Sovremennaja jelektronika*, 2006, no. 1, pp. 14—19 (in Russian).

3. **Somarin A.** *Komponenty i tehnologii*, 2005, no. 3, pp. 56—62 (in Russian).

4. **Barber S.** US Patent 7,417,641 B1: Aeronautical chart display apparatus and method, Aug. 26. 2008.

5. **Djatlov V. M.** *Nauchno-tehnicheskij vestnik "Voennaja jelektronika i jelektrotehnika",* 2010, vol. 62, pp. 280—291 (in Russian).

6. **Djatlov V. M.** *Nauchno-tehnicheskij vestnik "Voennaja jelektronika i jelektrotehnika",* 2010, vol. 62, pp. 270—279 (in Russian).

7. **Zharinov I. O., Zharinov O. O.** *Programmnaya ingeneria,* 2014, no. 8, pp. 40—47 (in Russian).

8. **Zharinov I. O., Zharinov O. O.** *Programmnaya ingeneria,* 2014, no. 11, pp. 35—43 (in Russian).

9. **Kostishin M. O., Zharinov I. O., Zharinov O. O.** *Informacionno-upravljajushhie sistemy,* 2014, no. 4, pp. 61—67 (in Russian).

10. **Bradley D. M., Gupta R. C.** On the distribution of the sum of *n* non-identically distributed uniform random variables, *Annals of the Institute of Statistical Mathematics*, 2002, vol. 54, no. 3, pp. 689—700.

11. **Sadooghi-Alvandi S. M., Nematollahi A. R., Habibi R.** On the distribution of the sum of independent uniform random variables, *Statistical papers*, 2009, vol. 50, no. 1, pp. 171—175.

12. **Gard M., Choudhary S., Kalla S. L.** On the sum of two triangular random variables, *International journal of optimization: theory, methods and applications*, 2009, vol. 1, no. 3, pp. 279—290.

13. **Buonocore A., Pirozzi E, Caputo L.** A note on the sum of uniform random variables, *Statistics & Probability Letters*, 2009, vol. 79, no. 19, pp. 2092—2097.

14. **Potuschak H., Muller W. G.** More on the distribution of the sum of uniform random variables, *Statistical papers*, 2009, vol. 50, no. 1, pp. 177—183.

15. **Murakami H.** A saddlepoint approximation to the distribution of the sum of independent non-identically uniform random variables, *Statistica Neerlandica*, 2014, vol. 68, no. 4, pp. 267—275.

УДК 004.75+004.41

А. С. Шундеев, канд. физ.-мат. наук, вед. науч. сотр., e-mail: alex.shundeev@gmail.com,
НИИ механики МГУ им. М. В. Ломоносова

Изучение механизмов ввода–вывода в процессе обучения программированию

Изложен подход к изучению механизмов ввода—вывода в рамках учебного курса "Работа на ЭВМ и программирование" для студентов первого и второго курсов механико-математического факультета МГУ им. М. В. Ломоносова. Подход основан на формировании набора рецептов. Каждый рецепт представляет собой взаимосвязанный набор общеупотребительных сценариев, практик и примеров использования. Рецепт посвящен конкретной теме и содержит необходимые теоретические сведения, примеры программ, методические указания по изложению материала. Рассмотрены три темы — чтение последовательности чисел из файла, буферизованный ввод—вывод, чтение и запись структурированных данных.

Ключевые слова: *программирование, механизмы ввода—вывода, обучение*

Введение

Механизмы ввода—вывода являются важной темой в учебных курсах по компьютерным наукам. Обучение компьютерным наукам, как правило, начинается с изучения первого языка программирования. В свою очередь, изучение языка программирования начинается с объяснения базовых средств языка, позволяющих программе взаимодействовать с внешним миром. Это прежде всего касается стандартного ввода, связанного с чтением данных с клавиатуры, и стандартного вывода, связанного с печатью текста на экран дисплея.

