Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет науки
и технологий имени академика М.Ф. Решетнева»

На правах рукописи

Панфилова Татьяна Александровна

СТОХАСТИЧЕСКИЕ АДАПТИВНЫЕ АЛГОРИТМЫ ПОВЫШЕНИЯ НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

05.13.01 – Системный анализ, управление и обработка информации
(космические и информационные технологии)

ДИССЕРТАЦИЯ
на соискание ученой степени кандидата технических наук

Научный руководитель
доктор технических наук, профессор
Ковалев И.В.

Красноярск – 2017

Оглавление

ВВЕДЕНИЕ

Актуальность. Конкуренция на рынке производителей программного обеспечения (ПО) сейчас высока как никогда. Сокращение сроков разработки ПО, необходимость адаптации программ под различные устройства, необходимость учета программной совместимости и повышенные требования к информационной безопасности – вот неполный перечень причин, почему вопрос о надежности программного обеспечения стоит так остро.

Важнейшим фактором, определяющим качество современного программного продукта, является надежность его функционирования. Этой проблемой занимаются исследователи и производители ПО по всему миру. Программные сбои, приводящие к остановке производств или обслуживанию клиентов, ошибки проектировщиков приводящие к утечкам клиентских данных – эти проблемы несут колоссальные финансовые и репутационные риски, как крупным, так и малым компаниям.

В то же время производители программного обеспечения вынуждены сокращать сроки разработки ПО, использовать унифицированные решения для обеспечения совместимости программных систем, привлекать к разработке удаленных специалистов. То есть несмотря на высокий запрос на надежное ПО, разработчики вынуждены экономить.

Самый эффективный способ обеспечить надежность функционирования программного обеспечения – это проведение работ на этапе проектирования ПО. Выбор надежной архитектуры программ позволит избежать большинства проблем и сократить затраты на устранение проблем надежности в будущем. Однако у современных исследователей нет единого подхода даже к определению термина «надежность программного обеспечения». Одни вкладывают в это понятие невосприимчивость к

внешним воздействиям, другие – простоту и прозрачность программных решений, исключающую возможность возникновения ошибок.

Причин возникновения ошибок и сбоев в программных системах также великое множество: проблемы, связанные с проектированием, с аппаратным обеспечением, с поведением пользователей в системе, с внешними воздействиями на систему. Отсюда и разнообразие подходов к обеспечению надежности. Это требует развития модельно-алгоритмического обеспечения, методов и средств выбора надёжного варианта ПО, что является **научной проблемой**.

Высокая конкуренция на рынке программного обеспечения, ограничения, связанные с запретом на поставку в нашу страну высокотехнологичной продукции, не оставляют сомнений в экономической целесообразности, своевременности и **актуальности** проведения исследований в данной области.

Предметом исследования являются стохастические алгоритмы моделирования и выбора надежного варианта программного обеспечения.

Объектом исследования является отказоустойчивая архитектура сложных программных систем.

Целью диссертационной работы является повышение обоснованности принятия решений при моделировании процесса функционирования программных систем, а также при выборе надежного варианта программного обеспечения.

Для достижения поставленной цели решались следующие **задачи**:

– провести анализ подходов оценки надежности программного обеспечения;

– модифицировать модель оценки надежности функционирования программной системы так, чтобы она позволяла учитывать как экспертные оценки надежности отдельных компонентов, так и оценки, полученные в результате статистического анализа экспериментальных данных;

– модифицировать модель оценки надежности функционирования программной системы на основе оценок надежности функционирования ее компонентов;

– разработать стохастические алгоритмы многокритериальной условной оптимизации, позволяющие находить надежные варианты архитектуры ПО с учетом трудозатрат на реализацию ПО и возможностью использования различных подходов к обеспечению отказоустойчивости;

– применить разработанные модели и алгоритмы при решении тестовых и реальных практических задач проектирования архитектуры ПО.

Методы исследования. При выполнении работы использовались методы прикладного системного анализа, элементы теории вероятностей, теория надёжности программного обеспечения, методология мультиверсионного проектирования отказоустойчивого программного обеспечения, эволюционные алгоритмы, методология разработки программного обеспечения, методы стохастического моделирования.

Научная новизна работы заключается в следующем:

1. Разработан стохастический алгоритм моделирования процесса функционирования программной системы для оценки надёжности архитектуры программного обеспечения, позволяющий учитывать как экспертные оценки надежности отдельных компонентов, так и статистические оценки на основе экспериментальных данных.

2. Предложена новая схема оценивания решений в многокритериальном генетическом алгоритме, отличающаяся от известных учетом одновременно всего множества критериев и позволяющая избегать преждевременной сходимости алгоритма.

3. Разработан специализированный генетический алгоритм многокритериальной оптимизации, позволяющий осуществлять поиск надежного варианта программной архитектуры путем реализации нескольких вариантов мультиверсионного подхода обеспечения избыточности программных систем.

Теоретическая значимость результатов диссертационного исследования состоит в разработке специализированных генетических алгоритмов решения задач многокритериальной оптимизации с переменной длиной хромосомы, позволяющих получать в результате своей работы репрезентативную аппроксимацию множества и фронта Парето.

Предложенный в ходе диссертационного исследования подход к оценке надежности программного обеспечения позволяет использовать неполную экспертную и статистическую информацию о функционировании программной системы.

Результаты, полученные при выполнении диссертационной работы, создают теоретическую основу для разработки моделей, методов и алгоритмов, направленных на повышение эффективности процессов разработки и модернизации программных систем.

Практическая ценность. Разработанное в результате работы над диссертацией алгоритмическое обеспечение позволяет использовать результаты тестирования программных систем для оценки надежности не только тестируемых компонентов, но и программной системы в целом. Разработанный алгоритм выбора надежного варианта программного обеспечения позволяет учесть программные сбои возникающие, как на этапе разработки программного обеспечения, так и появляющиеся в ходе его эксплуатации.

Достоверность полученных результатов подтверждается проведенными экспериментальными исследованиями программных систем с последующей независимой экспертизой результатов; согласованностью полученных расчетных оценок и экспериментальных данных о надежности программных систем; корректным применением современного математического аппарата.

Реализация результатов работы.

Алгоритмы моделирования функционирования и выбора надежной структуры программных систем реализованы в ходе выполнения работ по

проекту «Разработка протокола безопасного обмена данными в распределенной информационно-вычислительной системе на основе технологии защиты с использованием движущейся цели» (2014-2016 гг., ФЦП, соглашение № 14.574.21.0126).

Предложеный в диссертационном исследовании подход к проектированию сложных систем с помощью генетического алгоритма был реализован в виде программной системы и зарегистрирован в РОСПАТЕНТе, что делает его доступным широкому кругу специалистов по системному анализу, архитектурному проектированию и планированию разработки сложных информационных систем.

Решения, полученные в ходе выполнения диссертационного исследования, были реализованы в виде алгоритмического обеспечения, переданного в рамках реализации соглашения о предоставлении субсидии от 27.11.2014 г. (№14.574.21.0126) организации - индустриальному партнеру ООО «Гуарднет», являющемуся резидентом Инновационного центра Сколково, и представлены Министерству науки и образования РФ.

Основные положения, выносимые на защиту:

1. Стохастический алгоритм моделирования процесса функционирования программных систем для оценки надёжности архитектуры программного обеспечения позволяет учитывать оценки надежности отдельных связанных компонент программной системы для получения оценок надежности системы в целом.

2. Применение модифицированного подхода к оценке решений в генетическом алгоритме многокритериальной оптимизации, повышает эффективность его работы, а оператор процентного скрещивания расширяет возможности его применения.

3. Генетический алгоритм многокритериальной оптимизации для поиска надежного варианта программной архитектуры позволяет

осуществлять поиск эффективных по надежности программных систем, обеспечивая приемлемые затраты на разработку.

Апробация работы. Результаты проведенных исследований докладывались на научных конференциях различного уровня, в том числе:

– Международная конференция "Вычислительные и информационные технологии в науке, технике и образовании", Алматы, Казахстан 2008;

– 6-я Международная научно-практическая конференция «Ресурсы недр России: экономика и геополитика, геотехнологии и геоэкология, литосфера и геотехника», Пенза, Сентябрь 2007;

– Научно-практическая конференции «Наука – взгляд в будущее», филиал РГГУ, Красноярск,8-9 октября 2011;

– XVI Международная научная конференция «Решетневские чтения», Красноярск, 2012;

– Международная научная конференция «Молодежь. Общество. Современная наука, техника и инновации», Красноярск, 2013.

Ход выполнения диссертационной работы и диссертация в целом обсуждались на научных семинарах кафедры «Бизнес информатики» Сибирского федерального университета (2015-2017 гг.) и кафедры «Системный анализ и исследование операций» Сибирского государственного аэрокосмического университета (2010-2017 гг.).

Публикации. По теме диссертации опубликовано 9 работ, среди которых 3 статьи в журналах, входящих в перечень ВАК.

Структура и объем работы. Диссертация состоит из введения, трех глав, заключения, списка использованной литературы и приложения.

1 Моделирование оценки надежности программных систем

1.1 Понятие и определения надежности программного обеспечения

Задача оценки надежности программного обеспечения (ПО) возникла вместе с появлением первых программ для ЭВМ. На тот момент уже имели широкое распространение оценки надежности для аппаратной части информационных систем. Аппаратная и программная части информационных систем являются в равной степени необходимыми компонентами, поэтому подход к оцениванию надежности программной части первоначально мало отличался от оценивания надежности техники и заключался в переносе известных статистических методов классической теории надежности в новую область, определенную как теорию надежности ПО. Так, согласно ГОСТ 27.002-89, надежность – это свойство объекта сохранять во времени в установленных пределах значения всех параметров, характеризующих способность выполнять требуемые функции в заданных режимах и условиях применения, технического обслуживания, хранения и транспортирования [120]. В целом этот подход сохранился до сегодняшнего дня.

Однако, по мере развития вычислительной техники пришло четкое понимание того, что ПО – не просто составная часть информационных систем. Если на заре развития вычислительной техники на компьютере работала единственная программа, выпущенная в единственном экземпляре, то в настоящее время программы миллионными копиями функционируют на тысячах различных устройств. Даже если не касаться вопросов информационной безопасности, проблема обеспечения устойчивого функционирования расчетных программ, выявления их ошибок, сегодня крайне остро стоит перед разработчиками.

За прошедшие десятилетия было создано множество методов, методик и моделей исследования надежности ПО. Решению данной проблемы

посвящены исследования, таких ученых как Авижиенис [78, 79], Майерс [36], Дилон [14], Берман [86], Липаев [30, 31, 29, 32, 33, 34, 35], Луи [102, 103], Боэм [84, 85, 83], Черкесов [66], Ковалев [22, 24, 25, 26, 27, 28], Орлов [44] и других.

Единого подхода к решению этой проблемы предложено не было. Тем не менее, при разработке сложных программных систем, их создатели стараются в той или иной степени получить оценку надежности ПО.

Так выделяют несколько групп методов – подходов к оцениванию надежности: динамические методы, статистические методы, архитектурные методы, эмпирические и другие.

Динамические методы оценки надежности используют результаты выполнения программ. Данные методы используют информацию об обнаруженных в ходе выполнения программ отказах, учитывают время работы программы при каждом запуске, используют результаты тестов и траектории выполнения алгоритмов в программе. К этой группе методов относят модель Бернулли, «простую интуитивную модель», модели роста надежности, модель Шумана [41], модель Джелинского-Моранды [56], модель Шика-Волвертона [10], модель Вейбулла [39], модель Ла Падуа, модель Мусы, модель на основе неоднородного Пуассоновского распределения [29], модель Гоело-Окумото [53] и другие [36].

В целом достоверность оценок, получаемых динамическими подходами сильно зависит от качества исходных данных. При использовании прогнозных моделей обычно не учитываются влияния неформализуемых внешних факторов, имеющих место в процессе разработки и отладки ПО (особенности проекта, неравномерная плотность дефектов, квалификация персонала и др.). Также динамические модели обладают высокой трудоемкостью, используют упрощающие предположения о взаимных влияниях программных ошибок, сильно зависят от точности прогнозов от качества и объема исходной информации. К несомненным достоинствам

данной группы методов можно отнести возможность получения абсолютных значений показателей надежности.

Среди статистических методов оценки надежности ПО можно выделить модели сложности, основанные на различных метриках, таких как Метрика Холстеда, Метрика Маккейба, Метрика Джиббла; методы основанные на обнаружении дефектов, модель Миллса [7], модель Нельсона [16].

К недостаткам статистических методов относят невозможность получения абсолютных показателей надежности и высокую сложность реализации. Однако данные методы пользуются большой популярностью, благодаря возможности получать оценку в автоматическом режиме за короткое время.

Архитектурные методы выполняют декомпозицию программной системы на компоненты, выполняют оценку надежности каждого компонента в отдельности, определяют взаимное влияние компонентов и формируют общую оценку надежности всей системы. Для оценки отдельных компонентов системы могут быть задействованы как динамические, так и статистические методы. А для моделирования всей системы целиком используется представление системы в виде Марковских цепей [55], ГЕРТ-сетей [48] и сетей Петри [63].

Архитектурные методы осложнены следующими факторами: проблема выделения отдельных компонентов системы, проблема определения вероятностей переходов от одного компонента к другому, проблема учета взаимных влияний компонентов - свойства программных компонентов, в том числе и надежность, зависят от других компонентов системы. Эти зависимости могут носить сложный характер и не определяться только потоками данных и потоками управления в системе.

Наконец эмпирическая группа методов используют информацию о процессе проектирования. При получении оценок надежности в этих методах используются экспертные оценки для таких показателей как квалификация и

опыт разработчиков, их численность и время выполнения ими работ по созданию программной системы. Может учитываться организация процесса проектирования, сертификаты, опыт предыдущих проектов. Примерами эмпирических моделей могут выступать Римская модель, фазовая модель, модель фирмы IBM и модель Холстеда [6].

Классификация методов оценки надежности по их месту в процессе разработки

Классифицировать подходы оценки надежности по их месту в процессе разработки можно следующим образом [9, 11, 37 38, 40, 61, 62, 87]:

- подходы, использующиеся на стадии анализа;
- подходы, использующиеся на стадии дизайна архитектуры;
- подходы, использующиеся на стадии создания программного кода;
- подходы, использующиеся на стадии тестирования;
- подходы, использующиеся на стадии опытной эксплуатации;
- подходы, использующиеся на стадии промышленной эксплуатации.

Модели оптимизации сроков и стоимости проекта – такие модели используются для минимизации сроков разработки и стоимости проекта при сохранении должного уровня надежности ПО. Обычно, используются на ранних стадиях разработки ПО [95, 98, 113].

Один из подходов заключается в последовательном оценивании надежности программ на каждом этапе разработки.

Классификация по оптимизируемым параметрам надежности

Среди работ из области оценки параметров надежности программного обеспечения существует большое количество различных подходов к измерению количественных показателей, характеризующих надежность[110].

Надежность – имеет обозначение *R (reliability)* измеряется как вероятность не возникновения сбоя. Надежность используется практически во всех моделях как основной показатель.

Среднее время появления сбоя – *MTTF* (*Mean Time To Failure*); измеряет время между двумя последовательными сбоями [72, 73, 74].

Интенсивность сбоев – величина обратная *MTTF*, определяет количество сбоев в единицу времени.

Среднее время простоя (*TR*) – величина, определяющая время, затрачиваемое на выявление, устранение и восстановление системы или компонента системы после сбоя [75, 76].

Коэффициент готовности системы – *S*, определяется как отношение разности среднего времени появления сбоя и среднего времени простоя системы к среднему времени появления сбоя.

Плотность ошибок в коде - измеряется как общее число ошибок на тысячу строк кода [97].

Количество оставшихся ошибок в коде ПО – используется при разработке нового программного обеспечения. Показывает количество ошибок в коде на каждую тысячу строк программного кода.

Одним из важных параметров при оценке надежности программных систем является время. Время можно разделить на два типа: календарное время и процессорное время. Процессорное время преобразуется в календарное для сравнения параметров ПО, используемого на разном аппаратном обеспечении и разных операционных системах.

При комбинировании различных моделей оценки надежности следует учитывать тот факт, что понятия времени в них могут различаться. Такие модели надо унифицировать, приводя их единому пониманию времени, вводя соответствующие формулы с коэффициентами пересчета времени.

1.2 Причины программных сбоев и методы обеспечения отказоустойчивости

Что бы обеспечить отказоустойчивую работу информационной системы (ИС) в целом, необходимо обеспечить достаточный уровень

отказоустойчивости составляющих ее программных и аппаратных компонентов и узлов. Таким образом, необходимо обеспечить:

– надёжность программного обеспечения;

– надёжность аппаратного обеспечения;

– бесперебойную подачу электропитания и соответствие её предельно допустимым нормам, в том числе обеспечение работы систем охлаждения;

– отказоустойчивые и помехозащищенные каналы передачи данных;

– необходимый уровень подготовки персонала ИС;

– регламентное техническое обслуживание ИС;

– оперативную, непрерывную и квалифицированную техническую поддержку ИС.

Обеспечение надёжности программного обеспечения информационной системы является более сложной задачей, чем обеспечение надёжности аппаратного обеспечения так как [17, 69]:

– элементы ПО не стареют при длительной эксплуатации, однако возникают конфликты, связанные с интеграцией различных видов ПО в информационной системе, например, при переходе на новую операционную систему;

– число вариантов использования ПО, как правило, значительно больше числа вариантов использования аппаратуры;

– в ПО гораздо проще вносить исправления и дополнения, чем в аппаратуру, но в тоже время делать это приходится гораздо чаще, а при внесении исправлений могут быть внесены дополнительные ошибки.

Программное обеспечение содержит ошибку, если [101]:

1. Поведение ПО не соответствует спецификациям.

Обычно предполагается, что спецификации корректны. Подготовка спецификаций - один из основных источников ошибок. Если поведение программного продукта не соответствует его спецификациям, ошибка,

вероятно, имеется. Однако, даже если система ведёт себя в соответствии со спецификациями, мы не можем утверждать, что она не содержит ошибок.

2. Поведение ПО не соответствует спецификациям при использовании в установленных при разработке пределах. Если система случайно используется в непредусмотренной ситуации, её поведение должно оставаться разумным. Если это не так, она содержит ошибку.

3. Программное обеспечение ведёт себя не в соответствии с официальной документацией и поставленными пользователю спецификациями. Ошибки могут содержаться как в программе, так и в спецификациях, или в руководстве описана только ожидаемая и планируемая работа с системой.

4. Система не способна действовать в соответствии с исходной спецификацией и перечнем требований пользователя.

Это утверждение тоже не лишено недостатков, поскольку письменные требования пользователя редко детализированы настолько, чтобы описывать желаемое поведение программного обеспечения при всех мыслимых обстоятельствах.

Окончательное определение: в программном обеспечении имеется ошибка, если оно не выполняет того, что пользователю разумно от него ожидать. Отказ программного обеспечения - это проявление ошибки в нём.

Термины «ошибка», «сбой» и «дефект» часто используются без разделения по смыслу. В ПО «ошибка» – это действия программиста, которые приводят к дефектам в ПО. «Дефект» в ПО влечет за собой сбои во время исполнения кода [102]. «Сбой» – отклонение выхода системы от желаемого при выполнении кода.

Дефект влечет за собой сбой только тогда, когда выполняется код содержащий ошибку, и распространяется до выхода системы. Уровень тестируемости дефекта определяется как вероятность обнаружения сбоя на случайно выбранном выходе.

Уровни сбоя: катастрофичный, высокий, средний, низкий, незначительный. Определения этих уровней меняются от системы к системе.

Простой или «зависание» - особый вид сбоя, зависящий как от сбоя в аппаратной части, так и в программной части системы или же от некорректных действий пользователя.

Методы обеспечения отказоустойчивости программного обеспечения

Методы обеспечения устойчивости к ошибкам направлены на минимизацию ущерба, вызванного появлением ошибок, и включают в себя:

- обработку сбоев аппаратуры;
- повторное выполнение операций;
- динамическое изменение конфигурации;
- сокращенное обслуживание в случае отказа отдельных функций системы;
- копирование и восстановление данных;
- изоляцию ошибок.

На практике существует два дополняющих друг друга подхода, используемых при разработке отказоустойчивого программного обеспечения информационно-управляющих систем [5]:

1. Предотвращение сбоев. В процессе проектирования и реализации программных систем используются такие технологии разработки ПО, которые сводят к минимуму вероятность ошибки оператора и позволяют найти системные ошибки до того, как система будет запущена в промышленную эксплуатацию. Предотвращение сбоев, фактически, означает, что заказчику будет поставлена программная система, в которой вероятность возникновения сбоя сведена к нулю. Это достигается с помощью статических и динамических методов тестирования, которые позволяют обнаружить ошибки и исправить их до начала эксплуатации системы.

Однако, при уменьшении количества ошибок в программе стоимость обнаружения новых ошибок возрастает экспоненциально. Это значит, что обеспечить необходимый уровень надёжности достаточно сложной программной системы только за счет тестирования практически невозможно.

2. Устойчивость к сбоям. Программная система проектируется таким образом, чтобы дать возможность обнаружить и исправить ошибки, обрабатывая некорректное поведение системы до того, как это приведет к её отказу. Устойчивость к сбоям подразумевает наличие в системе возможности обработки ошибок отдельных модулей в процессе их исполнения.

Существует несколько подходов, которые позволяют обеспечить функционирование программной системы при наличии в ней ошибок [4, 36]:

1. Методы отступления, которые применяются, когда системе важно корректно закончить работу. Например, если сбой возникает в системе, управляющей технологическими процессами [60, 68], и в результате эта система выходит из строя, то может быть выполнен специальный фрагмент программы, обеспечивающий безаварийное завершение всех управляемых системой процессов.

2. Методы изоляции ошибок, основное назначение которых предотвратить распространение последствий ошибок на другие компоненты системы. Таким образом, в системе сбои могут возникать, но они изолируются. При этом происходит отключение отдельных функций, а оставшаяся часть системы остается работоспособной.

3. Применение отказоустойчивых программных архитектур. Такие архитектуры предполагают использование избыточного количества версий компонентов программного обеспечения и применение блока проверки допустимости выполнения версий компонента или алгоритма сравнения результатов выполнения.