На следующем этапе обучения студент, как правило, знакомится с элементами системного программированием [1, 2]. При этом оказывается, что варианты использования механизмов ввода—вывода не ограничиваются стандартными задачами работы с текстовыми или бинарными файлами. В мире POSIX-систем [3] механизмы ввода—вывода являются основой для организации межпроцессного взаимодействия.

Завершающий этап обучения может быть связан с изучением сетевого программирования [2] и технологий построения распределенных программных систем. Сетевое программирование базируется на использовании сетевых сокетов, представляющих собой специализированные средства ввода—вывода.

Из приведенных выше примеров можно сделать вывод о том, что механизмы ввода—вывода являются сквозной и важной темой, присутствующей в разнообразных учебных курсах по компьютерным наукам, с которой учащийся сталкивается практически на всех этапах своего обучения. По этой причине рассматриваемая тема требует постоянного совершенствования существующих и выработки новых, более эффективных методов обучения. Один из подходов состоит в формировании набора общеупотребительных сценариев, практик и примеров использования механизмов ввода—вывода при решении тех или иных задач. Описанию такого набора практик и посвящена данная статья. Подобный подход в целом не является новым. Он успешно апробирован в области изучения языка Си++ и его стандартной библиотеки [4]. В статье рассмотрены три темы: чтение последовательности чисел из файла, буферизованный ввод—вывод, чтение и запись структурированных данных.

Данная статья носит методический характер. Она написана на основе материалов, которые автор использует при проведении практических занятий в рамках курса "Работа на ЭВМ и программирование" на механико-математическом факультете МГУ им. М. В. Ломоносова.

Чтение последовательности чисел из файла

Рассмотрим следующую задачу. Во входном файле `input.txt` содержится последовательность вещественных чисел. Вначале будем предполагать, что их число заранее известно (например, равно `5`). Требуется считать эту последовательность и каким-то образом обработать. Соответствующая программа схематично представлена на листинге 1.

В строке `05` открывается на чтение входной файл `input.txt`. Указатель на состояние открытого файла помещается в переменную `f`. В рамках цикла (строки `06–09`) выполняется ровно пять итераций. Каждая итерация начинается с чтения очередного вещественного числа (значения типа doubl e) из входного файла и сохранение этого числа в переменную `a` (строка `07`). Строка `08` оставлена для обработки считанного значения.

Приведенный фрагмент программы рассчитан на идеальный случай, когда на вход всегда подаются только корректные входные данные и в процессе выполнения программы не возникает ошибок ввода—вывода.

Модифицированный фрагмент программы, в который включены необходимые проверки, приведен на листинге 2.

В программе на листинге 2 учитываются следующие нештатные ситуации. Операция открытия файла может окончиться неудачно. Например, указанного файла

```
01.  int     i;
02.  double  a;
03.  FILE   *f;
04.  ...
05.  f = fopen("input.txt", "r");
06.  for(i = 0; i < 5; i ++) {
07.      fscanf(f, "%lf", &a);
08.      // Обработать считанное a
09.  }
```

Листинг 1. Чтение чисел из файла

может просто не существовать, или у пользователя, запустившего программу, отсутствуют соответствующие права на чтение. В этом случае функция fopen возвратит нулевой указатель. Добавлена проверка (строки 06–07). Если возвращен нулевой указатель, то программа завершается с кодом ошибки - 1.

Попытка считывания вещественного числа может окончиться неудачно. Чтобы учесть эту ситуацию, необходимо вспомнить определение функции fscanf. Данная функция возвращает либо число успешно считанных значений, либо константу EOF. Константа EOF возвращается либо в случае достижения конца файла, либо в случае возникновения ошибки ввода—вывода. В нашем случае функция fscanf может возвратить значение 0 (ничего не было считано), 1 (успешное считывание) или EOF.

Соответствующим образом было изменено тело цикла (строки 09–13). Тело цикла заменено на оператор if-else. В рамках проверочного выражения вызывается функция fscanf. В случае успешно считанного вещественного числа осуществляется переход к его обработке (строка 10). В противном случае программа завершается с кодом ошибки –1 (строка 13).

Успешное выполнение цикла означает ситуацию, когда в начале входного файла записано пять вещественных чисел, и все они были успешно считаны. Однако содержимое входного файла может быть признано корректным только в том случае, если файл больше ничего не содержит. Исключение могут составлять пробелы или другие разделители.

Сделаем одно важное отступление, касающееся функции fscanf или другой стандартной функции ввода. Пусть в результате вызова функции fscanf было успешно считано значение и во входном файле уже отсутствуют непрочитанные данные. В этом случае в состоянии открытого файла (адрес которого хранится в переменной f) еще не установлен признак конца файла, который может быть проверен стандартной функцией feof. Требуется выполнение еще одной операции чтения, которая закончится неудачно, но приведет к установке признака конца файла. Подобная проверка достижения конца файла реализована в строках 14–16.

Чтобы убедиться в корректности входного файла в строке 14 осуществляется контрольный вызов функции fscanf. Нас не интересует значение, которое возможно будет считано в результате этого вызова. После этого в строке 15 вызывается функция feof, которая проверяет, был ли достигнут конец файла в результате последнего вызова функции fscanf. Если конец файла не был достигнут, то программа завершается с кодом ошибки –1.

```
01.  int     i;
02.  double  a;
03.  FILE   *f;
04.  ...
05.  f = fopen("input.txt", "r");
06.  if(!f)
07.      return -1;
08.  for(i = 0; i < 5; i ++)
09.      if(fscanf(f, "%lf", &a) == 1){
10.          // Обработать считанное a
11.      }
12.      else
13.          return -1;
14.  fscanf(f, "%lf", &a);
15.  if(!feof(f))
16.      return -1;
```

Листинг 2. Чтение чисел из файла с необходимыми проверками

```
01.  int     i;
02.  double  a;
03.  FILE   *f;
04.  ...
05.  f = fopen("input.txt", "r");
06.  if(!f)
07.      return -1;
08.  while(fscanf(f, "%lf", &a) == 1) {
09.      // Обработать считанное a
10.  }
11.  if(!feof(f))
12.      return -1;
```

Листинг 3. Чтение чисел из файла (количество чисел неизвестно)

Сравнение листингов 1 и 2 позволяет сделать важный вывод, непосредственно касающийся программной инженерии. Программа содержит не столько реализацию целевого алгоритма, сколько обработку всевозможных нештатных ситуаций. В реальной эксплуатации программы такие нештатные ситуации могут и не возникнуть, но их обработка должна быть предусмотрена.

Приведем фрагмент программы (листинг 3) для случая, когда размер последовательности чисел во входном файле заранее не известен.

В программе на листинге 3 модифицирован цикл (строки 08–10). В рамках проверочного выражения осуществляется попытка считать очередное число. В случае неудачной попытки цикл прекращается. Неудачная попытка возникнет либо в случае достижения конца файла, либо, если файл содержит последовательность символов, не являющуюся числом. Строки 11–12 отделяют эти два случая друг от друга.

Буферизованный ввод—вывод

Если просмотреть содержимое стандартного заголовочного файла stdio.h, то в нем можно обнаружить три глобальные переменные stdin, stdout и stderr типа FILE*. Данные переменные содержат указатели на состояния трех открытых файлов, которые называют стандартный поток ввода, стандартный поток вывода и стандартный протокол. По умолчанию стандартный поток ввода связан с чтением данных с терминального устройства, а стандартные потоки вывода и протокола — с печатью на терминальное устройство.

Отметим связь между парой стандартных функций printf, scanf и парой стандартных функций fprintf, fscanf. Вызов функции вида printf(…) эквивалентен вызову функции вида fprintf(stdout, …). Вызов функции вида scanf(…) эквивалентен вызову функции fscanf(stdin, …).