При проектировании программных систем с высокими требованиями к готовности, в том числе систем реального времени, которые должны выполнить свои функции в полном объеме и ни в коем случае не

останавливаться, методы отступления и изоляции ошибок не применимы. Для достижения устойчивости к сбоям таких систем наиболее эффективным является применение отказоустойчивых программных архитектур.

Мультиверсионное проектирование программного обеспечения

Одной из наиболее перспективных и положительно зарекомендовавших себя методологий обеспечения высокой надёжности программного обеспечения является введение программной избыточности. Но простое дублирование компонентов, как при аппаратном резервировании, недопустимо, так как в отличие от аппаратуры, программные ошибки имеют внутреннюю природу и при дублировании не исчезнут. При введении программной избыточности возникновение сбоя в функционально эквивалентных модулях (версиях) на одних и тех же входных данных происходит в различных точках его исполнения [43].

Создание функционально-эквивалентных, но, тем не менее, разных модулей может быть достигнуто при разработке версий одного модуля разными специалистами или применением разных языков программирования. Программную избыточность используют для разработки компонентов, к которым происходит наиболее частое обращение, или результаты работы которого участвуют в критически важных циклах управления [13].

Разнообразие может быть в следующем:

– квалификация и опыт разработчиков;

– алгоритмы и структуры данных;

– языки программирования;

– инструментальные средства программирования и среды проектирования (включая компиляторы);

– методы тестирования.

Существует два основных подхода к реализации мультиверсионности программного модуля. Первый подход – мультиверсионное программирование (*N-version programming, NVP*) [57, 78, 79, 102, 103]. При этом подходе версии выполняются параллельно (рисунок 1.1). Результат их работы определяется с помощью какого-либо алгоритма голосования.

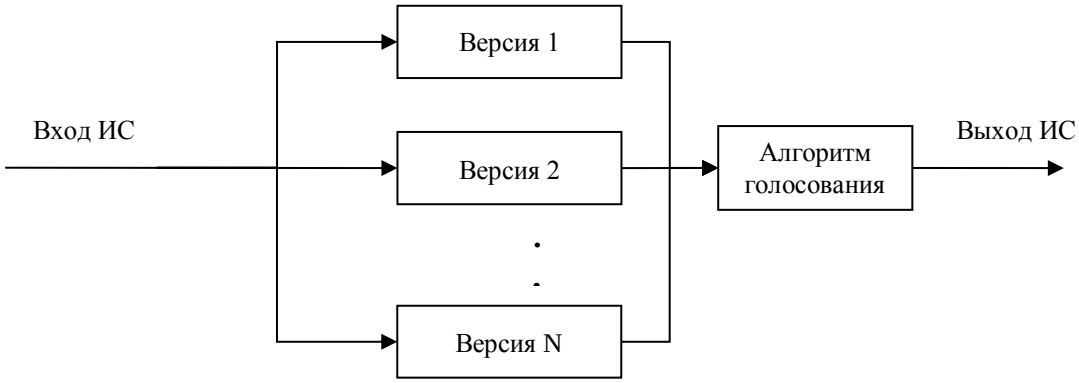


Рисунок 1.1 – Схема работы при методе *NVP*

Алгоритмы голосования могут быть различными в зависимости от задачи. Обычно используется выбор результата по абсолютному большинству (если эквивалентных выходов больше половины) или по согласованному большинству (если есть самая большая группа эквивалентных выходов) [42]. При этом выходные значения являются идентичными при заданной погрешности.

Второй подход - блок восстановления (*Recovery Block*) [57, 77, 102, 103, 104, 106, 107, 108, 109, 112, 117]. При этом подходе каждый критичный программный компонент содержит множество версий вычислительного модуля. Также содержит приемочный тест, который проверяет работу версий, и подпрограмму, которая по результатам выполнения теста либо принимает результаты вычисления, либо запускает их повторно, но уже с помощью другой версии вычислительного модуля (рисунок 1.2).

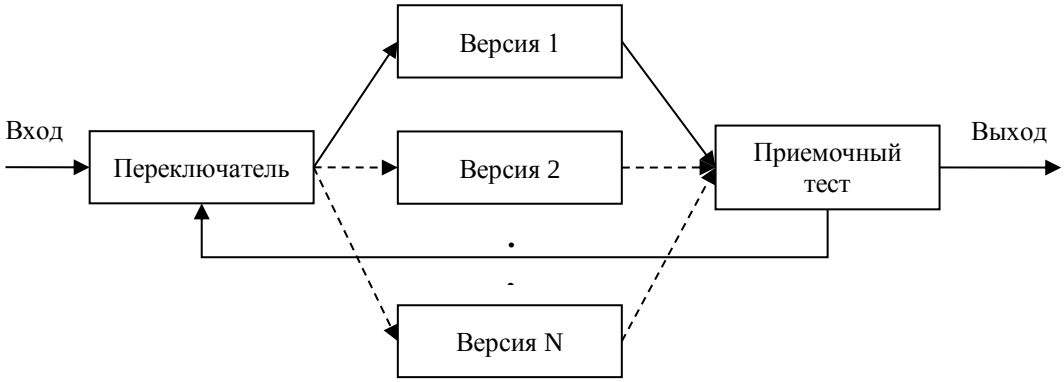


Рисунок 1.2 – Схема работы при методе *RB*

Также применяются гибридные методы введения программной избыточности [103, 117]:

1. Согласованный блок восстановления (*CRB – Consensus Recovery Block*). Если происходит сбой в алгоритме голосования, или алгоритм голосования не может выбрать результат, то модуль продолжает работу по схеме блока восстановления.

2. *NSCP* (*N-self checking programming*) или *NVP* с самоконтролем. Приемочный тест играет роль фильтра. Также используется соответствующий алгоритм голосования, учитывающий, что в некоторых версиях результат может быть не приемлемым.

1.3 Проектирование надежной архитектуры ПО

Как было указано выше, одним из самых распространенных способов повышения надежности программного обеспечения – это обеспечение отказоустойчивости и невосприимчивости к сбоям на этапе проектирования информационной системы. Несмотря на различающиеся подходы к процессу проектирования и разработки ПО практически все исследователи указывают следующие обязательные этапы [18]:

1) анализ требований, предъявляемых к информационной системе;

2) определение спецификаций;

3) проектирование архитектуры;

4) кодирование;

5) тестирование:

6) эксплуатация и сопровождение.

Именно на этапе проектирования архитектуры у разработчика есть возможность предупредить большую часть ошибок, возникающих в ИС. Строится структура взаимосвязей модулей таким образом, что каждый модуль четко разделен, разрабатывается и тестируется независимо. Структура может пересматриваться для выявления ошибок. Проектирование архитектуры — это процесс разбиения большой системы на более мелкие части. Для обозначения этих частей придумано множество названий: программы, компоненты, подсистемы и уровни абстракции. Процесс разработки архитектуры — необходимый шаг в процессе разработки ПО.

Надежность архитектуры включает в себя надежность индивидуальных элементов. Сбой отдельного элемента приводит к неработоспособности этого и, возможно, других элементов, но не для всей системы в целом. Сбой в системе или ее функциях может выразиться в периоде простоя системы. Период простоя системы определятся, как отрезок времени, на котором система или ее часть не может выполнять функции. Период простоя системы приводит к резкому снижению производительности, поэтому уменьшение времени простоя системы является одним из наиболее важных факторов при разработке архитектур распределенного программного обеспечения.

Для достижения низких показателей сбоев для компонентов аппаратного обеспечения можно увеличить надежность дублированием наиболее критических компонентов архитектуры. В некоторых случаях, при отказе компонента, его функции могут выполняться дублирующим компонентом. Надежность программного обеспечения нельзя увеличить таким методом, поскольку дублирование компонентов ПО приведет также к

дублированию его ошибок. Существуют методы, использующие разнообразие в спецификации, разработке, внедрении и тестировании.

Эксперименты показывают, что применение избыточности на самых ранних этапах разработки, насколько это возможно, уменьшает вероятность появления ошибок [18]. Надежное ПО можно создать с помощью тщательной разработки архитектуры и выявления ошибок в компонентах, которые больше всего оказывают влияние на надежность системы. Эти компоненты определяются как наиболее часто используемые или архитектурно связанные с множеством других компонентов, влияя, таким образом, на их надежность.

Модульная организация предполагает деление ПО на функционально завершенные части (модули), унификацию связей между ними и установление иерархии взаимодействия компонентов, которая определяется последовательностью их вызова. Вызов осуществляется передачей управления от вызывающего модуля к вызываемому, который по окончании выполнения возвращает управление вызывающему модулю. Вызов группы программ осуществляется передачей управления диспетчеру группы программ более низкого уровня иерархии, входящей в данный КП.

Основные методы современной практической разработки программных комплексов базируются на функциональной декомпозиции с использованием модульно-иерархических принципов [1, 2, 74, 75]. При этом на каждом иерархическом уровне ограничивается сложность компонентов и их связей. В результате общая сложность системы растет значительно медленнее с возрастанием объема задач, чем при неструктурированном проектировании. Эти принципы привели к созданию структурного программирования, как стандартного способа построения модульных программ.

Определение модульной программы опирается на ограничения по размерам программ и на понятие их независимости, т.е. модульная программа должна состоять из модулей конечных размеров, которые имеют точку входа и точку выхода. Преимущества модульности состоят в упрощении проектирования и модификации программы, облегчении

тестирования и отладки программ, возможности создания библиотеки стандартных модулей и т. д. Недостатки — увеличение времени исполнения программ и объема памяти, усложнение межмодульного взаимодействия — в целом окупаются выигрышем, получаемым от сокращения срока разработки и упрощения сопровождения программ.

Основным средством реализации модульности программ является структурное программирование. Целями структурного программирования служат повышение читабельности и ясности программ, увеличение производительности работы программистов и упрощение процесса разработки. Само понятие структурного программирования представляет собой некоторые принципы написания программ в соответствии с совокупностью определенных правил. Согласно «структурной теореме» для построения любой программы необходимы три основные базовые конструкции [45]:

- простая вычислительная последовательность, означающая, что два действия должны выполняться одно за другим;
- альтернатива или ветвление, при котором на основе проверки некоторого условия делается выбор между двумя возможными путями;
- цикл или итерация, обеспечивающая повторное выполнение некоторой последовательности действий.

Модульность построения приводит к использованию иерархической структуры взаимодействия модулей программы. Иерархическая схема, отражая функции модулей, одновременно показывает структуру связей между ними.

Архитектура программного обеспечения формируется из ряда компонентов, соединенных различными средствами зависимости и связи. Архитектурный компонент может быть определен по-разному в зависимости от архитектурного подхода и степени подробности описания архитектуры.

Наиболее критическими компонентами архитектуры программного обеспечения являются компоненты, к которым происходят частые обращения, или компоненты архитектурно связанные (через зависимости и связи) со множеством других компонентов, влияя таким образом на их надежность. Процесс - компонент, который занимает слот в таблице процессов процессора. Модуль состоит из нескольких файлов, логически связанных и образующих часть функционального кода.

Зависимости компонентов позволяют неисправности распространяться из компонента, в котором она происходит, к другим компонентам. Это распространение может вызывать сбои в цепочке (или в дереве) компонентов. Обнаружение отказов зависит от выполненных тестов или от количества и типа запросов пользователя. Ошибка может произойти в любом компоненте. Эта ошибка может быть вызвана сбоем, переданной другим компонентом, или это может быть сбой, произошедший именно в этом компоненте. Ошибка может быть прослежена через цепочку (или дерево) зависимости компонентов для устранения всех сбоев, которые связаны с этой ошибкой [71].

Надежность ПО зависит от уровня, соответствующего различным компонентам и их зависимостям. В зависимости от того, где произошел сбой, длительность отказа системы и его влияние на надежность системы различны. Сбой может происходить на различных уровнях архитектуры, в модуле, процессе, интерфейсе компонента или в связи и механизме контроля. Число архитектурных уровней в модели архитектуры ПО зависит от частного проекта системы.

Коэффициент надежности архитектуры ПО можно оценить по формуле [49]:

$$R = \sum_{j=1}^{M} \sum_{i=1}^{Nj} PU_{ij} \times R_{ij} \qquad (1.1)$$

где M - число уровней архитектуры ПО;

N_j - число компонент на уровне j, $j=1,..,M$;

PU_{ij} – вероятность того, что компонент i на уровне j, $i \in \{1,..,N_j\}$, $j \in \{1,..,M\}$ будет использоваться;

R_{ij} – надежность компонента i уровня j.

Оценить данные параметры на фазе разработки архитектуры невозможно, численные показатели в модель берутся непосредственно на фазах кодирования и тестирования.

Кроме того, на надежность влияют ошибки, допущенные на фазе кодирования ПО. Количество ошибок, в свою очередь, зависит от квалификации и опыта разработчиков, а также от способа тестирования ПО. Одним из самых часто используемых параметров оценки корректности ПО является плотность ошибок.

Начальную плотность ошибок можно оценить как:

$$D = C \times F_{pr} \times F_{pt} \times F_m \times F_s \qquad (1.2)$$

где *Fpt* – коэффициент фазы тестирования;

Fpr – коэффициент командного программирования;

Fm – коэффициент опытности и «зрелости» процесса разработки ПО;

Fs – коэффициент структурирования;

C – константа, определяющая количество ошибок/*KLOC* (ошибок на тысячу строк исходного кода).

Коэффициенты *Fpr, Fm, Fs и C* зависят только от мастерства и опыта команды разработчиков.

Коэффициент командного программирования (*Fpr*): плотность ошибок зависит от конкретных людей, их опыта написания программ и отладки.

Можно принять следующие значения параметра: «Высокий», «Средний», «Низкий». Числовые показатели определяются и задаются экспертом.

Коэффициент опытности и «зрелости» процесса разработки ПО (*Fm*).

Можно принять следующие значения параметра: «Уровень 1», «Уровень 2», «Уровень 3», «Уровень 4», «Уровень 5». Числовые показатели определяются и задаются экспертом.

Коэффициент структурирования (*Fs*): Этот параметр берет во внимание зависимость плотности ошибок от языка программирования (отношение количества кода на ассемблере и языка высокого уровня)

$$Fs=1+0.4a \ ,$$

где *a* - отношение количества кода на ассемблере и языка высокого уровня. Предполагается, что код на ассемблере может содержать на 40% ошибок больше.

На фазе тестирования возможно использование различных методик, каждая из которых обладает собственными свойствами, временными характеристиками и различаются по стоимости. Фаза тестирования самая продолжительная и может занять до 60% от всего проекта. Во время тестирования выявляется самое большое количество ошибок в ПО.

Рассмотрим меры покрытия теста, используемые на данной фазе:

- покрытие выражений – доля всех условных выражений, выполненных во время теста;

- покрытие ветвлений – доля всех ветвлений, выполненных во время теста;

- покрытие предикатов – доля всех переменных, которые используются для проверки условий перед условным переходом.

В начальной плотности ошибок (1.2) коэффициент фазы тестирования (*Fph*) принимает следующие значения: «Тестирование модуля», «Подсистемы», «Системы», «Приемлемость». Числовые показатели определяются и задаются экспертом.

Параметры для оценки *D* по выражению (1.2) должны быть скорректированы с использованием данных, накопленных в результате

деятельности разработчиков ПО. Коэффициент *C* обычно лежит в диапазоне от 6 до 20 ошибок/*KLOC*. Можно брать как средние значения, так и максимальные и минимальные значения для оценки диапазона плотности ошибок.

Кроме того, на стоимость, надежность и время разработки ПО часто накладываются жесткие ограничения. И наконец, разрабатываемое ПО, помимо удовлетворению требованиям надежности и стоимости, должно справляться с задачами, для которых оно создается.

Формализация задачи разработки надежного ПО должна привести к оптимизационным постановкам. При этом очевидны две группы критериев:

- критерии надежности, которые должны быть максимизированы,
- критерии стоимости, которые должны быть минимизированы.

1.4 Количественный метод оценки надежности отдельных программных модулей

Главным недостатком моделей надежности (1.1)-(1.2) является субъективность присутствующих в формулах показателей. Действительно, надежность *i*-го компонента ПО (R_i), надежность связей компонентов типа $j=1...M$ (L_j), коэффициент опытности и «зрелости» процесса разработки ПО (F_m) и др. обычно рассчитываются с помощью экспертных оценок. Их расчет занимает значительное время у экспертов, зависит от количества компонентов информационной системы и не может производится в автоматическом режиме. Примером методологии подобных расчетов может служить ГОСТ 28195-99 «ОЦЕНКА КАЧЕСТВА ПРОГРАММНЫХ СРЕДСТВ» [121]. В ГОСТе приводится методология проведения экспертной оценки по шести показателям (Надежность, Сопровождаемость, Удобство применения, Эффективность, Универсальность, Функциональность) для шести различных фаз жизненного цикла ПО (Фаза анализа, Фаза

проектирования, Фаза реализации, Фаза тестирования, Фаза изготовления, Фаза обслуживания). Общее количество оцениваемых по ГОСТу элементов может составлять до нескольких сотен. Причем согласно тому же ГОСТу оценки следует производить для каждого программного модуля, входящего в состав программного комплекса. Таким образом независимая количественная оценка надежностных характеристик программного обеспечения является важной задачей.

В [21] излагается подход к проектированию автоматизированных систем управления (АСУ) с помощью ГЕРТ-сетей (*GERT – Graphical Evaluation and Review Techniqu* (Метод графической оценки и анализа)). Процесс функционирования информационной системы представлен графом состояний, в которых может оказаться система. Каждое состояние рассматривается как случайная функция с известным распределением. Переход из одного состояния в другое рассматривается как случайный процесс, зависящий от результатов выполнения каждого состояния.

На рисунке 1.3 представлен пример фрагмента работы программы для ЭВМ. Состояние «1» - соответствует работе модуля «ввода данных». Состояние «2» - это следующий по очередности модуль информационной системы, переход к нему возможен с вероятностью p_1. Состояние «3» - соответствует системному сбою, который приводит к остановке работы всей программы. Состояние «4» - описывает работы модуля по корректировке вводимых данных, после которого возможен переход к состоянию «2», «3» или к состоянию «5», соответствующему вызову модуля по «ручной корректировке вводимых данных». При этом $p_1 + p_2 + p_3 = 1$ и $p_4 + p_5 + p_6 = 1$. Состояния «1» и «4» – носят вероятностный, а состояния «2», «3», «5» – дискретный характер.

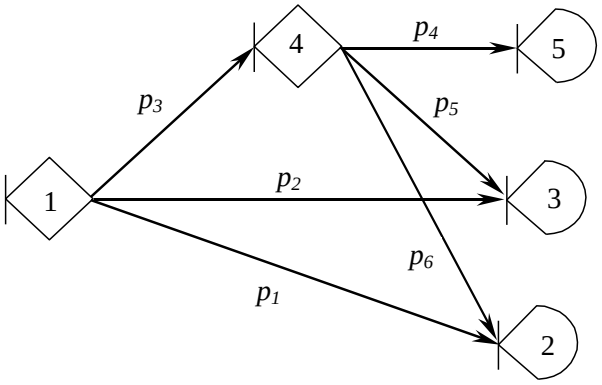


Рисунок 1.3 – Пример GERT-сети.

Моделирование процесса функционирования программного обеспечения ГЕРТ-сетью позволяют ответить на следующие вопросы:

1. Какова вероятность того, что работа программы приведет к системному сбою?

2. Какова вероятность того, что программа продолжит штатную работу?

3. Какова вероятность того, что программа продолжит работу в ручном режиме?

4. Чему равны математическое ожидание и дисперсия времени, необходимого для продолжения штатной работы?

5. Сколько времени будет потеряно, если после исправления программа все равно придет к системному сбою?

В [48] предложены прямой и обратный алгоритмы расчета вероятностно-временных характеристик для ГЕРТ-сетей предназначенных для моделирования надежностных характеристик программного обеспечения.

Один из подходов к организации процесса интенсивной обработки информации – это использование в качестве узлов обработки информации компьютеров пользователей в свободное от использования времени. Такие проекты, как *SETI, Climate Predictions*, *Legion* или *Condor*, получают все большее распространение и весьма эффективны для некоторых задач [23; 64]. В случае, когда для некоторого узла известна вероятность бесперебойной

работы в течение времени, необходимого для расчета задачи, то рассчитать функцию распределения времени выполнения задачи можно, используя ГЕРТ-сети [50].

В процессе данного диссертационного исследования стала очевидной возможность модификации математической модели ГЕРТ-сетей. Полученная модифицированная модель позволяет оценить время одновременного выполнения задачи на нескольких узлах (рабочих станциях) гетерогенной распределенной системы обработки информации.

Математическая модель модифицированных ГЕРТ-сетей. Для того чтобы дать определение математической модели модифицированной ГЕРТ-сети, введем следующие обозначения [65]:

p_i – вероятность активации узла i;

p_{ij} – функция условной вероятности активации узла j, при условии активации узла i, $p_{ij} = 1$, если узел i имеет детерминированный выход или $<i, j>$ – единственная дуга стохастического выхода;

$p_{ijн}$ – вероятность выполнения начала дуги $<i, j>$ при условии, что узел, из которого она входит, активирован;

$p_{ijк}$ – вероятность выполнения конца дуги $<i, j>$ при условии, что узел, в который она входит, активирован;

$F_i(t)$ – функция распределения времени выполнения узла i в момент его активации;

$F_ij(t)$ – функция условной вероятности распределения времени выполнения работы $<i, j>$;

$F_{ijн}(t)$ – функция распределения времени выполнения t_i в начале дуги $<i, j>$ при условии, что узел, из которого она входит, активирован;

$F_{ijк}(t)$ – функция распределения времени выполнения t_j на конце дуги $<i, j>$ при условии, что узел, в который она входит, активирован.

Также для модифицированных ГЕРТ-сетей (МГ сетей) справедливы следующие ограничения [51; 65]:

Ограничение 1’ (О1’): в течение каждого выполнения проекта для каждого стока активируется не более одного источника, из которого данный сток достижим.

Ограничение 2’ (О2’): для каждого узла *i* МГ-сети, если узел *i* активирован, то параметры всех выходящих из него дуг вычислимы.

Ограничение 3’ (О3’): для каждого узла *k* произвольной циклической структуры С существует путь из k к узлу вне *C* такой, что pij > 0 для каждой дуги <*i, j*> данного пути. Иными словами, есть из каждого цикла есть выход с положительной вероятностью.

Ограничение 4’ (О4’): для всякого узла *i* ГЕРТ-сети *G*, имеющего *IOR*- или *AND*-вход, для любых $j, k \in P(i)$, $Pr(i, j, k) = \{l\}$, причем l – единственный узел и l имеет детерминированный выход.