Несмотря на то что stdout и stderr оба представляют собой потоки вывода, реализованы эти потоки по-разному. Например, при выполнении следующего фрагмента программы

```
fprintf(stdout, "stdout");
fprintf(stderr, "stderr");
```

на экране будет напечатана строка

```
stderrstdout
```

Строчка, которая была отправлена в стандартный протокол, будет напечатана раньше, чем строчка, отправленная в стандартный поток вывода. Хотя в программе сначала осуществляется печать в стандартный поток вывода, и только затем — в стандартный протокол.

Чтобы объяснить данный феномен, можно провести один эксперимент, идея которого взята из работы [1]. Для проведения эксперимента нужен файл достаточного размера. Автор статьи использовал текстовый файл hamlet.txt[1], имеющий размер 191 734 байта. Эксперимент базируется на использовании команды dd. Команда dd позволяет копировать данные порциями фиксированного, заранее определенного размера.

Скопируем вначале файл hamlet.txt в новый файл h1.txt порциями по одному байту:

```
dd bs=1 count=191 734 if=hamlet.txt of=h1.txt
```

В результате выполнения команды была получена следующая статистика:

```
скопировано 191 734 байт (191 kB), 0,297755 c,
643 kB/c
```

Затем скопируем файл hamlet.txt в новый файл h2.txt порциями по 1024 байт:

```
dd bs=1024 count=188 if=hamlet.txt of=h2.txt
```

В результате выполнения команды была получена следующая статистика:

```
скопировано 191 734 байт (191 kB), 0,00183474 c,
104 MB/c
```

Можно сделать следующий вывод. Копировать по одному байту крайне невыгодно. Более эффективно копировать крупными порциями данных. Остается открытым вопрос выбора размера этих порций. Необходимо отметить, что весь файловый ввод—вывод осуществляется в терминах блоков. Размер блока для операций ввода—вывода определяется командой stat. Например, выполнив команду

```
stat -c"%o" h1.txt
```

автор статьи на своем компьютере получил для обычного файла h1.txt размер блока, равный 4096 байтам. Поэтому в рассмотренных примерах использования команды dd оптимальным значением параметра bs было бы 4096. Для терминального устройства размер блока для операций ввода—вывода равен 1024 байтам.

Учитывая приведенные выше соображения, в стандартной библиотеке языка Си [3] реализуются три типа механизма вывода (небуферизованный, буферизованный, построчно-буферизованный). При небуферизованном выводе операция записи (соответствующий системный вызов write) выполняется немедленно без задержки. По умолчанию стандартный протокол stderr реализует небуферизованный вывод. При буферизованном выводе данные вначале накапливаются в буфере, размер которого совпадает с размером блока соответствующего файла, в который осуществляется запись. Только при полном заполнении буфера осуществляется операция записи его содержимого. Отметим, что текущее состояние буфера может быть принудительно записано при помощи команды fflush. Построчно-буферизованный вывод отличается от буферизованного тем, что запись текущего состояния буфера осуществляется также, когда встречается символ перехода на новую строку. Стандартный поток вывода по умолчанию реализует построчно-буферизованный механизм вывода.

Можно вернуться к рассмотренному вначале примеру с печатью в стандартный поток вывода и в стандартный протокол. При выполнении фрагмента программы

```
fprintf(stdout, "stdout");
fflush(stdout);
fprintf(stderr, "stderr");
```

на экране будет напечатана строка

```
stdoutstderr
```

В данном примере сначала в рамках первого вызова функции fprintf в буфер была помещена строка "stdout". Данный вызов не повлек за собой выполнение операции записи. Далее следует вызов функции fflush, которая инициирует выполнение операции записи текущего состояния буфера с последующей его очисткой. Второй вызов функции fprintf сразу инициирует выполнение операции записи строки "stderr".

При выполнении фрагмента программы

```
fprintf(stdout, "stdout\n");
fprintf(stderr, "stderr");
```

на экране будет напечатана строка

```
stdout
stderr
```

В данном примере в рамках первого вызова функции fprintf в буфер помещается строка "stdout\n". Так как эта строка заканчивается символом перехода на новую строку, то сразу инициируется выполнение операции записи текущего состояния буфера с последующей его очисткой. Второй вызов функции fprintf сразу инициирует выполнение операции записи строки "stderr".