Ограничение 5’ (О5’): для всякого узла *i* ГЕРТ-сети *G*, имеющего детерминированный вход, для любых $j, k \in S(i)$, $Sc(i, j, k) = \{l\}$, причем *l* – единственный узел и *l* имеет *AND*- или *IOR*-вход.

Ограничение 6’ (О6’): реализация сети является допустимой, если в процессе выполнения узел i активируется с вероятностью, большей $maxP_i > 0$ и не более, чем $maxA_i \geq 1$ раз; узел *i* имеет *EOR*-вход, если в узел *i* циклической структуры *C* входит более одной дуги и хотя бы одна дуга не принадлежит циклу *C*; узел *i* имеет стохастический выход, если из узла *i* циклической структуры *C* выходит более одной дуги и хотя бы одна дуга не принадлежит циклу *C*; узел *j* с детерминированной выходной функцией, являющийся стохастическим началом узла *i*, принадлежит циклу, если узел *i* с *IOR*- или *AND*-входом принадлежит данному циклу.

Будем называть ГЕРТ-сеть *G* (*V* – множество вершин, *E* – множество направленных ребер) МГ-сетью, если:

1) *G* имеет единственный источник и, по крайней мере, один сток;

2) сеть *G* удовлетворяет ограничениям О1’–О4’;

3) задано множество параметров для каждого узла сети (по крайней мере, вероятность активации);

4) для каждой дуги указаны функции преобразования параметров узлов;

5) источник активируется в момент времени 0 (если параметр, отвечающий за время, определен).

Математическая модель модифицированных ГЕРТ-сетей изначально создавалась таким образом, чтобы всякая МГ-сеть, удовлетворяющая О1'–О4' и для которой заданы дополнительные ограничения О6', была бы вычислима при помощи некоторого алгоритма. По своей сути О1'–О4' гарантируют однозначность построения и расчета графа реализации, а О6' – конечность количества этих реализаций.

Рассмотрим МГ-сеть с вектором веса дуг [p_{ij}, F_{ij}] [47], тогда $p_{ijк}$ и $F_{ijк}$ (t) для дуги <i, j> имеют вид

$$p_{ijк} = p_{ijн} \cdot p_{ij};$$

$$F_{ijk}(t) = \int_{-\infty}^{+\infty} F_{ijH}(S) \cdot F'_{ij}(t-s)ds = F_{ijH}(t) + F_{ij} \qquad (1.3)$$

Если считать, что случайная величина $t > 0$, то пределы интегрирования можно заменить на 0 и t соответственно. Для *EOR*-входа p_j и Fj (t) имеют вид

$$p_j = p_{ijк};\ F_j(t) = F_{ijк}(t), \qquad (1.4)$$

где j – вычисляемый *EOR*-вход, имеющий начало в узле i.

Для *AND*-входа p_j и F_j (t) имеют вид

$$p_j = p_{l1,m1k} \cdot p_{l2,m2k} \cdot \ldots \cdot p_{lN,mNk}/p_i^{N-1},$$

$$F_j(t) = \max\left(t_{l1,m1k} \cdot t_{l2,m2k} \ldots t_{lN,mNk}\right) =$$

$$F_{l1,m1k}(t) \cdot F_{l2,m2k}(t) \ldots F_{l2,mNk}(t)$$

(1.5)

где j – вычисляемый *AND*-вход, имеющий стохастическое начало в узле i; $\{l_1 - m_1\}$, $\{l_2 - m_2\}$, …, $\{l_N - m_N\}$ – N непересекающихся подсетей.

Для *IOR*-входа p_j и $F_j(t)$ имеют вид

$$p_j = p_i \cdot [1 - (1 - p_{l1,m1k}) \cdot (1 - p_{l2,m2k}) \cdot \ldots \cdot (1 - p_{lN,mNk})],$$

$$F_j(t) = \min\left(t_{l1,m1k} \cdot t_{l2,m2k} \ldots t_{lN,mNk}\right) =$$

$$= 1 - (1 - F_{l1,m1k}(t)) \cdot (1 - F_{l2,m2k}(t)) \cdot \ldots \cdot (1 - F_{lN,mNk}(t)), \qquad (1.6)$$

где j – вычисляемый IOR-вход, имеющий стохастическое начало в узле i.

Формулы (1.3)–(1.6) позволяют рассчитать любой граф реализации, удовлетворяющий ограничениям O1’–O4’.

Результатом расчета сети будет множество реализаций W_r, сгруппированное по стокам i_r. Для стока в каждой реализации будет рассчитана вероятность наступления события (вероятность возникновения реализации), функция распределения времени выполнения данной реализации и т. д. Пусть *r* – элемент множества реализаций со стоком *i.*

Вероятность активации стока *i:*

$$P_j = \sum_r p_i^r \qquad (1.7)$$

Вероятность активации стока *i* ко времени *t* в реализации *r:*

$$P_{j,r}(t) = p_i^r \cdot F_i^r(t) \qquad (1.8)$$

Вероятность активации стока *i* ко времени *t:*

$$P_j(t) = \sum_r p_i^r \cdot F_i^r(t) \qquad (1.9)$$

Математическое ожидание времени активации стока *i:*

$$E(t_i) = \frac{\sum_r [p_i^r \cdot \int_{-\infty}^{\infty} t \cdot dF_i^r(t)]}{\sum_i p_i^r} \qquad (1.10)$$

Математическое ожидание времени выполнения всей сети:

$$E(t) = \frac{\sum_i \sum_r [p_i^r \cdot \int_{-\infty}^{\infty} t \cdot dF_i^r(t)]}{\sum_i p_i^r} \qquad (1.11)$$

Дисперсия времени активации стока *i:*

$$D(t_i) = \frac{\sum_r [p_i^r \cdot (\int_{-\infty}^{\infty} (t - E(t_i))^2 \cdot dF_i^r(t)]}{\sum_i p_i^r} \qquad (1.12)$$

Дисперсия времени выполнения всей сети:

$$D(t) = \frac{\sum_i \sum_r [p_i^r \cdot (\int_{-\infty}^{\infty} (t - E(t_i))^2 \cdot dF_i^r(t)]}{\sum_i p_i^r} \qquad (1.13)$$

Для записи узлов, дуг и относящихся к ним величин и функций используем обозначения, принятые для структур вида «[структура].[параметр]».

Пусть *Prs* и *DFs* – множества вещественных и стохастических параметров соответственно. Каждый узел графа реализации будет обладать

данными параметрами. Параметр *p* (вероятность активации узла) является одним из элементов множества *Prs (p ∈ Prs).*

Для узла *v ∈ V* графа *МГ*-сети *G = <E, V>:v.a* – количество активаций узла *v* в процессе построения реализации;

v.In – тип входной функции узла *(EOR, IOR* или *AND);*

v.Out – тип выходной функции узла *(STOCH* или *DET);*

v.InF(F) – тип входной функции узла для стохастических переменных с функцией распределения *F(AND, IOR, MIN, MAX, EQUAL) (*только для узлов с *IOR*- или *AND*-входом*);*

v.P – множество узлов, являющихся предками узла *v;*

v.S – множество узлов, являющихся потомками узла *v;*

v.MaxA – максимально допустимое количество активаций узла *v* в графе реализации;

v.MaxP – минимально допустимая вероятность события «узел *v* активирован»;

v.dR – узел, являющийся детерминированным источником узла *v;*

v.dS – узел, являющийся детерминированным стоком узла *v* (только для прямого алгоритма расчета МГ-сети).

Для дуги *e ∈ E* графа МГ-сети *G = <E, V>(e = <v1, v2>):*

e.v1 – узел-начало дуги*;*

e.v2 – узел-конец дуги;

e.F_Pr(Pr_i*)* – функция преобразования параметра $Pr_i \in Prs$, зависящая от параметров *Prs* и *DFs;*

e.F_DF(F_i*)* *и* *e.F_DF_op(*F_i*)* – функция распределения для стохастической переменной, заданной функцией распределения F_i и операция для нее («+», «-» или «=») соответственно.

Для графа реализации *w:*

$W = \{w_i\}$ – множество графов реализации*;*

w.H – начальный узел графа реализации *w;*

w.T – конечный узел графа реализации *w:*

x – узел графа реализации, где $x \in w$;

x.v – узел МГ-сети, которому соответствует узел графа реализации $(x.v \in V)$;

x.Prs – значения вещественных параметров узла графа реализации;

x.DFs – значения стохастических параметров узла графа реализации;

$w = \{x\}$ – граф реализации, состоящий из единственного узла *x.* Тогда $w.H = w.T = x$.

Введем операции построения графа реализации:

$w = M(w_1, w_2)$ – новый граф реализации сети, являющийся объединением графов *w1* и *w2* дугой из *w2.H в w1.T.* Частным случаем данной операции будем считать $M(x, w) = M(\{x\}, w)$ и $M(w, x) = M(w, \{x\})$.

$w = M(w_1, W, w_2)$ – новый граф реализации сети, являющийся объединением дугами конечного узла *w1.T* со всеми начальными узлами $w_i.H$ $(wi \in W)$ и объединением дугами конечных узлов $w_i.T$ с начальным узлом $w_2.H$.

Обратный алгоритм расчета модифицированных ГЕРТ-сетей. Общая идея обратного алгоритма расчета МГ-сети:

– обход графа МГ-сети ведется от выбранного стока к источнику;

– обход ведется с помощью рекурсивного алгоритма;

– для каждого узла, имеющего *EOR*-вход, запускается столько же рекурсивных вызовов процедур (ветвей обхода), сколько дуг входит в данный узел;

– для каждого узла *i,* имеющего *IOR*- или *AND*-вход, запускается рекурсивный вызов процедуры расчета узла *j,* являющегося детерминированным источником узла *i,* затем рассчитываются все пути от *j* к *i* и, используя найденные пути, строятся все реализации, завершаемые узлом *i;*

– расчет вещественных и стохастических переменных производится при «выходе» алгоритма из рекурсии;

– результатом работы главной процедуры «ОРасчитатьСеть» являются множества реализаций для каждого из стоков.

Процедура «ОРассчитатьСеть» для обратного алгоритма расчета модифицированной ГЕРТ-сети

Шаг 1. Для каждой вершины v_i из множества вершин V задать число активаций $v.a$ равное 0 и объявить пустое множество возможных реализаций $(W = \{\})$.

Шаг 2. Для каждого стока vs_i из множества стоков S выполнить процедуру «ОПостроитьРеализацию».

Шаг 3. Объединить полученные реализации W_i в результате шага 2 в результирующее множество W.

Шаг 4. Конец процедуры.

Процедура «ОПостроитьРеализацию» для обратного алгоритма расчета МГ-сети

Шаг 1. Если число активаций текущего узла больше максимально допустимого количества активаций $vc.a > vc.MaxA$, то перейти к шагу 18, иначе шаг 2.

Шаг 2. Если текущий узел vc указывает на узел источника подсети vs, то перейти к шагу 3, иначе – шаг 4.

Шаг 3. К множеству реализаций W добавим узел графа реализации x, для которого: $x.v = vs$, $x.Prs = vs.Prs$, $x.DFs = vs.DFs$. Перейти к шагу 18.

Шаг 4. Увеличить $vc.a$ на 1.

Шаг 5. Если типом входной функции текущего узла $vc.In$ является входная функция типа EOR, то перейти к шагу 6, иначе – шаг 12.

Шаг 6. Для множества узлов, являющихся предками текущего узла vc, выполнить процедуру «ОПостроитьРеализацию».

Шаг 7. Объединить полученные в результате шага 6 множества графов реализаций W_{ti} во множество W_1.

Шаг 8. Для каждого узла графа реализации W_1 $x.v = vs$.

Шаг 9. Для пар узел графа реализации W_1 x и vs выполнить процедуру «Рассчитать дугу».

Шаг 10. Попарно объединить графы реализаций W_1 с $M(w_i, x_i)$ во множество W_2, где i от 1 до $|W_1|$, перейти к шагу 17.

Шаг 11. Для детерминированного источника текущего узла $vc.dR$ выполнить процедуру «ОПостроитьРеализацию».

Шаг 12. Для каждого узла xs_i графа реализации Ws, полученного в результате выполнения шага 11, выполнить процедуру «ОПостроитьРеализацию».

Шаг 13. Для каждого графа реализации $w2_i$ из множества графов реализаций $Ws2$, полученных в результате выполнения шага 12, удалить первый узел.

Шаг 14. Для всех возможных пар узлов из $Ws2$ выполнить процедуру «Рассчитать дугу».

Шаг 15. Для каждого узла x множества графов реализаций $Ws2$: $x.v = vc$ – определить параметры узла графа реализации $x.Prs$ и $x.DFs$ по множеству $\{ws2_i\}$, используя формулы (1.3)–(1.6), (1.10), (1.12).

Шаг 16. Попарно объединить графы реализаций $Ws2$ с $M(ws2_i, xs2_i)$ во множество $W2$, где i от 1 до $|Ws2|$.

Шаг 17. Уменьшить $vc.a$ на 1.

Шаг 18. Конец процедуры.

Процедура «Рассчитать дугу» расчета МГ-сети Шаг 1. $e = <v1, v2>$, $Prs2(p) = Prs1(p) * e.F\_Pr(p)$, где $v1$, $v2$ – узлы, являющиеся началом и концом дуги; $Prs1$, $Prs2$ – множества вещественных параметров начала и конца дуги соответственно; $DFs1$, $DFs2$ – множества стохастических параметров начала и конца дуги соответственно.

Шаг 2. Для каждого Pr_i из $Prs1$, при $Pr_i \neq p$ выполнить: $Prs2(Pr_i) = e.F\_Pr(Pr_i)(Prs1, DFs1)$.

Шаг 3. Для каждого F_i из *DFs1* рассчитать $DFs2(F_i)$ по формулам (1.3)–(1.13), используя значения *Prs1, DFs1,* $e.F\_DF(F_i)$, $e.F\_DF\_op(F_i)$.

Шаг 4. Конец процедуры.

Прямой алгоритм расчета модифицированных ГЕРТ-сетей. Общая идея прямого алгоритма расчета МГ-сети:

– обход графа МГ-сети ведется от источника к стокам;

– обход ведется с помощью рекурсивного алгоритма;

– для каждого узла, имеющего стохастический выход, запускается столько же рекурсивных вызовов процедур (ветвей обхода), сколько дуг выходит из данного узла;

– для каждого узла *i,* имеющего детерминированный выход, рассчитываются все пути от *i* к *j*, и, используя найденные пути, строятся все реализации, завершаемые узлом *j,* где *j* – детерминированный сток узла *i*;

– расчет вещественных и стохастических переменных производится при «углублении» алгоритма рекурсии;

– результатом работы главной процедуры «ПРасчитатьСеть» являются множества реализаций.

Следует отметить, что для использования прямого алгоритма расчета МГ-сети необходимо выполнение ограничения О5’.

Процедура «ПРассчитатьСеть» для прямого алгоритма расчета модифицированной ГЕРТ-сети

Шаг 1. Для каждой вершины v_i из множества вершин *V* задать число активаций v.a, равное 0, и объявить множество возможных реализаций *Wt = {x},* где *x.v = vr, x.Prs = iPrs, x.DFs = iDfs (vr* – узел-источник МГ-сети; *iPrs* и *iDfs* – начальные значения параметров в узле-источнике).

Шаг 2. Для каждого стока vs_i из множества стоков *S*, используя множество реализаций *Wt*, выполнить процедуру «ППостроитьРеализацию».

Шаг 3. Объединить полученные реализации Wt_i в результате шага 2 в результирующее множество *W.*

Шаг 4. Конец процедуры.

Процедура «ППостроитьРеализацию» для прямого алгоритма расчета МГ-сети

Шаг 1. Если *Wc* (множество реализаций, построенное от источника до узла vc) пустое множеств, то перейти к шагу 23, иначе – шаг 2.

Шаг 2. Если число активаций текущего узла больше максимально допустимого количества активаций *vc.a > vc.MaxA,* то перейти к шагу 3, иначе – шаг 4.

Шаг 3. *Wc = {},* перейти к шагу 23.

Шаг 4. Если vc указывает на узел источник подсети *vs*, то перейти к шагу 5, иначе – шаг 6.

Шаг 5. Объединить результирующее множество реализаций сети W с множеством *Ws, Ws = {},* перейти к шагу 23.

Шаг 6. Увеличить *vc.a* на 1.

Шаг 7. Если тип выходной функции текущего узла – стохастическая функция (*vc.Out = Stoch*), то перейти к шагу 8, иначе – шаг 13.

Шаг 8. Объявить пустые множества реализаций *Wt* и *Wt1*.

Шаг 9. Для пар текущий узел (*vc*) и сток текущего узла (v_i из *vc.S*) выполнить процедуру «Рассчитать дугу».

Шаг 10. Объединить множества *M(wk, x)* в *Wt,* где *wk* – графы реализации из множества *Wc* и *x* – стоки текущего узла (*x.v* = v_i из *vc.S*)

Шаг 11. Для каждого стока текущего узла (v_i из *vc.S*), используя множество реализаций *Wt*, полученное на предыдущем шаге, выполнить процедуру «ППостроитьРеализацию».

Шаг 12. Объединить полученные множества реализаций *Wt1* в результате шага 11 с результирующим множеством *W*, перейти к шагу 22.

Шаг 13. Объявить пустое множество реализаций *Wt2.*

Шаг 14. Для пар текущий узел (*vc*) и сток текущего узла (v_i из *vc.S*) выполнить процедуру «Рассчитать дугу».

Шаг 15. Объявить множество реализаций *Wt ={x}*, где *x* - стоки текущего узла (*x.v* = v_i из *vc.S*).

Шаг 16. Для каждого стока текущего узла (vi из vc.S), используя множество реализаций *Wt*, полученное на предыдущем шаге, выполнить процедуру «ППостроитьРеализацию».

Шаг 17. Для каждого графа реализации $w2_i$ из множества графов реализации *W2*, полученных в результате выполнения шага 16, удалить последний узел.

Шаг 18. Для всех возможных пар узлов из *W2* выполнить процедуру «Рассчитать дугу».

Шаг 19. Для узла vc.dS, являющегося детерминированным стоком текущего узла vc, определить параметры узла графа *реализации x.Prs* и *x.DFs* по множеству графов $\{w2_i\}$, используя формулы (1.3)–(1.6), (1.10), (1.12).

Шаг 20. Объединить множества *M(wk, {*$w2_i$*}, x)* в *Wt2*, где *wk* – графы реализации из множества *Wc*, $w2_i$ графы реализаций из множества W2 и x – стоки текущего узла (*x.v* = v_i из *vc.S*).

Шаг 21. Для узла *vc.dS*, являющегося детерминированным стоком текущего узла *vc*, используя множество реализаций *Wt2*, полученное на предыдущем шаге, выполнить процедуру «ППостроитьРеализацию».

Шаг 22. Уменьшить *vc.a* на 1.

Шаг 23. Конец процедуры.

Сравнение производительности прямого и обратного алгоритмов расчета модифицированной ГЕРТ-сети. Приведем качественное сравнение производительности прямого и обратного алгоритмов расчета стохастической сети. Выполнить количественную оценку производительности алгоритмов для произвольной сети не представляется возможным.

Время работы программ, использующих прямой и обратный алгоритмы обхода МГ-сети, в основном формируется из времени выполнения трех

операций: обхода графа сети, дублирования множества реализаций и расчета параметров узлов графа реализаций.

Время выполнения алгоритма обхода графа МГ-сети примерно одинаково для прямого и обратного алгоритмов.

В ходе практических испытаний над различными сетями произвольного вида выявлено, что алгоритм на базе прямого обхода графа всегда не медленнее по производительности алгоритма на базе обратного обхода графа. Однако, как уже отмечалось выше, область его применения уже (требует выполнения О5’).

Один из способов повышения точности расчетов интегральной свертки при анализе вероятностно-временных характеристик стохастических сетей заключается в том, чтобы дать возможность использовать аналитически заданные функции распределения в качестве стохастических параметров (весов) дуг сети.

Прямой алгоритм расчета стохастической сети занимает время выполнения меньшее или равное времени выполнения обратного алгоритма. Однако прямой алгоритм может быть использован только при выполнении ограничения О5’. Проверка выполнения данного ограничения может выполняться как исследователем, так и программным путем. Прямой алгоритм дает выигрыш в быстродействии, если сеть имеет хотя бы один цикл.

Предложенный способ представления функций распределения стохастических параметров при анализе вероятностно-временных характеристик стохастических сетей и численные методы расчета параметров дуг позволил создать универсальный алгоритм расчета произвольной стохастической сети.

Данный подход по расчету надежности оказывается применим для оценки программных систем, состоящих из нескольких связанных исполняемых программных модулей, когда передача данных из модуля в модуль, и активация следующего программного модуля происходит

автоматически, без участия пользователя. Такая схема предполагает не только последовательное исполнение модулей, но и варианты с ветвлением дерева исполняемых модулей. Однако, если речь идет об участии некоторого пользовательского интерфейса, где инициация того или иного программного модуля зависит от решения пользователя или наборов внешних данных, то моделирование процесса функционирования с помощью GERT-сетей в описанном режиме не представляется возможным. Таким образом, предложенный вид оценки надежности программного обеспечения также нельзя считать универсальным.

Выводы

Проблема оценки надежности программных систем является важной научной задачей к которой обращаются ученые по всему миру. Несмотря на большое количество проведенных в данной области исследований не выявлено общепринятого подхода к оценки надежности программ. Более того, ведутся споры о самих критериях оценки, многие из которых являются качественными или строятся на основании субъективных экспертных мнений.

В главе проведен анализ существующих методов оценки надежности программных систем и причин возникновения программных сбоев и ошибок. Подробно были рассмотрены методы, позволяющие учитывать качественные характеристики программных систем на этапах разработки, когда существует возможности их устранения. Также были рассмотрены методы, основанные на моделировании стохастическими GERT-сетями, позволяющие давать количественную оценку надежности программных систем в автоматическом режиме на основе анализа статистики работы модулей программной системы.

2 Разработка алгоритма выбора надежного варианта программного обеспечения

2.1 Постановка задачи выбора надежного варианта программного обеспечения

В п. 1.3 уже шла речь о том, что надежность программного обеспечения - это показатель зависящий от многих критериев. Также очевидно, что увеличивая надежность программного обеспечения разработчики обязаны думать и о других его качествах, таких как производительность, функциональность и стоимость.

С критерием функциональности вопрос стоит просто – разработчики, как правило обязаны выполнить все требования пользователей или технического задания, иначе программная система не будет считаться принятой заказчиком.

Сложнее дело обстоит с критерием производительности. Этот показатель тесно связан с понятием надежности. Действительно, при сбоях и отказах программного обеспечения его производительность, вне зависимости от способов ее измерения, будет падать. В то же время, рост производительности программного обеспечения, например, ускорение выполнения операций или одновременное выполнение большего количества операций, может привести к снижению надежности программной системы. Имеющийся опыт разработки программных систем показывает, что часто в техническом задании заказчиком устанавливаются жесткие ограничения по производительности.

Становится ясно, что речь идет о многокритериальной задаче оптимизации. Совершенно очевидно противоречие критерия стоимости разработки по отношению к критерию надежности. Время разработки можно отнести к критериям стоимости – они оказываются вполне согласованными

(при увеличении одного – второй также растет и рост обоих должен приводить к увеличению надежности).

В [70] предлагается модель оценки стоимости разработки надежного программного обеспечения, основанного на мультиверсионном подходе обеспечения надежности. Стоимость программного обеспечения рассчитывается как трудозатраты на разработку системы:

$$T_S = \sum_{j=1}^{M} \sum_{i=1}^{N_j} \left(B_{ij} T_{ij} + \left(NVP_{ij} + RB_{ij} \right) \cdot \left(NVX_{ij} + \sum_{k \in Z_{ij}}^{Kij} T_{ij}^{k} \right) \right), \qquad (2.1)$$

при ограничении

$$Z_{ij} \geq 1$$

где M – количество архитектурных уровней в архитектуре ПО;

N_j – количество компонентов на уровне j, $j \in \{1,..,M\}$;

K_{ij} – глубина программной избыточности компонента i, на уровне j;

Z_{ij} – множество версий компонента i, на уровне j;

T_{ij} – трудоемкость разработки компонента i на уровне j;

T^k_{ij} – трудоемкость разработки версии k компонента i на уровне j, $k \in Z_{ij}$, в чел-часах;

NVX_{ij} – трудоемкость разработки среды исполнения версий (приемочного теста для *RB* (*recovery block*, блок восстановления) или алгоритма голосования для *NVP* (*N-version programming*, *N*-версионное программирование);

B_{ij} – бинарная переменная, принимающая значение 1 (тогда NVP_{ij}=0, RB_{ij}=0), если в программном компоненте не используется программная избыточность, иначе равна 0;

NVP_{ij} – бинарная переменная, принимающая значение 1 (тогда B_{ij}=0, RB_{ij}=0), если в программном компоненте введена программная избыточность методом *NVP*, иначе равна 0.

RB_{ij} – бинарная переменная, принимающая значение 1 (тогда B_{ij}=0, NVP_{ij}=0), если в программном компоненте введена программная избыточность методом *RB*, иначе равна 0.

T_s – общая трудоемкость реализации программной системы.

Надежность в данном исследовании оценивается с учетом ошибок допущенных на этапе разработки (1.2) и коэффициента надежности архитектуры ПО (1.1):

$$H_{no}=R+D \rightarrow max, \quad (2.2)$$

$$T_s \rightarrow min, \quad (2.3)$$

где

$$R=\sum_{j=1}^{M}\sum_{i=1}^{Nj} PU_{ij} \times R_{ij}$$

$$D = T_s \cdot F_{pr}$$

при ограничениях

$$H_{no} \geq Lim(H_{no}),$$

$$T_s \leq Lim(T_s)$$

Коэффициент *D* связывает показатель надежности со стоимостью (трудоемкостью) разработки.

В данной постановке оптимизируемыми параметрами будут выступать:

- N_j – количество компонентов на уровне j, $j \in \{1,..,M\}$;
- Z_{ij} – множество версий компонента i, на уровне j;
- Бинарные переменные B_{ij}, NVP_{ij}, RB_{ij}, характеризующие применение подходов программной избыточности при разработке.

Каждый раз при оценивании надежности для новой программной системы экспертам придется задавать следующие константы:

T_{ij} – трудоемкость разработки компонента i на уровне j;

T^k_{ij} – трудоемкость разработки версии k компонента i на уровне j, $k \in Z_{ij}$,

в чел-часах.

M – количество архитектурных уровней в архитектуре ПО является исходными данными для данной задачи.

Fpr – коэффициент командного программирования;

Трудоемкость часто рассчитывается, исходя из объема программного кода и выбранного метода обеспечения программной избыточности.

Наибольшую сложность представляет собой расчет коэффициентов надежности каждого модуля R_{ij}. В данном исследовании для каждого варианта конфигурации проектируемой программной системы расчет коэффициентов R_{ij} проводился по алгоритмам изложенным в п. 1.4.

Таким образом, для автоматизации процесса разработки надежного ПО необходимо решить задачу многокритериальной условной смешанной оптимизации с алгоритмически заданными функциями. В качестве приемлемого средства решения такой задачи был выбран генетический алгоритм [94], который подробно будет рассмотрен в следующем разделе.

2.2 Стандартный генетический алгоритм

Эволюционные алгоритмы (ЭА), предложенные впервые для решения задач адаптации, в дальнейшем интенсивно развивались как методы решения сложных задач оптимизации. Их относят классу адаптивных стохастических алгоритмов глобальной оптимизации. Генетические алгоритмы (ГА) представляют собой семейство ЭА, имеющих общую схему. В настоящее время ГА доказали свою эффективность при решении многих NP-трудных задач поиска и оптимизации и особенно в практических приложениях, где математические модели имеют сложную структуру, или отсутствуют вовсе, и применение классических методов невозможно [8].

В основе многих концепций искусственного интеллекта лежат различные природные явления. К ним относят: искусственные нейронные

сети, основная идея которых основывается на функционировании нейронов мозга; поведенческие алгоритмы, имитирующие поведение некоторых животных в природе. ГА являются направлением более общей теории ЭА, основанной на следующем принципе: «каждый биологический вид целенаправленно развивается и изменяется для того, чтобы наилучшим образом приспособиться к окружающей среде».

ЭА базируются на коллективном обучающем процессе внутри популяции индивидуумов, каждый из которых представляет собой поисковую точку в пространстве допустимых решений данной задачи. Популяция случайно инициализируется и затем охватывает лучшие регионы поискового пространства посредством случайных процессов селекции, мутации и рекомбинации. Окружающая среда представляет качественную информацию (степень пригодности) о поисковых точках (индивидуумах), а процесс селекции отбирает тех индивидуумов, у которых значение пригодности выше. Отобранные потомки являются, в свою очередь, родителями в следующем поколении. Механизм рекомбинации перемешивает генетическую информацию родителей (тем самым рождается один или несколько потомков), и наконец, механизм мутации способствует в некоторой степени обновлению генетической информации потомков [54].

Основоположником теории эволюционных алгоритмов считают Дж. Холланда его основополагающий труд “Адаптация в естественных и искусственных системах“ лег в основу всего направления. Термин "генетические алгоритмы" ввел в 1975 г. Д. Голдберг [94]. Его книга содержит детальное обсуждение, как теоретических аспектов ГА, так и возможных областей их применения [82]:

- Искусственная жизнь,
- Автоматическое обучение,
- Извлечение данных (знаний),
- Инженерное проектирование,

– Планирование и управление,
– Моделирование, идентификация,
– Численная и комбинаторная оптимизация.

Итак, ГА базируются на теоретических достижениях теории эволюции, учитывающей микробиологические механизмы наследования признаков в природных и искусственных популяциях организмов, а также на накопленном человечеством опыте в селекции животных и растений.

Методологическая основа ГА основывается на гипотезе селекции, которая в самом общем виде может быть сформулирована так: чем лучше приспособленность особи, тем выше вероятность того, что в потомстве, полученном с ее участием, признаки, определяющие приспособленность, будут выражены еще сильнее [8].

Т.о. ГА заимствуют из биологии [80]:

– понятийный аппарат;
– идею коллективного поиска экстремума при помощи популяции особей;
– способы представления генетической информации;
– способы передачи генетической информации из поколения в поколение (генетические операторы);
– идею о превосходстве в популяции наиболее приспособленных особей.

Подобно тому, как природный хромосомный материал представляет собой линейную последовательность различных комбинаций четырех нуклеотидов, решения в ГА представляются в виде хромосом (генотипов). Генотип – строка конечной длины, состоящая из генов, представленных символами некоторого алфавита.

В ГА существует строгое различие между фенотипом (решением, выраженным в терминах поставленной задачи) и генотипом (хромосомой, представлением решения). ГА работает с генотипом, фенотип служит для

определения пригодности индивида (оценки качества решения поставленной задачи), поэтому для работы алгоритма необходимо определить некоторую функцию кодирования ($e: D \to S$, где D - пространство поиска, S - пространство представлений решений) и функцию декодирования ($e^{-1}: S \to D$). Т.о. на самом деле ГА решают не задачу $f(d) \to \underset{d \in D}{opt}$, где $f: D \to R^1$, а задачу $\mu(s) \to \underset{s \in S}{opt}$, где $\mu: S \to R^1$ и $s = e(x),\ \mu(s) = f(e^{-1}(s)) = f(x)$.

На практике наибольшее распространение получили ГА с бинарным представлением решений. Формально они решают задачу псевдобулевой оптимизации, т.е.

$$\chi(X) \to \underset{X \in B_{2^n}}{opt}, \text{ где } \chi: B_{2^n} \to R^1.$$

К этой задаче сводятся практически любые задачи с дискретными переменными (возможно выраженные в разных шкалах), а также задачи с непрерывными переменными (заменяя непрерывные переменные дискретными с заданной точностью). Наиболее часто используются стандартное бинарное кодирование и бинарные коды Грея [118].

В общем виде работу генетического алгоритма можно представить следующим образом:

1. Инициализировать случайным образом популяцию решений.
2. С помощью оператора селекции выбрать часть популяции (родителей) для порождения потомков.
3. Применить оператор скрещивания.
4. Новые решения (потомки) подвергаются мутации.
5. Формируется новая популяция: выбрать решения из родителей и потомков.
6. Повторять 2 – 5 пока не выполнится условие остановки.

На шаге инициализации задаются параметры алгоритма: длина хромосомы, размер популяции и др., а также типы и вероятности применения

основных генетических операторов (скрещивания, мутации и селекции). Если априорные сведения о пространстве поиска отсутствуют, начальная популяция генерируется случайным образом.

Оператор селекции - оператор, посредством которого индивиды выбираются для порождения потомков или для перехода в следующее поколение. Наиболее приспособленные особи должны выбираться с большей вероятностью для сохранения своих генов в следующем поколении. Т.о., оператор селекции позволяет сконцентрировать поиск на наиболее многообещающих регионах пространства.

Предложено множество различных схем отбора в ГА. Конкретный тип оператора селекции (а также и др. генетических операторов) проектируется исходя из решаемой задачи (например, когда требуется контроль допустимости решений, учет ограничений, многокритериальности и т.д.), однако наиболее распространены следующие базовые типы селекции.

В *пропорциональной селекции* вероятность индивида быть отобранным пропорциональна его пригодности. Вероятность вычисляется следующим образом (для задачи минимизации):

$$P(X^i) = \frac{-\ fitness(X^i) + C}{r * C - \sum_{j=1}^{r} fitness(X^j)},$$

где r - размер популяции, $C : P(X^i) \geq 0, \forall i$, $\sum_{j=1}^{r} P(X^j) = 1$.

Пропорциональная селекция обладает следующими недостатками: преждевременная сходимость и стагнация.

Стагнация возникает, когда на определенном этапе поиска все индивиды получают относительно высокую и примерно равную пригодность, что приводит к очень низкому селективному давлению (наилучшее решение лишь немного предпочитается худшему).

Преждевременная сходимость (проблема супериндивида) возникает, когда на ранних этапах появляется индивид с пригодностью намного

большей, чем у других индивидов в популяции, но очень плохой с точки зрения решаемой задачи. Вероятность супериндивида быть отобранным стремится к единице, в то время как вероятности других членов популяции – к нулю. В итоге он копирует себя в следующее поколение и вскоре «широкий» поиск прекращается.

При применении *ранговой селекции* индивиды популяции ранжируются в соответствии с их пригодностью: $R_i < R_j$ если $f(X^i) \le f(X^j)$. Тогда

$$P(X^i) = \frac{R_i}{\sum_{k=1}^{r} R_k} = \frac{i}{\sum_{k=1}^{r} i} = \frac{2i}{r(r+1)}, \text{ где } \sum_{j=1}^{r} P(X^j) = 1.$$

Ранговая селекция устраняет недостатки пропорциональной: нет стагнации, т.к. даже к концу работы алгоритма $P(X^1) \neq P(X^2) \neq \ldots$, нет преждевременной сходимости, т.к. нет индивидов с вероятностью отбора близкой к единице.

В *турнирной селекции* для отбора индивида создается группа из t ($t \ge 2$) индивидов, выбранных случайным образом. Индивид с наибольшей пригодностью в группе отбирается, остальные – отбрасываются. Параметр t называется размером турнира. Наиболее популярным является бинарный турнир. Этот тип селекции не требует сортировки популяции и вычисления пригодности для всех индивидов. Недостатки: худший индивид никогда не выбирается.

Селекция с усечением. В процессе селекции с усечением с порогом τ, только доля τ из всех лучших индивидов может быть отобрана, причем в этой доле каждый имеет одинаковую вероятность отбора.

Элитарная селекция. Как минимум одна копия лучшего индивида всегда переходит в следующее поколение. Преимущества: гарантия сходимости. Недостатки: большой риск захвата локальным оптимумом.

Оператор скрещивания (рекомбинации) – генетический оператор поиска. При скрещивании отобранные индивиды (родители) по заданному правилу передают части своих хромосом. Потомок может унаследовать

только те гены, которые есть у его родителей. Существуют различные схемы скрещивания: точечное, равномерное, скрещивание более чем двух родителей и другие. При точечном скрещивании для выбранных родителей выбираются точки разрыва хромосомы, родители определенным образом обмениваются соответствующими участками хромосом. При равномерном – соответствующий ген потомка может быть унаследован от любого родителя с равной вероятностью. Скрещивание осуществляется с вероятностью p_{xover}, иначе с вероятностью $(1-p_{xover})$ родители клонируются в следующее поколение [81, 89].

Наиболее популярным типом скрещивания является одноточечное скрещивание – случайно выбирается точка разрыва, родительские хромосомы разрываются в этой точке и обмениваются правыми частями (рисунок 2.1).

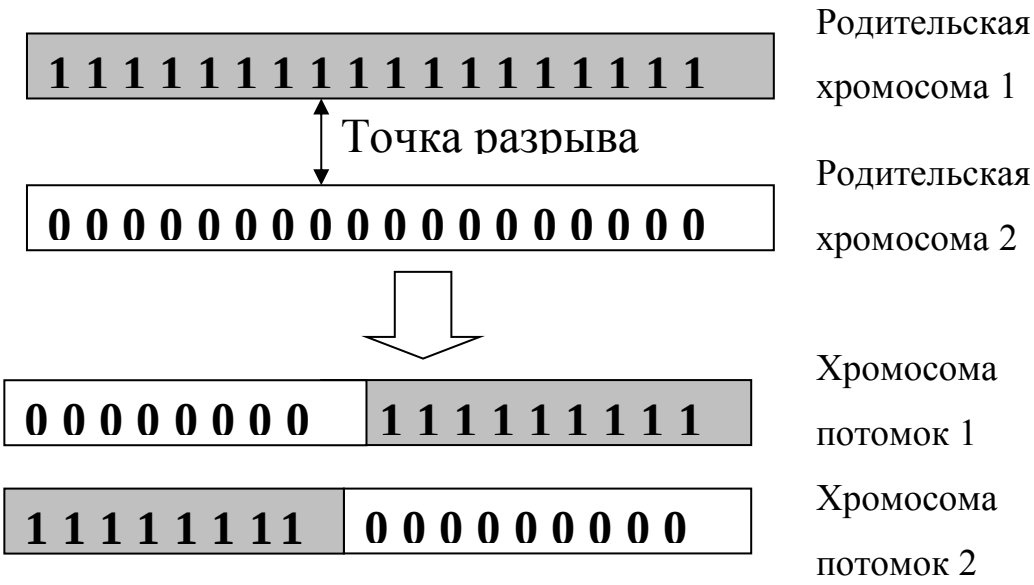


Рисунок 2.1 - Одноточечное скрещивание.

При двухточечном скрещивании хромосома как бы замыкается в кольцо, выбираются 2 точки разрыва, родители обмениваются частями (рисунок 2.2).

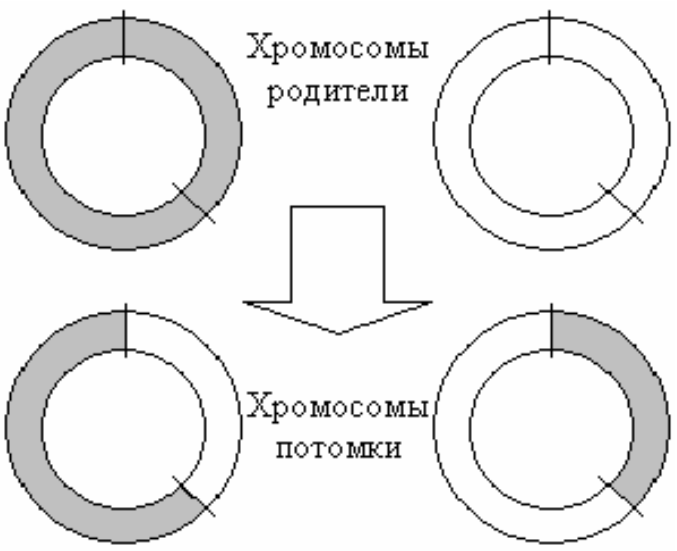


Рисунок 2.2 - Двухточечное скрещивание.

При равномерном скрещивании потомок может унаследовать с равной вероятностью гены любого из родителей (рисунок 2.3).

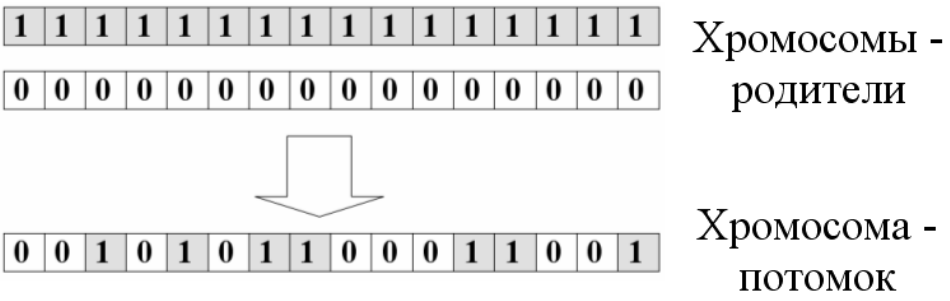


Рисунок 2.3 - Равномерное скрещивание.

Равномерное скрещивание по всей популяции (uniform gene pool recombination) получается применением равномерного скрещивание к всем членам популяции, т.е. потомок может унаследовать любой ген, имеющийся в популяции в заданной позиции хромосомы [105].

Оператор мутации – одноместный оператор поиска, случайное изменение в одном или нескольких генах индивида. В ГА мутация рассматривается как метод восстановления потерянного генетического материала, а не как поиск лучшего решения. Мутация применяется к генам с очень низкой вероятностью $p_m \in [0.001; 0.01]$. Хорошим эмпирическим

правилом считается выбор вероятности мутации равным $p_m = \frac{1}{n}$, где n - число генов в хромосоме (в среднем хотя бы один ген будет подвержен мутации). В случае бинарного алфавита мутация состоит в инвертировании случайно выбранных битов (рисунок 2.4).

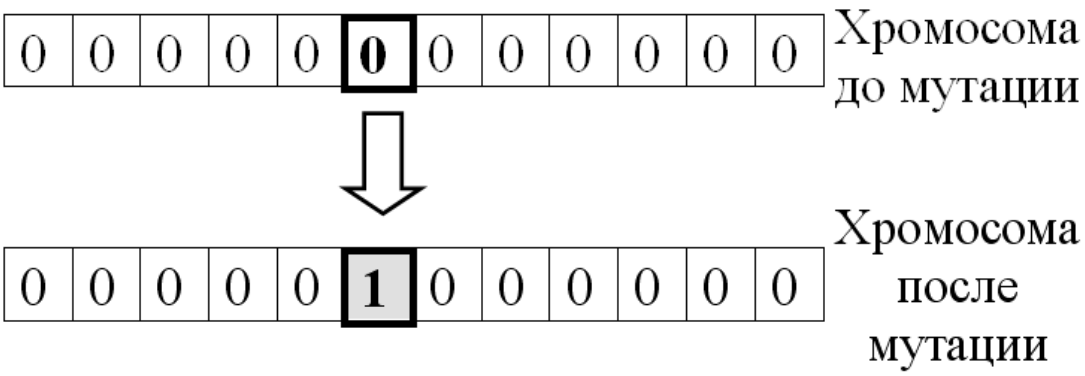


Рисунок 2.4 - Пример мутации в ГА.

Пригодность индивида – это некоторая количественная оценка качества решения поставленной задачи. Функция пригодности может быть спроектирована с учетом особенностей решаемой задачи (что делает ГА довольно гибким, универсальным). Обычно функция пригодности принимает положительные значения, и значение пригодности максимизируют.

В общем случае при решении задач безусловной оптимизации в качестве значения функции пригодности может выступать значение оптимизируемого функционала ($fitness(X^i) = \chi(X^i)$).

2.3 Алгоритмы многокритериальной оптимизации

Как было отмечено в п. 2.1 выбор надежного варианта программного обеспечения – это многокритериальная задача. Для проведения оптимизации

сразу по нескольким критериям, необходимо использовать специально разработанные для этого методы.

Задачи многокритериальной оптимизации встречаются во многих областях. Примером может служить классическая задача из экономики о производстве масла и пушек. Здесь в качестве критериев выступает благосостояние и безопасность. В зависимости от области применения, в многокритериальных задачах могут фигурировать различные критерии. В нашем случае в качестве критериев многокритериальной задачи будут выступать стоимость и производительность многопроцессорной вычислительной системы. Однако, независимо от области применения, для решения этих задач данного класса применяются одни и те же, специально разработанные алгоритмы и методы [67].