Чтение и запись структурированных данных

Ранее рассматривались примитивные типы данных в контексте решения задач ввода—вывода. В данном разделе будут рассмотрены структурированные данные. В качестве демонстрационного примера будет использоваться упрощенная структура student _ t (листинг 4), содержащая информацию о студенте. Структура имеет два поля: name — массив символов длины 10 для хранения фамилии студента и numb — 4-байтовое целое число для хранения номера зачетной книжки.

Будем рассматривать задачу записи структуры student _ t в бинарный файл и ее чтения из него. Разберем первое решение этой задачи (листинг 5).

В строке 02 определяются две переменные x и y типа student _ t. Переменная x инициализируется значением, которое будет записано в строке 08 в бинарный файл db.bin. В строке 17 из бинарного файла db.bin считывается значение, которое сохраняется в переменную y.

Под переменную x выделяется непрерывный участок памяти размером sizeof x байт[2]. Адресом этого участка является значение &x. Функция fwrite пытается записать последовательность байт, которая располагается в этом участке памяти. Данная функция возвращает число байт, которое удалось записать. По этой причине вызов функции fwrite помещен в проверочное выражение условного оператора if. Если удалось записать число байт меньшее, чем размер переменной x, то программа завершается с кодом ошибки -1 (строка 09).

Аналогично, функция fread пытается прочитать из бинарного файла последовательность байт размера sizeof y. Прочитанная последовательность записывается в участок памяти, выделенный под хранение

[1] Адрес файла в сети Интернет http://erdani.com/tdpl/hamlet.txt

[2] Переменные x и y имеют одинаковый размер, равный sizeof(student _ t) байт.

```
01.  typedef struct {
02.       char name[10];
03.       int  numb;
04.  } student_t;
```

Листинг 4. Структура данных "Студент"

```
01.  FILE     *o, *i;
02.  student_t x = {"ivanov", 1001}, y;
03.
04.  o = fopen("db.bin", "wb");
05.  if(!o)
06.      return -1;
07.
08.  if(fwrite(&x, 1, sizeof x, o) != sizeof x)
09.      return -1;
10.
11.  fclose(o);
12.
13.  i = fopen("db.bin", "rb");
14.  if(!i)
15.      return -1;
16.
17.  if(fread(&y, 1, sizeof y, i) != sizeof y)
18.      return -1;
19.
20.  fclose(i);
```

Листинг 5. Запись/чтение структуры данных (вариант 1)

переменной y. Вызов функции fread также помещен в проверочное выражение условного оператора if. Если было прочитано недостаточное число байт, то программа завершается с кодом ошибки -1 (строка 18).

Подход, представленный на листинге 5, выглядит логичным, но не является эффективным. Соответствующий бинарный файл имеет избыточный размер. Для объяснения этой ситуации требуется ответить на следующий вопрос: "Какой размер имеет значение типа student _ t?" Первый вариант ответа — 14 байт. Действительно, поле name имеет размер 10 байт, поле numb — 4 байта. В сумме это дает 14 байт. Логично предположить, что размер значения типа student _ t будет равен 14 байт. Однако правильный ответ — 16 байт.

Адрес памяти, выделяемой под хранение значения типа T, должен быть кратен размеру типа T. Поэтому значение &x.numb должно быть всегда кратно четырем. С учетом этого обстоятельства значение типа student _ t хранится в памяти следующим образом. Сначала идут

```
01.  FILE     *o, *i;
02.  student_t x = {"ivanov", 1001}, y;
03.
04.  o = fopen("db.bin", "wb");
05.  if(!o)
06.      return -1;
07.
08.  if(fwrite(&x.name, 1, sizeof x.name, o) != sizeof x.name)
09.      return -1;
10.
11.  if(fwrite(&x.numb, 1, sizeof x.numb, o) != sizeof x.numb)
12.      return -1;
13.
14.  fclose(o);
15.
16.  i = fopen("db.bin", "rb");
17.  if(!i)
18.      return -1;
19.
20.  if(fread(&y, 1, sizeof y.name, i) != sizeof y.name)
21.      return -1;
22.
23.  if(fread(&y.numb, 1, sizeof y.numb, i) != sizeof y.numb)
24.      return -1;
25.
26.  fclose(i);
```