Многокритериальная оптимизация, также известная как векторная оптимизация, может быть определена как задача нахождения вектора переменных-решений, которые удовлетворяют ограничениям и доставляют оптимум вектор-функции, чьи элементы представляют собой целевые функции. Эти функции формируют математическое описание представления критериев, которые обычно находятся в конфликте друг с другом. Поэтому термин «оптимизировать» означает нахождение такого рода решения, которое бы давало значения всех целевых функций, приемлемых для ЛПР [19].

Определение: (Задача многокритериальной оптимизации) Общая ЗМО включает множество параметров n (переменные решения), k целевых функций, и m ограничений. Целевые функции и ограничения - функции переменных пространства решения. Цель оптимизации:

максимизировать $\boldsymbol{y = f(x)} = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x}))$

при условии $\boldsymbol{e(x)} = (e_1(\boldsymbol{x}), e_2(\boldsymbol{x}), \ldots, e_m(\boldsymbol{x})) \leq \boldsymbol{0}$

где $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \boldsymbol{X}$

$\boldsymbol{y} = (y_1, y_2, \ldots, y_n) \in \boldsymbol{Y}$,

то есть, $\boldsymbol{x}$ - вектор решения, $\boldsymbol{y}$ - целевой вектор, $\boldsymbol{X}$ – пространство решений и $\boldsymbol{Y}$ – целевое пространство. Ограничения $\boldsymbol{e(x) \leq 0}$ определяют множество допустимых решений [20, 119].

Определение: (Множество допустимых решений) Множеством допустимых решений *D* называется множество векторов решений *x,* которые удовлетворяют ограничениям *e(x)*:

$$D = \{ x \in X \mid e(x) \leq 0 \}$$

Критерии $f_1(x),\ldots,f_k(x)$ могут быть согласованными, нейтральными и конфликтующими. В первом случае оптимизация одного из критериев приводит к улучшению других. Во втором случае оптимизация одного критерия никак не влияет на другие. Интерес представляет случай конфликтующих (противоречивых) критериев, когда попытка улучшить один из них приводит к ухудшению других. В таком случае решение возможно только на основе компромисса. Математическая модель компромисса в оптимизации обычно строится на основе понятия множества Парето, названного так в честь итальянского экономиста, первым исследовавшего такие модели в начале 20-го века [54]. В нашей задаче, сформулированной в разделе 1.3, представленные критерии надежности и стоимости – конфликтующие. Следовательно, компромиссное решение следует искать как множество Парето.

Определение: решение $x_0 \in D$, называется Парето-оптимальным (недоминируемым, неулучшаемым, паретовским), если во множестве допустимых решений D не существует решения, которое по целевым функциям было бы не хуже чем $\mathbf{x_0}$ и, по крайней мере, по одной целевой функции было бы лучше, чем $\mathbf{x_0}$.

Все Парето-оптимальные решения называются множеством Парето или паретовским множеством; соответствующие целевые векторы формируют Парето-оптимальный фронт или поверхность [52].

Из всего вышесказанного можно сделать следующий вывод. Для любой допустимой точки, лежащей вне множества Парето, найдется точка во множестве Парето, дающая по всем целевым функциям значения не хуже, чем в этой точке и хотя бы по одной целевой функции – строго лучше. Отсюда следует, что решение многокритериальной задачи оптимизации целесообразно выбирать из множества Парето, т.к. любое другое, очевидно, может быть улучшено некоторой точкой Парето как минимум по одному критерию без ухудшения других критериев. С точки зрения математики решения из множества Парето не могут быть предпочтены друг другу, поэтому после формирования множества Парето задача может считаться математически решенной [54].

Классические методы, для построения множества Парето, объединяют все критерии в одну функцию-свертку по аналогии со способом принятия решения до поиска. Однако, параметры этой функции не контролируются при оптимизации, они систематически варьируются. Выполняется несколько запусков процесса оптимизации с различными параметрами настройки, чтобы найти оптимальное паретовское множество. В основном, эта процедура независима от основного алгоритма оптимизации. Некоторые представители этого класса методов - метод аддитивной свертки [88], метод ограничений [88], целевое программирование [115], и минимаксный подход [99]. Принимая во внимание разнообразие методов, обсуждены два первых упомянутых.

Аддитивная свертка

Оригинальная задача многокритериальной оптимизации (ЗМО) преобразовывается к задаче однокритериальной оптимизации ЗОО, формированием линейной комбинации из исходных критериев:

максимизировать $y = f(x) = w_1 \cdot f_1(x) + w_2 \cdot f_2(x) + \ldots + w_k \cdot f_k(x)$,

при условии $x \in X_f$

w_i называют весами, которые удовлетворяют условию нормировки $\Sigma\ w_i=1$, $w_i>0\ \forall i$. Решая вышеупомянутую задачу оптимизации для различных комбинаций весов, получаем несколько решений. При условии, что используется точный алгоритм оптимизации, и все веса положительные, этот метод найдет только те Парето-оптимальные решения, которые могут быть найдены достаточно просто. Предполагается, что допустимый вектор решения *a* максимизирует *f* для данной комбинации весов и не Парето-оптимален. Таким образом, есть решение *b,* которое доминирует *a*, то есть, например, $f_1(b)>f_1(a)$ и $f_i(a)\geq f_i(b)$, для $i=2, ...,k$. Поэтому, $f(b) > f(a)$, что является противоречием в предположении, что *f(a)* является максимумом. Главный недостаток этого метода - то, что он не может строить все Парето-оптимальные решения с невыпуклыми поверхностями фронта. Это проиллюстрировано на рисунке 2.5, основанном на примере проектирования системы [119]. Для фиксированных весов w_1, w_2 найдено решение $\boldsymbol{x}$, которое максимизирует $y=w_1 \cdot f_1(\boldsymbol{x})+w_2 \cdot f_2(\boldsymbol{x})$. Это равенство можно преобразовать к равенству $f_2(\boldsymbol{x})=-w_1/w_2 \cdot f_1(\boldsymbol{x})+y/w_2$, которое задает линию с наклоном $-(w_1/w_2)$ и смещением y/w_2 в целевом пространстве (сплошная линия на рисунке 2.5). Графически, процесс оптимизации соответствует перемещению этой линии вверх до тех пор, пока над ней не останется ни одного допустимого целевого вектора и по крайней мере один допустимый целевой вектор (здесь A и D) не будет находиться на ней. Тем не менее, точки *B* и *C* никогда не максимизируют *f*. Если наклон увеличить, *D* придаст большее значение *f* (верхний пунктир); если наклон уменьшить, *A* придаст большее значение *f,* чем *B* и *D* (нижний пунктир).

Однако, неоднократно отмечались ошибки и противоречия, которые делает человек при назначении весов критериев. Достаточно обстоятельный обзор различных методов назначения весов подводит к выводу, что не существует корректных методов решения человеком этой задачи. Такое

поведение человека при решении многокритериальных зада является повторяющимся и устойчивым. Имеются результаты экспериментов, из которых следует, что человек назначает веса критериев с существенными ошибками по сравнению с объективно известными, что назначаемые веса противоречат его непосредственным оценкам альтернатив и т.д. Хотя дискуссия о возможности использования весов в методах принятия решений еще продолжается, полученных данных уже достаточно, чтобы считать эту операцию достаточно сложной для ЛПР [15].

Метод ограничений

Другая методика, которая не тяготеет к выпуклым частям Парето-оптимального фронта преобразовывает k-1 из k критериев в ограничения. Оставшийся критерий, который может быть выбран произвольно, является целевой функцией вновь получившейся задачей однокритериальной оптимизации:

максимизировать $y = f(\boldsymbol{x}) = f_h(\boldsymbol{x})$,

при условии $e_i(\boldsymbol{x}) = f_i(\boldsymbol{x}) \geq \epsilon_i$, $(1 \leq i \leq k, i \neq h)$,

$x \in X_f$.

Различные границы, ϵ_i, являются параметрами, которые варьируются оптимизатором для нахождения множества Парето-оптимальных решений.

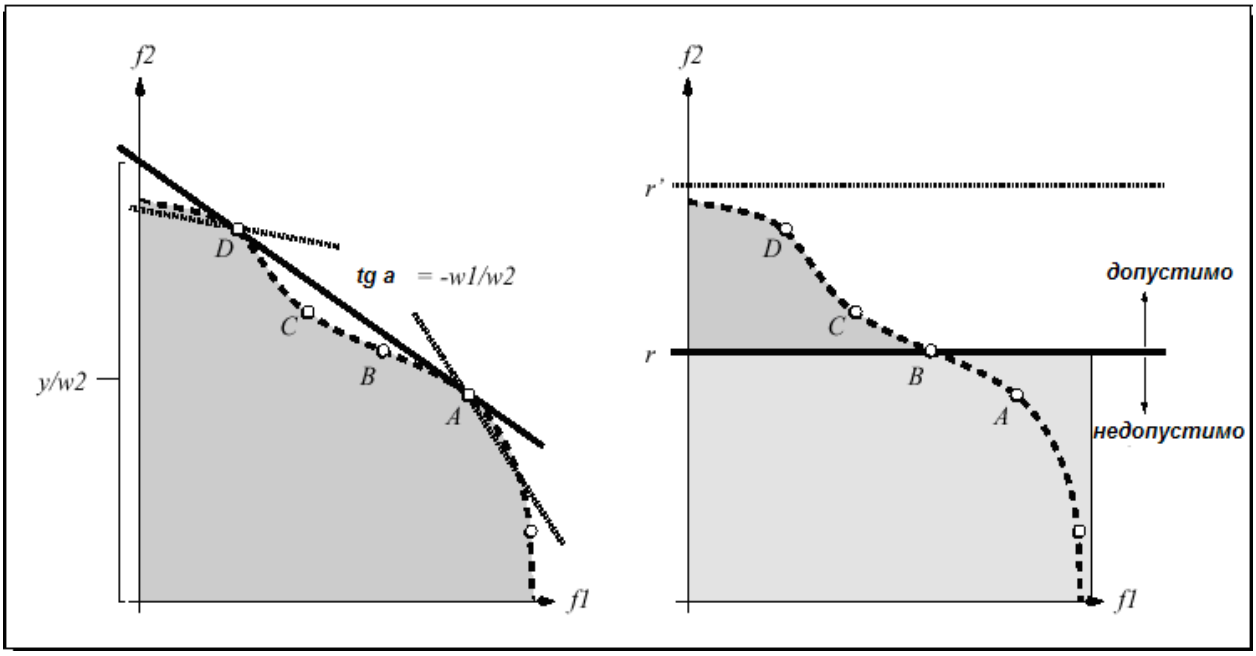


Рисунок 2.5 - Графическая интерпретация метода аддитивной свертки (слева) и метода ограничений (справа)

Как изображено на рисунке справа, метод ограничения способен найти решения, связанные с невыпуклыми частями кривой фронта. Установка h =1 и $\epsilon_2 = r$ (сплошная линия) делает решение, представленное A, недопустимым относительно ограничений, в то время как вектор решения, связанный с *B,* максимизирует f_2 среди остающихся решений. Рисунок 2.5 также показывает недостаток данного метода. Если не выбрать более низкие границы, соответственно, (ϵ_2=r'), полученное множество решений может быть пустым, то есть, решение задачи однокритериальной оптимизации не будет получено. Чтобы избежать этой ситуации, нужно знать заранее подходящий диапазон значений для ϵ_i.

Для задач однокритериальной оптимизации могут использоваться хорошо изученные алгоритмы и это делает традиционные подходы привлекательными и популярными. Для крупномасштабных задач, едва ли подходит любой метод многокритериальной оптимизации. Предшествующие разделы о методе аддитивной свертки и методе ограничений показали, что

классические стратегии оптимизации могут сопровождаться некоторыми недостатками.

- Некоторые методы, такие как, метод аддитивной свертки, могут быть чувствительны к характеру формы Парето-оптимального фронта.
- Может требоваться некоторая дополнительная информация, которая чаще всего неизвестна.

Кроме того, все классические методы требуют несколько циклов оптимизации для получения Парето-оптимального множества. Поскольку циклы выполняются независимо друг от друга, они не могут использоваться согласованно, а их согласование, в свою очередь, может вызвать чрезмерное количество вычислений. Хотя, это, опять же, зависит от приложения.

Недавно, альтернативой классическим методам стали эволюционные алгоритмы, с помощью которых, во-первых, могут быть обработаны большие пространства поиска и, во-вторых, за один цикл оптимизации может быть получено много альтернативных решений. Кроме того, они могут быть реализованы таким образом, чтобы избегать вышеупомянутых трудностей.

Из-за свойственного им параллелизма, эволюционные алгоритмы имеют возможность нахождения разнообразных Парето-оптимальных решений за один цикл оптимизации. Однако, для многих приложений не представляется возможным найти приемлемые решения, включающие полное Парето-оптимальное множество. Таким образом, главные цели оптимизации для ЗМО могут быть сформулированы как:

- ✓ Расстояние от найденного недоминируемого фронта до Парето-оптимального фронта должно быть минимальным.
- ✓ Желателен широкий спектр полученных решений.
- ✓ Распределение полученного недоминируемого фронта должно быть максимизировано, то есть, для каждого критерия недоминируемости решениями должен быть покрыт широкий диапазон значений.

Согласно этим трем целям в случае использования эволюционного

алгоритма многокритериальной оптимизации, должны быть решены две главных проблемы:

1. Как выполнять назначение пригодностей и отбор, чтобы достичь Парето-оптимального множества.
2. Как разнообразить популяцию, чтобы предотвратить преждевременную сходимость для достижения хорошо распределенного и равномерного недоминированного множества [114].

Ниже представлена классификация общих методов, которые решают данные проблемы. В случае однократного процесса оптимизации, для достижения Парето-оптимального множества упор делается на всей популяции в целом.

В отличие от однокритериальной оптимизации, где целевая функция и функция пригодности часто идентичны, для многокритериальной оптимизации при назначении пригодностей и отборе надо учитывать именно многокритериальность поставленной задачи. Вообще, различаются эволюционные методы, где критерии рассматриваются отдельно, подходы, основанные на классическом методе уплотнения, и методы, напрямую использующие концепцию доминирования по Парето.

Вместо объединения критериев в функцию-свертку этот класс эволюционных алгоритмов переключается между критериями в течение стадии отбора. Каждый раз, когда выбирается кандидат для репродукции, потенциально любой критерий может решить, какой именно индивид будет отправлен во временную популяцию. Вследствие этого, шаги назначения пригодности и селекции общего эволюционного алгоритма обычно объединяются или выполняются поочередно. Например, Schaffer [111] предложил заполнять равные части временной популяции согласно разным критериям, в то время как Fourman [93] осуществил схему отбора, где индивидуумы сравниваются по определенному или случайному порядку критериев. Позже, Kursawe [100] предложил назначать вероятность для

каждого критерия, которая определяет, будет ли критерий использован в следующем шаге отбора и может задаваться пользователем или назначаться случайно. Все эти подходы, вероятно, имеют склонность к, так называемому, «крайнему» решению и чувствительны к невыпуклому Парето-оптимальному фронту (Horn, [96]).

Концепция назначения пригодности индивидууму на основе доминирования по Парето сначала была предложена в [94]. Он представил “революционную схему” итеративной процедуры ранжирования: сначала всем первостепенно недоминируемым индивидуумам назначается ранг один и они временно удаляются из популяции. Затем, следующим недоминируемым индивидуумам назначается ранг два, и так далее. В конечном итоге, значение пригодности индивидуума определяет его ранг. Примечателен тот факт, что пригодности назначаются исходя из качества всей популяции, в то время как в других методах используется независимое назначение пригодности каждому индивидууму.

Эта идея была продолжена многочисленными исследователями, вследствие чего появились новые схемы назначения пригодностей, основанные на принципе доминирования по Парето, например, (Fonseca и Fleming [91, 92]; Horn [96], Goldberg [94]). Хотя этот класс ЭА теоретически способен к обнаружению любого Парето-оптимального решения, на его работу может повлиять размерность пространства поиска.

Существует еще множество подходов к решению задач многокритериальной оптимизации с помощью эволюционных методов, перечислим только основные из них: методы, основанные на массовом отборе с вариацией параметров, доопределение, переинициализация популяции, уплотнение, элитизм и целая группа подходов, основанная на разнообразии популяции или “нишировании”, это такие подходы как выравнивание пригодностей, ограниченное скрещивании, изоляция.

Метод SPEA

Данный метод признан одним из самых эффективных среди всех известных подходов в генетических алгоритмах для решения задач многокритериальной оптимизации.

Особенностями данного метода являются:

- Для назначения индивидам скалярного значения пригодности используется концепция Парето доминирования;
- Индивиды, недоминируемые относительно других членов популяции, хранятся внешне в специальном внешнем множестве;
- Для уменьшения количества индивидов, хранящихся во внешнем множестве, выполняется кластеризация, что в свою очередь никак не влияет на приобретенные в процессе поиска свойства индивидов.

Уникальность и преимущества метода *SPEA* заключаются в том, что пригодность каждого индивида популяции в данном методе определяется только относительно индивидов внешнего множества, независимо от того, доминируют ли индивиды популяции друг друга;

Несмотря на то, что «лучшие» индивиды, полученные в предыдущих поколениях, хранятся отдельно – во внешнем множестве, все они принимают участие в селекции;

Для предотвращения преждевременной сходимости, в методе *SPEA* используется особый механизм образования ниш, где деление общей пригодности осуществляется не в смысле расстояния между индивидами, а на основе Парето доминирования.

Процедура метода *SPEA* [12]:

Вход: N (размер популяции), $\overline{N}$ (размер внешнего множества), T (максимальное число поколений), p_c (вероятность скрещивания), p_m (вероятность мутации).

Выход: A (недоминируемое множество).

Шаг 1. Инициализация: создать начальную популяцию P_0, согласно 1 этапу схемы общего эволюционного алгоритма, и пустое внешнее множество $\overline{P_0} = 0$. Положить t=0.

Шаг 2. Модернизация внешнего множества: положить промежуточное внешнее множество $\overline{P'} = \overline{P_t}$.

а) копировать индивидов, чьи векторы решений недоминируемы относительно $m(P_t)$ в $\overline{P'}$: $\overline{P'} = \overline{P'} + \{i \mid i \in P_t \wedge m(i) \in p(m(P_t))\}$.

б) удалить тех индивидов из $\overline{P'}$, чьи соответствующие векторы решений слабо доминируемы относительно $m(\overline{P'})$, то есть, если существует пара (i, j) с индивидами $i, j \in \overline{P'}$ и $m(i) \succeq m(j)$, то $\overline{P'} = \overline{P'} - \{j\}$.

в) уменьшить посредством кластеризации (с параметрами $\overline{P'}$ и $\overline{N}$) число индивидов, хранящихся во внешнем множестве, и поместить результирующее уменьшенное множество в $\overline{P}_{t+1}$.

Шаг 3. Назначение пригодности: вычислить значения пригодности индивидов в P_t и $\overline{P_t}$.

Шаг 4. Селекция: положить P'=0 и для $s = \overline{1, N}$ выполнить:

а) случайным образом выбрать двух индивидов $i, j \in P_t + \overline{P}_t$,

б) если $F(i) < F(j)$, то $P' = P' + \{i\}$, иначе $P' = P' + \{j\}$ (в случае задачи минимизации).

Шаг 5. Рекомбинация.

Шаг 6. Мутация

Шаг 7. Окончание: положить $P_{t+1} = P'''$ и $t = t + 1$. Если $t \geq T$ или выполняется какой-то другой критерий остановки, тогда $A = p(m(\overline{P}_t))$ – есть искомое недоминируемое множество, иначе перейти на шаг 2.

Назначение пригодности в методе *SPEA*:

Вход: P_t (популяция), $\overline{P_t}$ (внешнее множество).

Выход: F (значения пригодности).

Шаг 1. Каждому индивиду $i \in \overline{P_t}$ присваивается значение $S(i) \in [0,1)$, называемое «силой» индивида (отражает пригодность недоминируемого решения), которое пропорционально количеству членов популяции $j \in P_t$, для которых $m(i) \succeq m(j)$. $S(i) = \frac{|\{j \mid j \in P_t \wedge m(i) \succeq m(j)\}|}{N+1}$.

Таким образом, пригодность индивида i, принадлежащего внешнему множеству, равна его «силе»: $F(i) = S(i)$.

Шаг 2. Пригодность индивида $j \in P_t$ вычисляется посредством суммирования «сил» всех индивидов $i \in \overline{P_t}$, хранящихся во внешнем множестве, чьи векторы решений слабо доминируют $m(j)$. К общему полученному числу добавляется единица, чтобы гарантировать, что индивиды внешнего множества $\overline{P_t}$ имеют лучшую пригодность, по сравнению с индивидами из P_t (то есть, чем меньше пригодность, тем у индивида j больше шансов перейти в следующее поколение:

$F(j) = 1 + \sum_{i \in \overline{P}_t, m(i) \succeq m(j)} S(i)$, где $F(j) \in [1, N)$.

Механизм кластеризации в методе *SPEA*:

Вход: $\overline{P'}$ (внешнее множество), $\overline{N}$ (размер внешнего множества).

Выход: $\overline{P_{t+1}}$ (модернизированное внешнее множество).

Шаг 1. Инициализировать множество кластеров C. Каждый индивид $\overline{i \in P'}$ образует отдельный кластер: $C = \bigcup_{i \in \overline{P'}} \{\{i\}\}$.

Шаг 2. Если $|C| \leq \overline{N}$, перейти на Шаг 5, иначе перейти на Шаг 3.

Шаг 3. Вычислить расстояния между всеми возможными парами кластеров. Отдаленность d_c двух кластеров c_1 и $c_2 \in C$ друг от друга

определяется как среднее расстояние между парами индивидов, принадлежащих этим кластерам:

$$d_c = \frac{1}{|c_1| \cdot |c_2|} \cdot \sum_{i_1 \in c_1, i_2 \in c_2} d(i_1, i_2),$$

где функция d отражает расстояние (в пространстве целей) между двумя индивидами i_1 и i_2.

Шаг 4. Определить два кластера c_1 и $c_2 \in C$ с минимальным расстоянием d_c. Эти кластеры объединяются в один больший по размеру кластер: $C = C \setminus \{c_1, c_2\} \bigcup \{c_1 \cup c_2\}$. Перейти на Шаг 2.

Шаг 5. Для каждого кластера выбрать репрезентативного индивида, а всех остальных индивидов из него удалить. (Таким образом, репрезентативный индивид – это точка с минимальным средним расстоянием до всех остальных точек кластера.) Определить уменьшенное недоминируемое множество путем объединения репрезентативных индивидов всех кластеров: $\overline{P}_{t+1} = \bigcup_{c \in C} c$.

Метод *SPEA2*

В 2001 г. разработана модификация *SPEA*, получившая название *SPEA2* [12], При разработке этого алгоритма были учтены следующие недостатки исходной схемы *SPEA*:

В случае наличия во внешнем множестве только одного индивида все представители основной популяции будут иметь одно и то же значение пригодности, вне зависимости от их доминируемости / недоминируемости относительно друг друга;

Поддержание разнообразия осуществляется только во внешнем множестве посредством механизма кластеризации; при слабом разбросе

значений пригодности индивидов основной популяции также целесообразно учитывать степень их различия в критериальном пространстве;

Существующий механизм кластеризации не всегда обеспечивает наилучшее распределение недоминируемых решений.

В соответствии с этими замечаниями в алгоритм внесен ряд изменений: [12]

1) «Сила» *S* определяется не только для внешнего множества $\overline{P}$, но и для основной популяции *P* по следующей формуле:

$$S(i) = |\{ j \mid j \in P_t + \overline{P}_t \wedge m(i) \succeq m(j) \}|.$$

2) Для индивидов из обоих множеств назначается «сырая» пригодность

$$R(i) = \sum_{i \in P_t + \overline{P}_t, m(i) \succeq m(j)} S(i).$$

3) Окончательная пригодность определяется с целью поддержания разнообразия популяции. Для этого вычисляются все возможные взаимные расстояния между индивидами в обоих множествах в критериальном пространстве. Далее для каждого индивида расстояния от него до остальных представителей популяций сортируются в порядке возрастания. Определяется *k*-е расстояние для *i*-го индивида. Как правило, $k = \sqrt{N + \overline{N}}$. Вычисляются значения:

$$D(i) = \frac{1}{d_i^k + 2}.$$

Конечное значение пригодности определяется как сумма *R(i)* и *D(i): F(i) = R(i) + D(i).*

4) Размер внешнего множества является постоянным. В случае недостаточности числа недоминируемых индивидов из *P* внешнее множество «доукомплектовывается» наилучшими доминируемыми индивидами из *P*. В случае избыточности числа недоминируемых индивидов осуществляется модифицированная кластеризация:

- определяются все взаимные расстояния между недоминируемыми индивидами d_{ij};

- расстояния d_{ij}, соответствующие каждому i-у индивиду, сортируются в порядке возрастания; получаем значения d_i^k;

- определяются индивиды с минимальным значением d_i^1;

- среди индивидов с минимальным значением d_i^1 (таких индивидов как минимум два) удаляется индивид с минимальным значением d_i^2; при совпадении значений d_i^2 сравниваются значения d_i^3 и т.д.

- процедура сокращения числа недоминируемых индивидов продолжается до достижения их числа размера внешнего множества.

Подобная процедура обеспечивает более репрезентативное распределение недоминируемых решений по сравнению с кластеризацией в SPEA.

5) В селекции принимают участие только индивиды из внешнего множества. Это позволяет использовать в алгоритме не только турнирную селекцию, но и другие ее типы – пропорциональную, ранговую. Кроме того, при совпадении значений N и $\overline{N}$ множества P и $\overline{P}$, по сути, становятся идентичными.

Метод аппроксимации фронта Парето с помощью метрик *GD, IGD*

Существует множество методов оценки недоминируемых решений (Парето фронта). Очевидно, что исследователи заинтересованы иметь в качестве результата работы генетического алгоритма парето-недоминируемые решения. Часто при этом много внимания уделяется вопросу разнообразия решений. Рассмотренные методы *SPEA*, ориентированы на поддержание разнообразия в популяциях. При этом первостепенное значение имеет способ измерения расстояний между решениями. В данном исследовании предлагается использовать в качестве способа измерения расстояний между решениями в популяции генетического

алгоритма метрику *GD (Generational Distance) и IGD (Inverted Generational Distance)*[90]. По данным метрикам вычисляется расстояние от точек из текущего множества решений до точек из истинного множества Парето данной задачи. Речь идет о задачах в которых заранее известен истинный фронт Парето.

Generational Distance

$$GD(A) := \frac{1}{N}\left(\sum_{i=1}^{N} d_i^p\right)^{1/p},$$

$A = \{a_1, \ldots, a_N\}$ – аппроксимация фронта Парето (результат работы алгоритма), $F(P_Q) = \{y_1, \ldots, y_M\}$ - Истинный фронт Парето, где d_i - минимальное расстояние от a_i до $F(P_Q)$ в эвклидовой метрике.

Другими словами в метрике *GD* измеряется расстояние (минимальное или максимальное, в зависимости от типа задачи оптимизации) от каждой точки полученного набора решений, до каждой точки истинного фронта Парето, а затем оно усредняется по количеству точек, что даёт нам примерную погрешность между найденным и истинным решениями.

Inverted Generational Distance

$$IGD(A) := \frac{1}{M}\left(\sum_{i=1}^{M} \tilde{d}_i^p\right)^{1/p}$$

$A = \{a_1, \ldots, a_N\}$ – аппроксимация фронта Парето (результат работы алгоритма), $F(P_Q) = \{y_1, \ldots, y_M\}$ - Фронт Парето, где $\tilde{d}_i$ - минимальное расстояние от y_i до A в эвклидовой метрике.

В метрике *IGD* берется обратное расстояние, то есть вычисляется расстояние от каждой точки полученного решения (минимальное или

максимальное, в зависимости от типа задачи оптимизации) до каждой точки истинного решения и усредняется уже по количеству точек найденного решения, что дает более обширную информацию о полученных результатах, так как такой подход к измерению расстояния позволяет узнать равномерность покрытия найденных решений. [12]

Основная идея разработанного алгоритма

В рамках проведения данного исследования, при анализе алгоритмов *SPEA* и *SPEA2*, было решено использовать в них для оценки решений метрики *GD* и *IGD*. Основанием для такого решения послужило то, что для работы указанных алгоритмов и так используется нормировка по каждому из критериев задачи.

Главной проблемой использования метрик *GD* и *IGD* является то, что для их применения требуется информация о истинном фронте Парето задачи. А это возможно только для уже решенных задач или для задач, в которых возможен полный перебор всего множества решений. Таким образом для решения реальных задач такой подход не годится. В данном исследовании предлагается использовать *GD* и *IGD* в которых вместо фронта Парето при оценке решений будут подставляться «Недостижимый фронт Парето». Под этим термином будет пониматься множество недоминируемых решений, которые не могут быть достигнуты в ходе работы алгоритма. В системном анализе такой подход называется «Идеальным проектированием» [59], когда исследователю предлагают забыть об ограничениях задачи и использовать ресурсы, которыми он не располагает. В практических задачах в качестве «Недостижимого фронта Парето» можно использовать решение, которое по всем параметрам устраивает заказчика.

Суть в том, что изначально для задачи будет задаваться недостижимый фронт (набор недостижимых решений), а получаемые решения в ходе

генетического алгоритма будут сравниваться с ним при помощи метрики GD (IGD), тем самым, в зависимости от типа задачи оптимизации (максимизации или минимизации) будет вычисляться среднее расстояние между полученным набором решений в результате работы алгоритма и заданным недостижимым фронтом. Что позволит отслеживать приближение (или отдаление, в случае максимизации) получаемых решений-альтернатив и заданным недостижимым фронтом, и позволит оценивать пригодность каждого из индивидов-решений. Тут, конечно же, возникает вопрос о том, почему решения не скатятся в изначально заданный фронт. Этот факт объясняется тем, что переменные задачи ограничены своей областью определения и не могут выйти за неё, тем самым, исключая такую возможность.

Схема разработанных алгоритмов (*MGAGD, MGAIGD*)

Для работы алгоритма входными данными будут являться:

1.Размер популяции

2.Максимальное число поколений

3.Вероятность скрещивания

4.Вероятность мутации

5.Фронт недостижимых решений

Результат:

Недоминируемое множество решений

Шаги алгоритма:

1) Создание начальной популяции.
2) Перевод хромосомы в вещественные переменные, расчет значения критериев задачи.
3) Отбор недоминируемых индивидов.

4) Расчет значения метрики *GD(IGD)* для каждого индивида на основе критериев.
5) Расчет пригодности каждого индивида за счет значения метрики *GD(IGD).*
6) Селекция.
7) Скрещивание.
8) Мутация.
9) Ранее отобранные недоминируемые индивиды помещаются в новую популяцию, и алгоритм запускается заново.

2.4 Исследование модифицированного генетического алгоритма многокритериальной оптимизации

Исследование предложенного подхода проводилось на множестве тестовых задач из [122].

Задача №1:

Два объекта для минимизации

$$f_1 = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2$$

$$f_2 = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2} y_j^2$$

Где J_1 = *{j|j четно и 2 ≤ j ≤ n} и J2 = {j|j нечетно и 2 ≤ j ≤ n}*, и

$$y_j = \begin{cases} x_j - [0.3x_1^2\cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1]\cos(6\pi x_1 + \frac{j\pi}{n}) & j \in J_1 \\ x_j - [0.3x_1^2\cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1]\sin(6\pi x_1 + \frac{j\pi}{n}) & j \in J_2 \end{cases}$$

Парето Фронт:

$$f_2 = 1 - \sqrt{f_1}, \quad 0 \le f_1 \le 1$$

Множество Парето:

$$x_j = \begin{cases} \{0.3x_1^2 \cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1\} \cos(6\pi x_1 + \frac{j\pi}{n}) & j \in J_1 \\ \{0.3x_1^2 \cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1\} \sin(6\pi x_1 + \frac{j\pi}{n}) & j \in J_2 \end{cases}$$

$$0 \le x_1 \le 1.$$

Пространство поиска решений: $[0,1] \times [-1,1]^{n-1}$

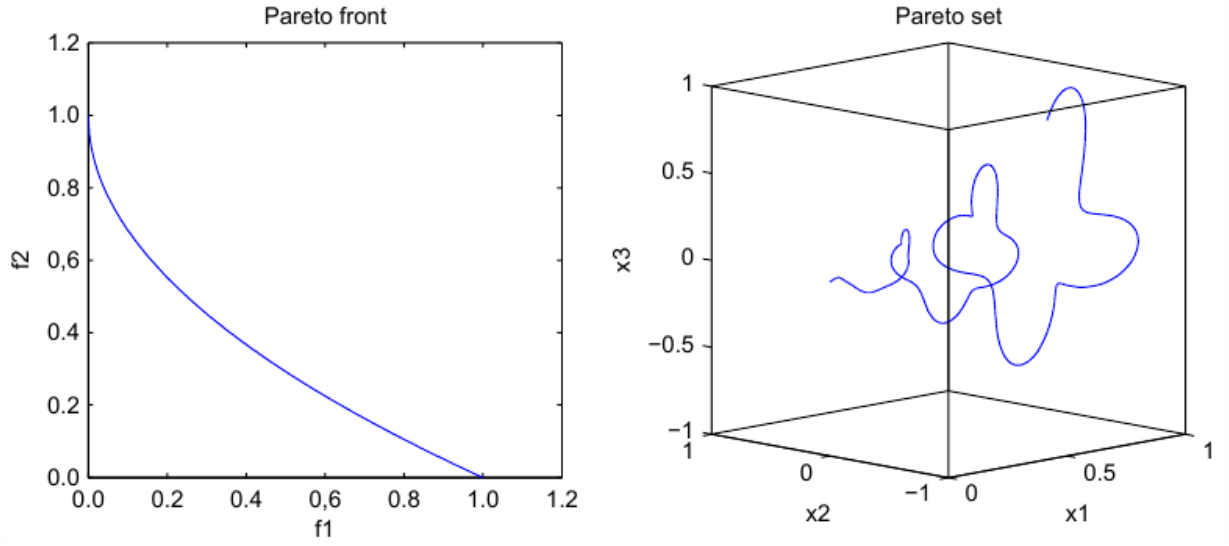


Рисунок 2.6 - Отображение истинного Фронта Парето и Множества Парето для задачи 1

Задача №2:

Два объекта для минимизации:

$$f_1 = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} h(y_j)$$

$$f_2 = 1 - x_1^2 + \frac{2}{|J_2|} \sum_{j \in J_2} h(y_j)$$

Где $J_1 = \{j | j$ *четно и* $2 \le j \le n\}$ *и* $J_2 = \{j | j$ *нечетно и* $2 \le j \le n\}$,

$$y_i = x_j - \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2, \ldots, n. \qquad h(t) = \frac{|t|}{1 + e^{2|t|}}.$$

,

Пространство поиска решений: $[0,1] \times [-2,2]^{n-1}$

Фронт Парето:

$$f_2 = 1 - f_2^2, \quad 0 \le f_1 \le 1.$$

Множество Парето:

$$x_j = \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2, \ldots, n. \quad 0 \le x_1 \le 1.$$

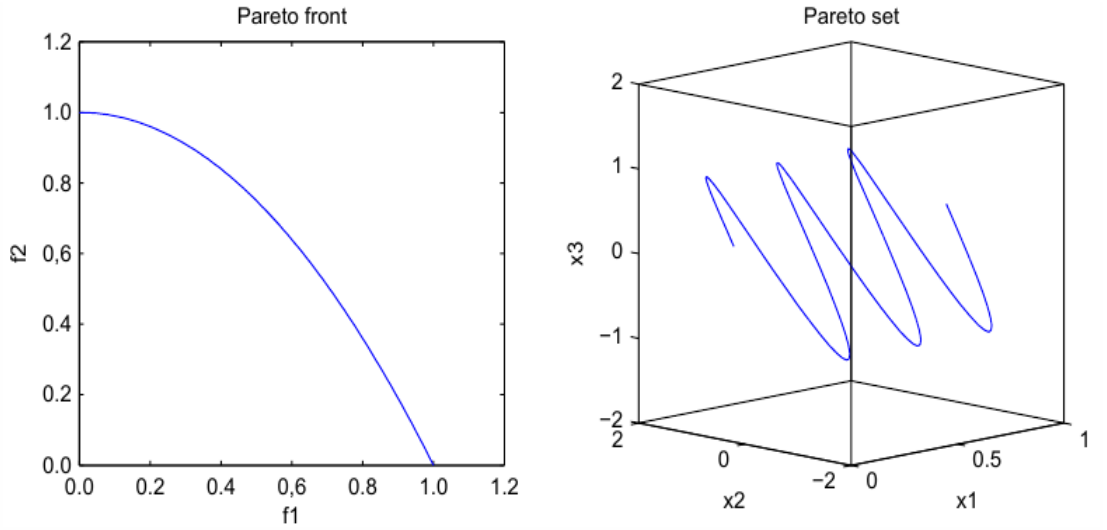


Рисунок 2.7 - Отображение истинного Фронта Парето и Множества Парето для задачи 2

Задача 3.

$I_1(\bar{x}) = (x_1 - 6)^2 + (x_2 - 4)^2 \to \min,$

$I_2(\bar{x}) = (x_1 + 2)^2 + (x_2 - 5)^2 \to \min.$

Условие принадлежности точки множеству Парето:

$(-2 \le x_1 \le 6) И (x_2 + 0.125 \cdot x_1 - 4.75 = 0)$ (множество Парето – отрезок, соединяющий точки (6;4) и (-2;5)).

Задача 4.

$I_1(\bar{x}) = (x_1 - 6)^2 + (x_2 - 4)^2 \to \min,$

$I_2(\bar{x}) = (x_1 + 2)^2 + (x_2 - 5)^2 \to \min,$

$I_3(\bar{x}) = (x_1 - 4)^2 + (x_2 + 4)^2 \to \min.$

Условие принадлежности точки множеству Парето:

$$(-0.125 \cdot x_1 - x_2 + 4.75 \geq 0) И (4 \cdot x_1 - x_1 - 20 \leq 0) И (-1.5 \cdot x_1 - x_2 + 2 \leq 0)$$

(множество Парето – треугольник с вершинами в точках (-2;5), (6;4) и (4;-4)).

Задача 5.

$$I_1(\bar{x}) = (x_1 - 1)^2 + (x_2 + 1)^2 \to \min,$$
$$I_2(\bar{x}) = (x_1 + 2)^2 + (x_2 - 2)^2 \to \min,$$
$$I_3(\bar{x}) = (x_1 - 3)^2 + (x_2 - 4)^2 \to \min,$$
$$I_4(\bar{x}) = (x_1 - 4)^2 + (x_2 - 2)^2 \to \min.$$

Условие принадлежности точки множеству Парето:

$$(0.4 \cdot x_1 - x_2 + 2.8 \geq 0) И (-2 \cdot x_1 - x_1 + 10 \geq 0) И (-x_1 - x_2 - 2 \leq 0) И (-x_1 - x_2 \leq 0)$$

(множество Парето – четырёхугольник с вершинами в точках (-2;2), (3;4), (4;2), (1;-1)).

Задачи условной многокритериальной оптимизации:

Задача 6.

$$I_1(\bar{x}) = (x_1 - 6)^2 + (x_2 - 4)^2 \to \min,$$
$$I_2(\bar{x}) = (x_1 + 2)^2 + (x_2 - 5)^2 \to \min,$$

$$\begin{cases} (x_1 - 1)^2 + (x_2 - 4)^2 \leq 4, \\ (x_1 - 3)^2 + (x_2 - 4)^2 \leq 6.25. \end{cases}$$

Условие принадлежности точки множеству Парето без учёта ограничений: $(-2 \leq x_1 \leq 6) И (x_2 + 0.125 \cdot x_1 - 4.75 = 0)$. Множество Парето и допустимая область пересекаются, за истинные решения принимаются допустимые решения из множества Парето.

Задача 7.

$$I_1(\bar{x}) = (x_1 - 6)^2 + (x_2 - 4)^2 \to \min,$$
$$I_2(\bar{x}) = (x_1 + 2)^2 + (x_2 - 5)^2 \to \min,$$
$$I_3(\bar{x}) = (x_1 - 4)^2 + (x_2 + 4)^2 \to \min,$$

$$\begin{cases}(x_1-2)^2+(x_2-2)^2\le 10.24,\\(x_1-5)^2+(x_2-1)^2\le 16,\\(x_1-3)^2+(x_2-4)^2\le 9.\end{cases}$$

Условие принадлежности точки множеству Парето без учёта ограничений: $(-0.125\cdot x_1-x_2+4.75\ge 0)И(4\cdot x_1-x_1-20\le 0)И(-1.5\cdot x_1-x_2+2\le 0)$.

Множество Парето и допустимая область пересекаются, за истинные решения принимаются допустимые решения из множества Парето.

Задача 8.

$$I_1(\bar{x})=(x_1-1)^2+(x_2+1)^2\to\min,$$
$$I_2(\bar{x})=(x_1+2)^2+(x_2-2)^2\to\min,$$
$$I_3(\bar{x})=(x_1-3)^2+(x_2-4)^2\to\min,$$
$$I_4(\bar{x})=(x_1-4)^2+(x_2-2)^2\to\min,$$

$$\begin{cases}(x_1+1.8)^2+(x_2-2)^2\ge 4,\\(x_1-1)^2+(x_2-4.5)^2\ge 4,\\(x_1-5.2)^2+(x_2-2)^2\ge 9,\\(x_1-1)^2+(x_2+2)^2\ge 9,\\(x_1-1)^2+(x_2-2)^2\le 6.25.\end{cases}$$

Условие принадлежности точки множеству Парето без учёта ограничений:

$$(0.4\cdot x_1-x_2+2.8\ge 0)И(-2\cdot x_1-x_1+10\ge 0)И(-x_1-x_2-2\le 0)И(-x_1-x_2\le 0).$$

Множество Парето и допустимая область пересекаются, за истинные решения принимаются допустимые решения из множества Парето.

Задача 9.

$$I_1(\bar{x})=(x_1-1)^2+(x_2+1)^2\to\min,$$
$$I_2(\bar{x})=(x_1+2)^2+(x_2-2)^2\to\min,$$
$$I_3(\bar{x})=(x_1-3)^2+(x_2-4)^2\to\min,$$
$$I_4(\bar{x})=(x_1-4)^2+(x_2-2)^2\to\min,$$

$$\begin{cases} x_1^{\,2} + (x_2 - 6)^2 \le 4, \\ (x_1 + 2)^2 + (x_2 - 5)^2 \le 4. \end{cases}$$

Условие принадлежности точки множеству Парето без учёта ограничений:

$$(0.4 \cdot x_1 - x_2 + 2.8 \ge 0) И (-2 \cdot x_1 - x_1 + 10 \ge 0) И (-x_1 - x_2 - 2 \le 0) И (-x_1 - x_2 \le 0).$$

Множество Парето и допустимая область не пересекаются. Условие принадлежности точки множеству Парето с учетом ограничений:

$$(\sqrt{x_1^{\,2} + (x_2 - 6)^2} = 2) И (-0.95 \le x_1 \le 0) И (4 \le x_2 \le 4.17)$$ (дуга окружности).

Задачи решались стандартными алгоритмами *SPEA* и *SPEA2*, а также предложенными модифицированными алгоритмами с метриками измерения расстояний до известного фронта Парето *GD* и *GDI*. Результаты работы алгоритмов усреднялись по 100 запускам. Для генетических алгоритмов была проведена серия экспериментов с варьированием следующих параметров ГА:

Тип скрещивания: одноточечное, двухточечное, равномерное.

Тип мутации: низкая, средняя, высокая.

Размер популяции: 50, 100 и 200 индивидов.

Количество поколений: 100, 150, 200, 250 и 300.

Последний параметр (количество поколений) в данном исследовании представляет особую важность, так как при стремлении алгоритма к недостижимому решению остановка алгоритма из-за обнаружения искомого решения невозможна.

Ниже приведены результаты для 4 вариантов настроек генетических алгоритмов, показавших наилучшие результаты. Алгоритмы *SPEA* и *SPEA2* работали со стандартным оператором оценки решений и только для финальной популяции проводилась их оценка по метрикам *GD* и *GDI*. Алгоритмы сравнивались по близости финальной популяции к известному фронту Парето. В таблицах и на графиках (рисунок 2.8-2.11) приведены абсолютные значения отклонений от фронта Парето для всех настроек всех

рассмотренных алгоритмов. Для иллюстрации приведены результаты решения для задач 1 и 2, а на остальных тестовых задачах получалась аналогичная картина.