Листинг 6. Запись/чтение структуры данных (вариант 2)

десять байт под хранение поля name. Далее идут два байта, которые выделяются, но никак не используются. Они лишь служат для "выравнивания" адреса поля num, которое следует за ними и занимает четыре байта.

На листинге 6 представлен модифицированный вариант программы, реализующий экономное размещение данных в бинарном файле.

При сохранении значения переменной x каждое поле этой переменной сохраняется отдельно. Сначала сохраняется поле name (строка 08), а затем поле numb (строка 11). Таким образом, записывается ровно 14 байт. Аналогично, при чтении сначала считываются десять байт, которые сохраняются в поле name переменной y. После этого считываются четыре байта, которые сохраняются в поле numb переменной y.

Довольно часто (особенно это касается сетевых приложений) используется следующий сценарий для чтения данных, который приводится на листинге 7.

Выделяется буфер (массив байт) достаточного размера. В рассматриваемом примере это переменная b (строка 03). Дальше осуществляется заполнение этого буфера. В строке 09 это один вызов функции fread. В сетевых приложениях буфер может заполняться в результате многократного выполнения операции чтения. После заполнения буфера различные его части при помощи стандартной функции memcpy (или ее аналога) копируются во внутренние структуры данных выполняемой программы (строки 14-15). В этом месте необходимо сделать предостережение.

Очень часто появляется соблазн преобразовать какой-то адрес памяти к указателю нужного типа, а потом выполнить операцию разыменования указателя. Например, строку 15 можно было бы переписать следующим образом

```
y.numb = *(int*)(b + sizeof y.name);
```

Использование этого выражения может привести к аварийному завершению программы, либо к снижению ее производительности. Как уже отмечалось ранее, не каждый адрес может служить указателем рассматриваемого типа данных. Для этого адрес должен быть кратен размеру этого типа данных.

Рассмотренные примеры программ позволяют продемонстрировать уровни моделирования данных (концептуальный, логический и физический), которые традиционно выделяют в теории баз данных. В этих примерах на концептуальном уровне введена сущность "Студент". При этом полагаем, что данная сущность имеет два атрибута "Имя" и "Номер зачетной книжки". Концептуальный уровень используется для моделирования предметной области.

```
01.  FILE     *i;
02.  student_t y;
03.  char      b[14];
04.
05.  i = fopen("db.bin", "rb");
06.  if(!i)
07.      return -1;
08.
09.  if(fread(b, 1, 14, i3) != 14)
10.      return -1;
11.
12.  fclose(i);
13.
14.  memcpy(&y, b, sizeof y.name);
15.  memcpy(&y.numb, b + sizeof y.name, sizeof y.numb);
```

Листинг 7. Чтение структуры данных (вариант 3)

На логическом уровне происходит формализация абстрактной сущности "Студент" в виде структурного типа данных `student_t`. Абстрактный атрибут "Имя" формализуется в виде массива символов длины 10, атрибут "Номер зачетной книжки" — 4-байтовым целым числом. Логический уровень соответствует программному манипулированию данными.

На физическом уровне сущность "Студент" представляет собой просто последовательность байт длины 14. Данный уровень используется для хранения данных на запоминающих устройствах, а также для их передачи по сети.

Заключение

Изложен подход к изучению механизмов ввода—вывода в рамках учебного курса "Работа на ЭВМ и программирование". Подход основан на формировании набора рецептов. Каждый рецепт представляет собой взаимосвязанный набор общеупотребительных сценариев, практик и примеров использования. Рецепт посвящен конкретной теме и содержит необходимые теоретические сведения, примеры программ, методические указания по изложению материала.