Первый вариант настройки:

Тип селекции – турнирный (количество особей в подгруппе – 2), тип скрещивания – одноточечное, тип мутации – средний, размер популяции – 200, количество поколений – 250. В таблице 2.1 приведены полученные результаты оценки решения задач.

Второй вариант настройки:

Тип селекции – турнирный (количество особей в подгруппе – 2), тип скрещивания – одноточечное, тип мутации – высокий (50%), размер популяции – 200, количество поколений – 250. В таблице 2.2 приведены полученные результаты оценки решения задач.

Третий вариант настройки:

Тип селекции – турнирный (количество особей в подгруппе – 2), тип скрещивания – одноточечное, тип мутации – средний, размер популяции – 200, количество поколений – 300. В таблице 2.3 приведены полученные результаты оценки решения задач.

Четвертый вариант настройки:

Тип селекции – турнирный (количество особей в подгруппе – 2), тип скрещивания – одноточечное, тип мутации – высокий (50%), размер популяции – 200, количество поколений – 300, заданная точность решения 0,01. В таблице 2.4 приведены полученные результаты оценки решения задач.

Результаты решения Задачи №1 с первым и вторым вариантом настроек приведены ниже в виде таблицы и графика. Запись MGAGD(50) (и аналогичные) означает, что использовалась высокая мутация (50%).

Таблица 2.1 - Результаты оценки решения Задачи №1 многокритериальной оптимизации генетическими алгоритмами

Метод	**Среднее значение метрики GD**	**Среднее значение метрики IGD**
SPEA	0,070954	0,094949
SPEA(50)	0,128017	0,137606
SPEA2	0,078932	0,099997
SPEA2(50)	0,125661	0,133616
MGAGD	0,05858	0,080806
MGAGD(50)	0,083956	0,108438
MGAIGD	0,040704	0,118091
MGAIGD(50)	0,209114	0,392589

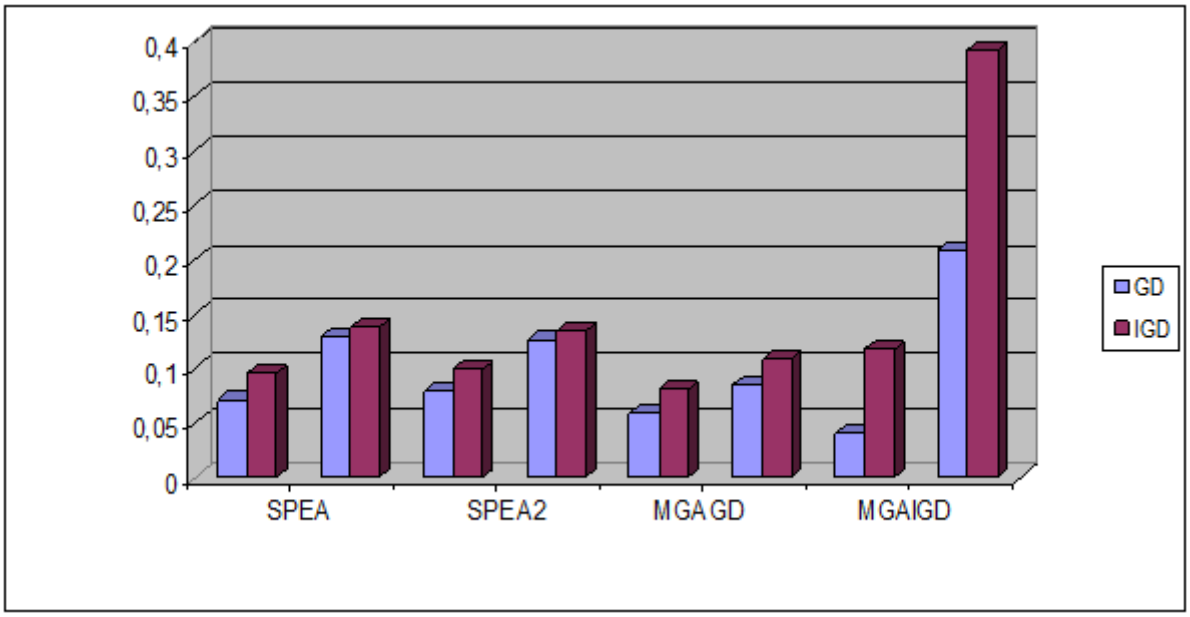


Рисунок 2.8 – Сравнение результатов оценки решения Задачи №1 алгоритмами MGAGD, MGAIGD, SPEA и SPEA2 (результат идет соответственно порядку методов таблицы 2.1)

Результаты решения Задачи №1 с третьим и четвертым вариантом настроек приведены ниже в виде таблицы и графика. Запись MGAGD(50) (и аналогичные) означает, что использовалась высокая мутация (50%).

Таблица 2.2 - Результаты оценки решения Задачи №1 многокритериальной оптимизации генетическими алгоритмами

Метод	**Среднее значение метрики GD**	**Среднее значение метрики IGD**
SPEA	0,04562	0,07999
SPEA(50)	0,121712	0,138778
SPEA2	0,048417	0,084077

SPEA2(50)	0,12303	0,136073
MGAGD	0,042291	0,067307
MGAGD(50)	0,080361	0,106684
MGAIGD	0,03654	0,138022
MGAIGD(50)	0,210639	0,420412

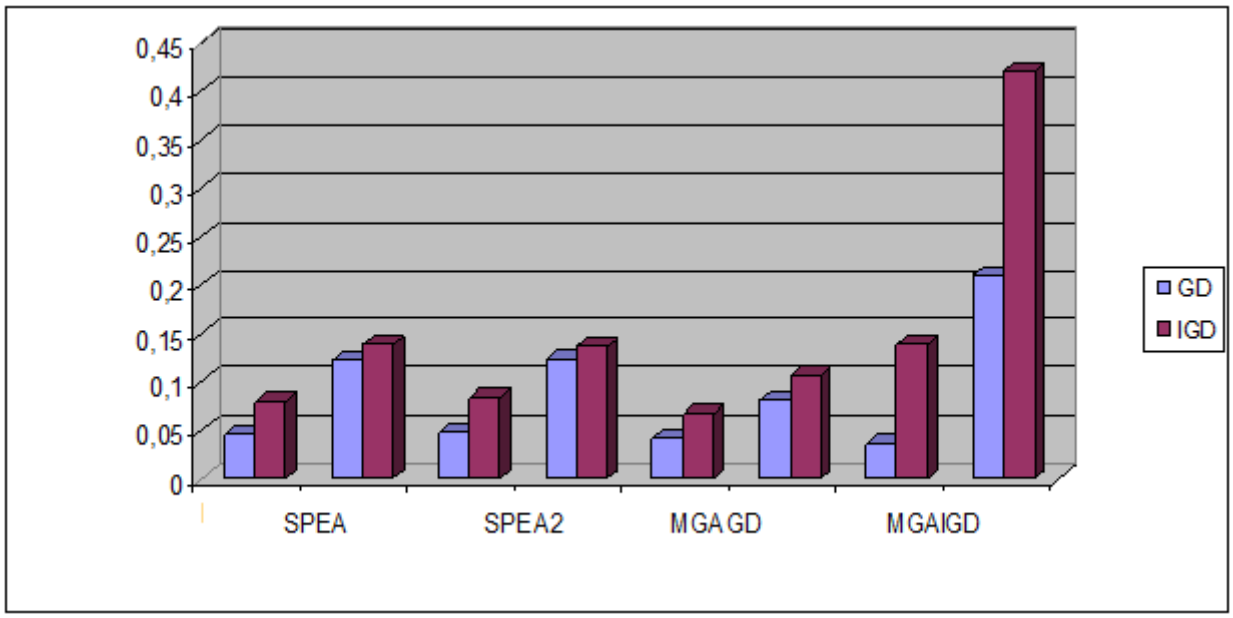


Рисунок 2.9 – Сравнение результатов оценки решения Задачи №1 алгоритмами MGAGD, MGAIGD, SPEA и SPEA2 (результат идет соответственно порядку методов таблицы 2.2)

Результаты решения Задачи №2 с первым и вторым вариантом настроек приведены ниже в виде таблицы и графика. Запись MGAGD(50) (и аналогичные) означает, что использовалась высокая мутация (50%).

Таблица 2.3 - Результаты оценки решения Задачи №2 многокритериальной оптимизации генетическими алгоритмами

Метод	**Среднее значение метрики GD**	**Среднее значение метрики IGD**
SPEA	0,097647	0,0999
SPEA(50)	0,115234	0,107027
SPEA2	0,095483	0,091091
SPEA2(50)	0,114601	0,107406
MGAGD	0,151753	0,081726
MGAGD(50)	0,086909	0,113686
MGAIGD	0,059735	0,076733
MGAIGD(50)	0,083329	0,24045

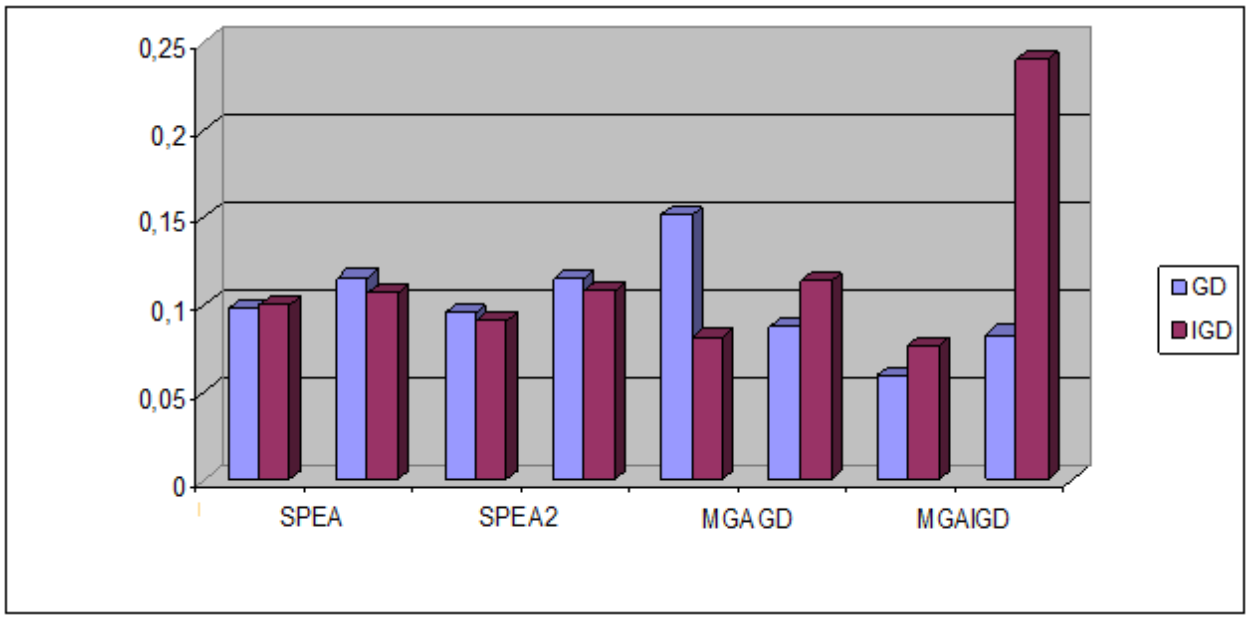


Рисунок 2.10 – Сравнение результатов оценки решения Задачи №2 алгоритмами MGAGD, MGAIGD, SPEA и SPEA2 (результат идет соответственно порядку методов таблицы 2.3)

Результаты решения Задачи №2 с третьим и четвертым вариантом настроек приведены ниже в виде таблицы и графика. Запись MGAGD(50) (и аналогичные) означает, что использовалась высокая мутация (50%).

Таблица 2.4 - Результаты оценки решения Задачи №2 многокритериальной оптимизации генетическими алгоритмами

Метод	**Среднее значение метрики GD**	**Среднее значение метрики IGD**
SPEA	0,020164	0,078912
SPEA(50)	0,119784	0,109578
SPEA2	0,081312	0,080166
SPEA2(50)	0,112059	0,10434
MGAGD	0,068538	0,067991
MGAGD(50)	0,075678	0,076008
MGAIGD	0,049283	0,092416
MGAIGD(50)	0,089233	0,31121

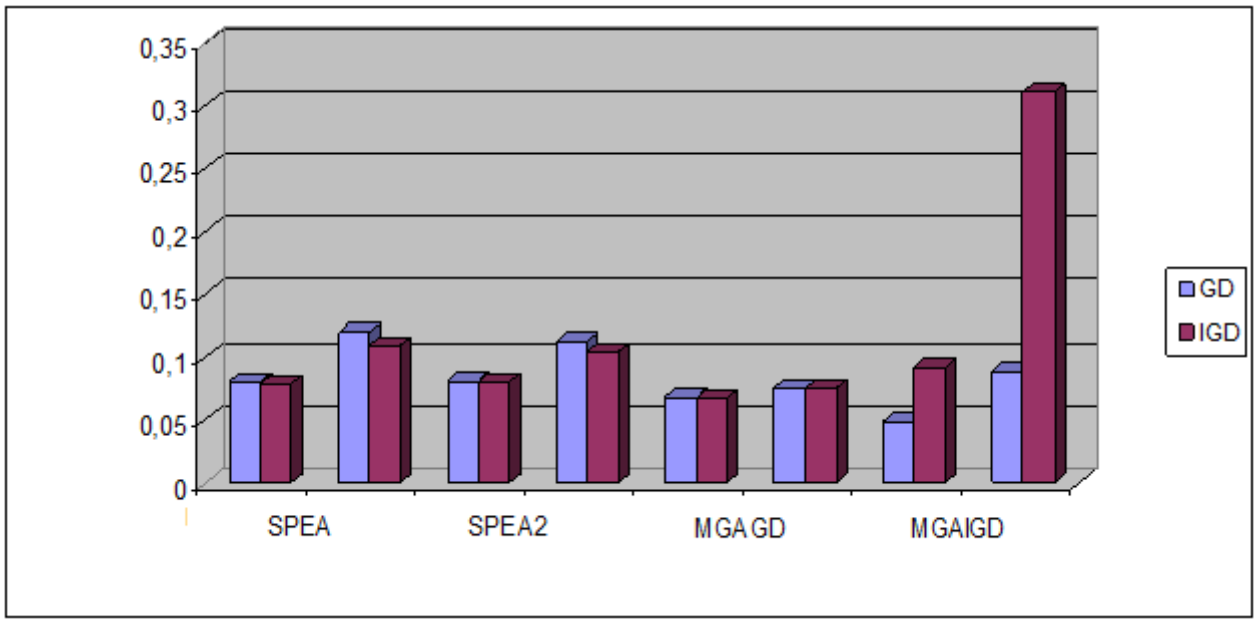


Рисунок 2.11 – Сравнение результатов оценки решения Задачи №2 алгоритмами MGAGD, MGAIGD, SPEA и SPEA2 (результат идет соответственно порядку методов таблицы 2.4)

По итогу исследования алгоритмов *SPEA*, *SPEA2* и разработанных *MGAGD* и *MGAIGD* были сделаны следующие выводы:

1. Предлагаемые алгоритмы показали свою эффективность решения задач многокритериальной оптимизации, так как его результаты их работы не уступают известным алгоритмам *SPEA* и *SPEA2*.
2. Наилучший результат на большинстве задач показал алгоритм *MGAGD*.

2.5 Разработка модифицированного генетического алгоритма для решения задачи выбора надежного варианта программного обеспечения

В качестве приемлемого средства решения задачи выбора надежного варианта программного обеспечения был выбран генетический алгоритм. Однако, для работы генетического алгоритма необходимо закодировать каждое решение в бинарную строку конечной длины – хромосому. Здесь

решение – это набор параметров, однозначно определяющий структуру ПО: N_j – количество компонентов на уровне j (целочисленное значение), Z_{ij} – множество версий компонента i, на уровне j (целочисленное значение), а также бинарные переменные B_{ij}, NVP_{ij}, RB_{ij}. При этом количество версий на уровне j заранее неизвестно и является одним из параметров задачи (N_j). Наш же алгоритм должен работать с вариантами ПО произвольной конфигурации. То есть, нам заранее не известно не только значение параметров ПО, но и их количество. Действительно, если в качестве одного из параметров, который должен настроить генетический алгоритм, выступает N_j – количество компонентов, то среди вариантов ПО, которые рассматривает алгоритм, могут оказаться такие, которые отличаются количеством компонентов, а значит и количеством наборов данных. Поэтому для решения задачи выбора надежного варианта ПО генетический алгоритм должен быть существенно модифицирован. Поэтому в диссертации рассмотрен модифицированный генетический алгоритм с переменной длиной хромосомы [49].

Впервые такой подход был рассмотрен в [46]. Главной особенностью данного подхода является необходимость сократить пространство поиска. Для задач проектирования, даже сравнительно не больших систем, поисковые пространства могут содержать 2^{100} и более решений. Происходит это от того, что в хромосоме ГА кодируются структура проектируемой системы избыточной сложности. Как правило, накладываемые ограничения не дают алгоритму выбирать наиболее сложную структуру, однако формально поисковое пространство при этом оказывается чрезмерно большим.

На этапе инициализации генетического алгоритма каждому решению случайным образом присваиваются значения N_j, определяющие количество компонентов на уровне j, $j \in \{1,..,M\}$, значение M – известная величина для каждого разрабатываемого ПО. Эти параметры N_j определяет длину хромосомы и одновременно выступают как переменные в работе

генетических операторов. На кодирование каждой переменной N_j в хромосоме выделяется по 3 гена, таким образом занимается $3M$ генов. Соответственно количество компонентов N_j может составлять от 1 до 8 штук для каждого M. Затем в хромосоме формируется блок содержащий информацию о множестве $\{Z_{ij}\}$ – множество версий компонента i, на уровне j. Число версий компонентов варьируется от 1 до 4 и занимает в хромосоме 2 гена. Там же кодируется информация о переменных B_{ij}, NVP_{ij}, RB_{ij}, для каждого блока j. На рисунке 2.12 представлен пример решения в генетическом алгоритме, для которого определены следующие параметры: количество архитектурных уровней M=2, количество компонент на 1-м уровне N_1=2, количество компонент на 2-м уровне N_2=4, для первой компоненты первого уровня количество версий $Z_{11}=4$, в программном компоненте не используется программная избыточность $B_{11}=0$, программном компоненте введена программная избыточность методом NVP NVP_{11}=1, в программном компоненте введена программная избыточность методом RB RB_{11}=1, и так далее.

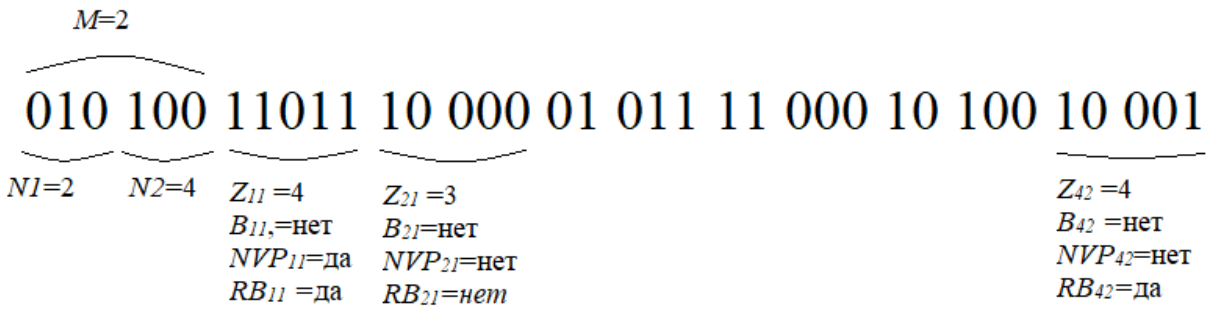


Рисунок 2.12 – Пример хромосомы в генетическом алгоритме

Для работы с хромосомами разной длины не могут быть использованы традиционные операторы генетического алгоритма. Для решения этой проблемы был предложен и реализован оригинальный оператор процентного скрещивания, позволяющий скрещивать хромосомы разной длины. Схема одноточечного процентного скрещивания для случая, когда в качестве точки

скрещивания выбирается точка – 25%, представлена на рисунке 2.13. Схема двухточечного процентного скрещивания представлена на рисунке 2.14, где в качестве точек скрещивания выбраны точки 25% и 70%.

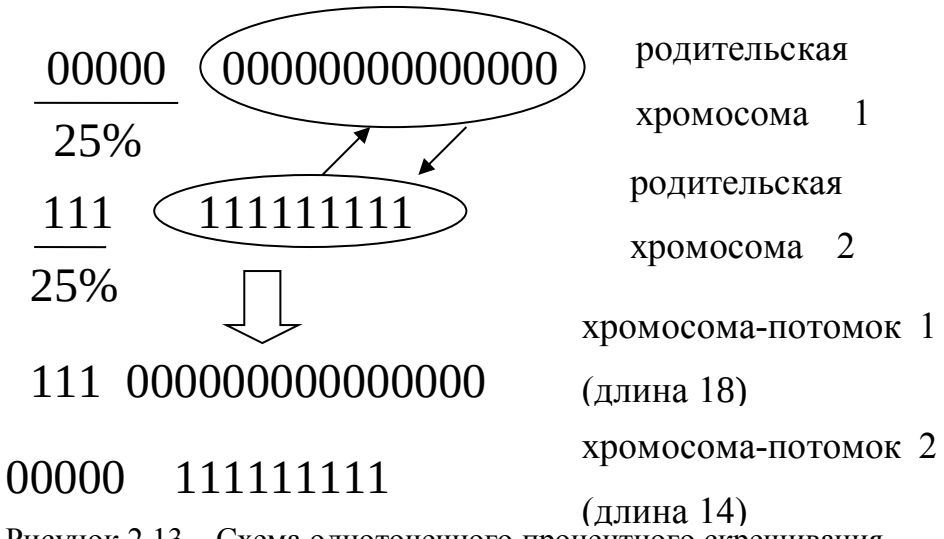


Рисунок 2.13 – Схема одноточечного процентного скрещивания.