В работе рассмотрены три темы. Первая тема — чтение последовательности чисел из файла. В рамках изложения данной темы удалось продемонстрировать важный вывод, непосредственно касающийся программной инженерии. Программа содержит не столько реализацию целевого алгоритма, сколько обработку всевозможных нештатных ситуаций. В реальной эксплуатации программы данные нештатные ситуации могут и не возникнуть, однако их обработка должна быть предусмотрена.

Вторая тема посвящена обоснованию необходимости и принципам реализации буферизованного ввода—вывода на примере стандартной библиотеки языка Си.

Третья тема посвящена чтению и записи структурированных данных. В рамках этой темы объяснены "тонкости" работы с указателями языка Си, а также принципы выравнивания полей структур этого языка. Рассмотренные примеры позволяют познакомить учащегося с уровнями моделирования данных (концептуальным, логическим и физическим), традиционно выделяемыми в теории баз данных.

Представленный в работе материал может быть расширен за счет добавления новых рецептов. Первое направление — продолжение изучения стандартной библиотеки языка Си. Второе направление — спуск на уровень стандарта POSIX и работы с файловыми дескрипторами. Третье направление — изучение существующих промышленных решений. Например, рассмотренная тема, связанная с чтением-записью структурированных данных, может перерасти в рассмотрение таких продуктов, как Google Protocol Buffers [5] или Apache Thrift [6].

Список литературы

1. **Лав Р.** Linux. Системное программирование. СПб.: Питер, 2008. 416 с.
2. **Стивенс У. Р., Феннер Б., Рудофф Э. М**. UNIX: разработка сетевых приложений. 3-е изд.СПб.: Питер, 2007. 1039 с.
3. **Галатенко В. А.** Программирование в стандарте POSIX: Курс лекций: учеб. пособие. М.: Интернет-университет информационных технологий, 2012. 560 с.
4. **Саттер Г., Александреску А.** Стандарты программирования на C++.: Пер. с англ. М.: Вильямс, 2005. 224 с.
5. **Google** Protocol Buffers. URL: https://developers.google.com/protocol-buffers
6. **Apache** Thrift. URL:https://thrift.apache.org

A. S. Shundeev, Leading Researcher, e-mail: alex.shundeev@gmail.com Institute of Mechanics Lomonosov Moscow State University

Learning Input–Output Mechanisms in Programming Courses

The paper describes an approach to the study of mechanisms of input-output of the training course "Working on computers and programming." The approach is based on the formation of a set of recipes. Each recipe is an interrelated set of commonly used scripts, practices and examples of usage. The recipe is dedicated to a specific topic and contains the necessary theoretical information, sample programs, guidelines for the presentation of the material. The article deals with three subjects—reading a sequence of numbers from a file, buffered input and output, reading and writing structured data.

Keywords: *programming, input-output mechanisms, learning courses*

References

1. **Love R.** *Linux System Programming: Talking Directly to the Kernel and C Library*, O'Reilly Media, 2007, 392 p.
2. **Stevens W. R., Fenner B., Rudoff A. M.** *Unix Network Programming, Vol. I: The Sockets Networking API* (3rd Edition), Addison-Wesley Professional, 2003, 1024 p.
3. **Galatenko V.A.** *Programmirovanie v standarte POSIX: Kurs lekcij: Uchebnoe posobie* (Programming in the POSIX standard: Course of lectures: Tutorial), Moscow: Internet-universitet informacionnyh tehnologij, 2012, 560 p. (in Russian).
4. **Sutter H., Alexandrescu A.** *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*, Addison-Wesley Professional, 2004, 240 p.
5. **Google** Protocol Buffers, available at: https://developers.google.com/protocol-buffers
6. **Apache** Thrift, available at: https://thrift.apache.org

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

Сдано в набор 29.12.2014 г.Подписано в печать 17.02.2014 г.Формат 60×88 1/8.Заказ PI315
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**