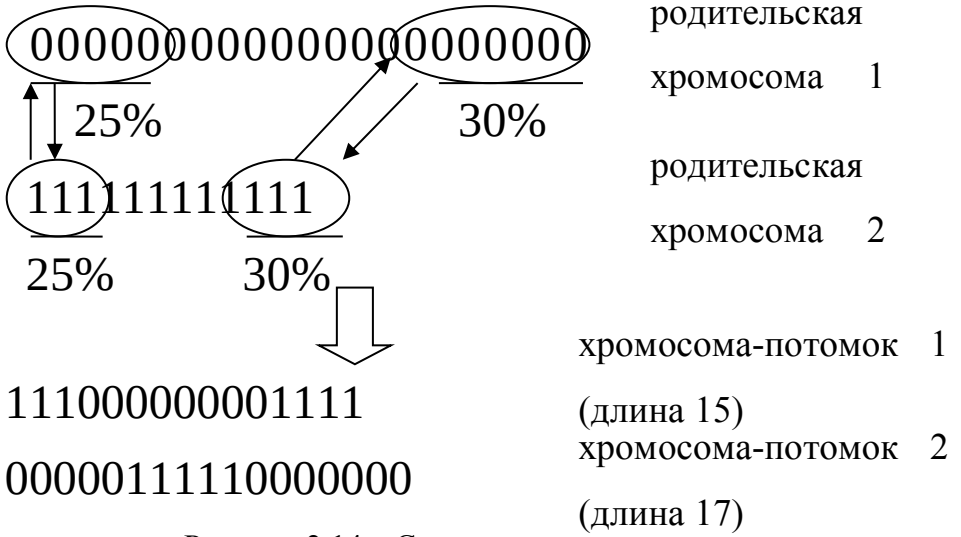


Рисунок 2.14 – Схема двухточечного процентного

Другой серьезной проблемой применения модификации ГА с переменной длинной хромосомы является специализация данного подхода под решение конкретной задачи. Так для данной модификации генетических алгоритмов нет признанных тестовых задач, поэтому проверку эффективности работы алгоритма проводят уже на реальных постановках.

Использование в модифицированном генетическом алгоритме хромосом переменной длины позволяет значительно сократить время вычисления оценок надежности ПО, а также позволяет алгоритму самому определять архитектуру ПО. Алгоритм с фиксированной длиной хромосом должен искать решение в пространстве размером 2^{90}, просматривая хромосомы с бÓльшим количеством неиспользуемых бит. Модифицированный генетический алгоритм, благодаря использованию хромосом переменной длины, ищет решение в пространстве размером примерно 2^{60}.

Для расчета обобщенного показателя надежности (2.2) необходимо получить значения оценок R_{ij}. Данный показатель рассчитывался алгоритмически, как результат моделирования ГЕРТ-сети, с соответствующим количеством N_j компонентов и количеством Z_{ij} версий программного модуля. Вычисление каждой оценки R_{ij} занимает достаточно длительное время, поэтому остро встал вопрос экономии вычислительного времени для работы алгоритма. Для решения задачи выбора надежного варианта программного обеспечения применялись предложенные в данной работе методы *MGAGD и MGAIGD,* которые показали наилучшие результаты при решении тестовых задач.

Выводы

Во второй главе была формализована задача проектирования надежного варианта программного обеспечения в виде задачи многокритериальной условной смешанной оптимизации. Были даны математические модели критериев стоимости разрабатываемого программного обеспечения и обобщенный критерий надежности ПО, учитывающий ошибки возникающие на этапе проектирования ПО и сбои происходящие во время функционирования ПО.

В качестве приемлемого средства решения задачи оптимизации был выбран генетический алгоритм. В главе приводится описание основных операторов генетического алгоритма и подходов к решению задач многокритериальной оптимизации ГА. В разделе 2.3 предложена оригинальная модификация генетического алгоритма для решения задач многокритериальной оптимизации. Проведены численные исследования предложенной модификации, продемонстрирована ее работоспособность. Отличие предлагаемой модификации (алгоритмы *MGAGD* и *MGAIGD*) отличаются от известных своей универсальностью, они спроектированы таким образом, что в результате своей работы гарантировано приближаются к фронту недоминируемых решений задачи (фронту Парето).

В разделе 2.5 предлагается оригинальная модификация генетического алгоритма, учитывающая специфику решаемой задачи. В предложенной модификации учитывается возможность включения в разрабатываемое программное обеспечение произвольного количество программных компонентов.

3 Проектирование надежного варианта программной системы

3.1 Описание проекта программной системы

На практическом уровне задача разработки программного обеспечения апробировалась на программном комплексе экспериментального образца протокола безопасного обмена данными (ЭО ПБОД), созданном в ходе исполнения соглашения о предоставлении субсидии от 27.11.2014 г. №14.574.21.0126 «Разработка протокола безопасного обмена данными в распределенной информационно-вычислительной системе на основе технологии защиты с использованием движущейся цели». Для рассмотрения требований обратимся к техническому заданию на разработку программного обеспечения, а также к характеристикам используемых протоколов и технологий.

Указано, что ЭО ПБОД должен обладать следующими характеристиками (таблица 3.1).

Таблица 3.1 – Характеристики ЭО ПБОД

Модуль	Функции	Параметры
1	2	3
1. Модуль аутентификации пользователя по логину и паролю	Предназначен для аутентификации пользователя с использованием логина (набора численных и буквенных символов) и пароля (набора численных, буквенных и специальных символов длиной не менее 6 символов). Пароль должен иметь возможность изменения по требованию администратора системы	ЭО ПБОД должен обеспечивать возможность одновременной работы не менее 3 пользователей. ЭО ПБОД должен обеспечивать возможность регистрации новых пользователей, назначения им ролей и отнесения к группам. ЭО ПБОД должен быть разработан по модульному принципу и обеспечивать возможность доступа через открытый программный интерфейс (API). ЭО ПБОД должен обеспечивать возможность отслеживания динамических характеристик распределенной информационно-
2. Модуль авторизации пользователя с использованием принадлежности пользователя к группам и его ролям в них	Предназначен для сопоставления прав доступа пользователей, указанных в матрице доступа ЭО ПБОД, и информации о прохождении аутентификации. В случае успешной аутентификации должны быть сопоставлены логин и соответствующие ему права доступа пользователей из матрицы доступа ЭО ПБОД	

		вычислительной системы, изменяемых в соответствии с технологией движущейся цели.
3. Модуль делегирования прав пользователя, полученных на основе аутентификации и авторизации, связанным с пользователем программным компонентам	Предназначен для сопоставления субъектов распределенной информационно-вычислительной системы, реализованных в виде программных компонентов, с конкретным пользователем и наделения указанных субъектов набором прав доступа из матрицы доступа, соответствующем логину пользователя	ЭО ПБОД должен обеспечивать работу с тремя и более ресурсами, причем в их число должны входить ресурсы по крайней мере двух типов. ЭО ПБОД должен быть реализован с использованием высокоуровневых языков программирования C/C++/C# и/или Python. Обмен данными между пользователями ЭО ПБОД должен осуществляться в формате JSON. Конфигурационные файлы должны быть в формате ключ-значение.
4. Модуль определения динамического адресного пространства в распределенной информационно-вычислительной системе	Предназначен для определения маршрутных характеристик передачи данных конкретного пользователя в начале и по завершению сеанса доступа пользователя	
5. Модуль защиты от исследования программных компонентов распределенной информационно-вычислительной системы	Предназначен для реализации модификации компонентов с использованием не менее чем двух внешних для программных компонентов параметров среды, при этом без нарушения функционала указанных компонентов и только для определенных заранее участков программного кода или параметров программного компонента распределенной информационно-вычислительной системы	

Как видно из таблицы 3.1, исходные данные предполагают существование программного обеспечения модульной структуры, обладающего сложным взаимодействием. Поскольку программное обеспечение является частью распределенной информационно-вычислительной системы, то к нему предъявляются высокие требования по надежности компонентов и связей, в том числе по предотвращению

существования скрытых ошибок и внезапных отказов, вызванных некорректной работой программного обеспечения.

Логически программа состоит из пяти модулей и двух хранилищ данных.

Первое хранилище данных содержит входные конфигурационные данные.

Хранилище данных №1 может быть организовано как комбинация из шести модулей:

1. Блок ввода данных
2. Блок проверки формата
3. База данных
4. Блок резервирования
5. Блок проверки резервной копии
6. Блок защиты резервной копии от несанкционированного доступа

Второе хранилище данных содержит итоговые сформированные пакеты данных.

Хранилище данных №2 может быть разбито на восемь модулей:

1. Блок передачи данных
2. Блок проверки формата
3. База данных
4. Система управления базами данных
5. Блок проверки резервной копии
6. Блок защиты резервной копии от несанкционированного доступа
7. Блок анализа итоговых пакетов
8. Блок формирования отчетов

Модуль аутентификации пользователя по логину и паролю. Предназначен для аутентификации пользователя с использованием логина (набора численных и буквенных символов) и пароля (набора численных, буквенных и специальных символов длиной не менее 6 символов) (модуль 1).

Модуль №1 разбит на шесть блоков:

1. Блок формирования логина
2. Блок проверки формата логина
3. Блок формирования пароля
4. Блок проверки формата пароля
5. Блок защиты пары логин-пароль от несанкционированного доступа
6. Блок хеширования

Модуль авторизации пользователя с использованием принадлежности пользователя к группам и его ролям в них. Предназначен для сопоставления прав доступа пользователей, указанных в матрице доступа ЭО ПБОД, и информации о прохождении аутентификации (модуль 2).

Модуль №2 разбит на шесть блоков:

1. Блок доступа к данным
2. Блок защиты данных о правах доступа от несанкционированного доступа
3. Входной интерфейс блока
4. Блок сопоставления данных
5. Блок формирования отчета
6. Блок доступа к базе входных данных

Модуль делегирования прав пользователя, полученных на основе аутентификации и авторизации, связанным с пользователем программным компонентам. Предназначен для сопоставления субъектов распределенной информационно-вычислительной системы, реализованных в виде программных компонентов, с конкретным пользователем и наделения указанных субъектов набором прав доступа из матрицы доступа, соответствующем логину пользователя (частично за счет реализованных в других модулях функций) (модуль 3).

Модуль №3 делится на шесть блоков:

1. Блок доступа к данным матрицы доступа
2. Блок доступа к базе входных данных

3. Блок защиты данных о правах доступа от несанкционированного доступа
4. Входной интерфейс блока
5. Блок сопоставления данных
6. Блок формирования отчета

Модуль определения динамического адресного пространства в распределенной информационно-вычислительной системе (модуль 4). Предназначен для определения маршрутных характеристик передачи данных конкретного пользователя в начале и по завершению сеанса доступа пользователя.

Модуль №4 разбит на семь компонентов:

1. Блок доступа к адресной информации
2. Блок доступа к базе входных данных
3. Входной интерфейс блока
4. Блок проверки формата
5. Блок расчета маршрутных характеристик
6. Блок сопоставления данных
7. Блок защиты адресной информации от несанкционированного доступа.

Модуль №5 разбит на семь компонентов:

1. Блок доступа к адресной информации
2. Блок доступа к базе входных данных
3. Входной интерфейс блока
4. Блок проверки формата
5. Блок модификации программного кода

Итого программный комплекс, исследуемый в работе, состоит из 5 базовых модулей и 2 хранилищ данных (рисунок 3.1), которые в свою очередь разбиты на 44 компонента, имеющих сложные внутренние связи и повышенные требования к надежности.

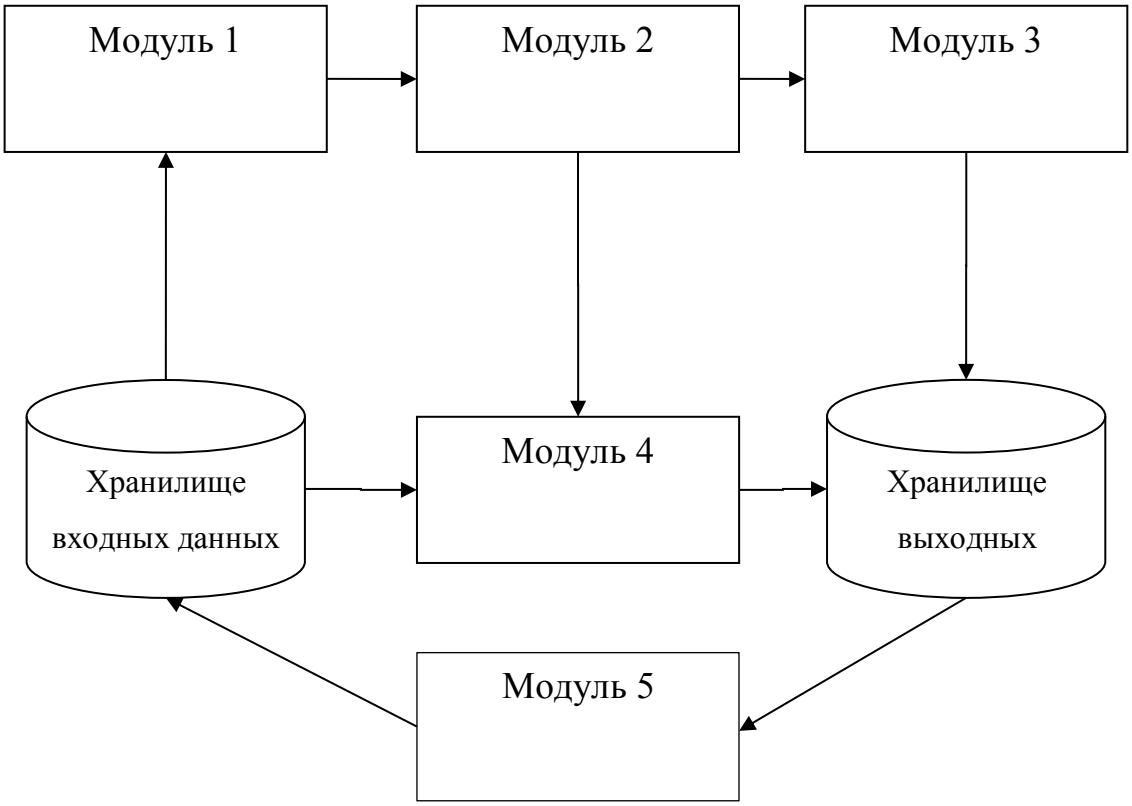


Рисунок 3.1 – Логическая структура программы

Для работы с группами пользователей, определения динамического адресного пространства и делегированием прав пользователя, полученных на основе аутентификации и авторизации, программное обеспечение использует протокол на основе VLAN (Virtual Local Area Network). Указанный протокол реализует виртуальные, объединенные логическими правилами построения каналов передачи данных группы устройств, имеющих возможность взаимодействовать между собой напрямую на канальном уровне, хотя физически при этом они могут быть подключены к разным сетевым коммутаторам. Полезным свойством протокола в данном случае является возможность разделять логически потоки данных между устройствами, находящимися в одной локальной сети, что дает возможность реализации свойств ЭО ПБОД, соответствующих функциям модулей 1-4 (таблица 3.1).

Также интересным свойством виртуальных логических сетей, соответствующих их назначению и использованном в разработке модуля 4 ЭО ПБОД, было отсутствие предварительной информации о структуре сети на ПЭВМ, представляющей собой конечный узел. Основное управление такой сетью осуществляется с помощью коммутаторов и иных сетевых устройств, что легко может быть подменено механизмами определения динамического адресного пространства, работающими по технологии ТДЦ [58].

При этом заказчик требовал, чтобы уровень показателей надежности программного обеспечения (в том числе наработки на отказ) позволял без ошибок и повторных передач проходить как минимум серии испытаний (Приложение А), а в эксплуатационном смысле постоянно функционировать с учетом оригинальной маршрутизации ПО, влияние которого на функционирование распределенной информационно-вычислительной сети в целом может быть критичным, но при этом не должна снижаться производительность, теряться функциональность и стоимость.

Итак, в качестве задачи оптимизации в данном случае применима многокритериальная задача оптимизации (раздел 2 настоящей работы), заключающаяся в обеспечении надежности модульной структуры ЭО ПБОД с учетом критерия стоимости разработки.

Далее коротко приведем данные об особенностях работы отдельных компонентов ЭО ПБОД, влияющих на расчет коэффициентов надежности модулей R_{ij}.

Такими особенностями, согласно протоколам испытаний (Приложение А) и описанию ЭО ПБОД, являются:

а) использование множественных VLAN. При ошибке конфигурирования или адресации комплекс в целом будет выведен из строя, критично взаимодействие модулей и корректная работа модулей 1-4;

б) разделение трафика и доступ к нему внутри сети, управление нагрузкой отдельных VLAN, рандомизация адресного пространства,

критична корректная работа модулей 1,4. При этом аутентификация пользователей, пригодная для реализации модулей 1-3, решается с применением протокол RADIUS (Remote Authentication Dial In User Service). Описан в стандартах RFC 2865 и RFC 2866. Протокол подразумевает взаимодействие клиент-сервер внутри распределенной информационно-вычислительной сети, что также накладывает ограничения по надежности ПО;

в) обмен информацией внутри указанных виртуальных локальных сетей выполнен с использованием групповой передачи и протокола управления групповой рассылкой в интернет. Таким образом, ошибки в работе модулей 2 и 4 могут напрямую отразиться на корректности функционирования всего комплекса, а также на отдельных показателях, требуемых техническим заданием на ЭО ПБОД;

г) для управления групповой рассылкой также может быть использован протокол *IGMP (Internet Group Management Protocol)* — протокол управления групповой (*multicast*) передачей данных в сетях, основанных на протоколе IP. IGMP используется маршрутизаторами и *IP*-узлами для организации сетевых устройств в группы. Этот протокол является частью спецификации групповой передачи пакетов в *IP*-сетях. *IGMP* расположен на сетевом уровне. Применение протокола управления групповой передачей данных в сети требует постоянной доступности ключевых узлов распределенной информационно-вычислительной сети;

Следовательно, основными критичными технологиями, реализованными ПО ЭО ПБОД и предъявляющими повышенные требования к надежности программного обеспечения, являются:

а) технология формирования динамического адресного пространства и рандомизации адреса узла, реализованная как децентрализованная и сеансовая;

б) технология формирования блоков программного обеспечения, заключающаяся в доработке алгоритма ПО без потери основных выполняемых функций;

в) технология аутентификации, совмещающая технологии распределенной аутентификации и выделения виртуальных локальных сетей в общей структуре распределенной информационно-вычислительной системы;

г) технология делегирования прав пользователей, учитывающая динамически формируемые виртуальные локальные сети и реализованная с учетом матрицы доступа ЭО ПБОД;

д) технология дополнительной идентификации пользователей за счет принадлежности к виртуальным локальным сетям, группам в групповой рассылке.

На схеме рисунка 3.2 показано общее взаимодействие в рамках ЭО ПБОД, представляющее корректный режим работы взаимодействующих модулей.

Результаты предварительного тестирования [116] показали, что использование подобной технологии увеличивает время сканирования сети для внешнего по отношению к защищаемому сегменту узла сети.

Очевидно, что подобное увеличение времени может сказаться и на накладных расходах времени передачи внутри самой распределенной информационно-вычислительной системе.

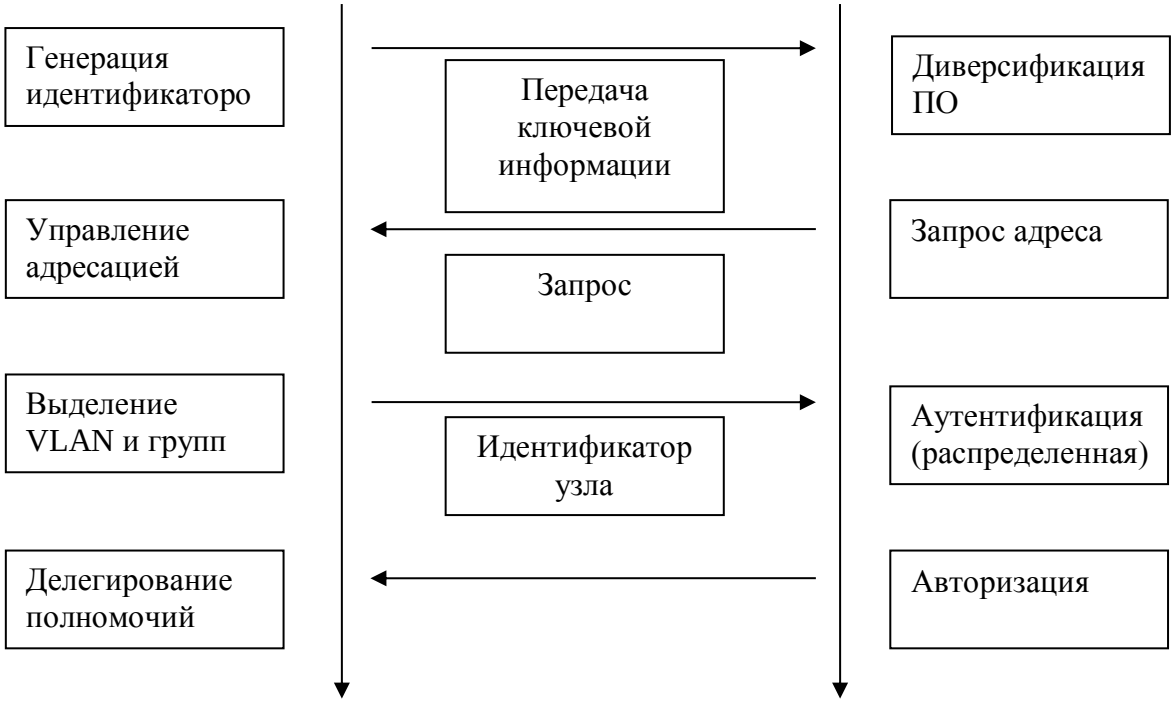


Рисунок 3.2 – Общая схема работы ЭО ПБОД

Для управления групповой рассылкой и протоколами на ее основе было разработано специальное приложение, соответствующее как техническому заданию в части требований к программному обеспечению, так и требованиям реализуемой технологии движущейся цели.

3.2 Экспериментальное исследование программной системы

В рамках проведения диссертационного исследования, а также проведения работ по четвертому этапу проекта «Разработка протокола безопасного обмена данными в распределенной информационно-вычислительной системе на основе технологии защиты с использованием движущейся цели» были проведены испытания экспериментального образца, составлены протоколы испытаний. Протоколы испытаний первых шести параметров экспериментального образца приведены в Приложении А.

Всего в ходе исследовательских испытаний было проверено 17 показателей, всесторонне демонстрирующих надежность выполнения ЭО ПБОД своих функций. В таблице ххх приведены исследуемые показатели, единицы их измерения, номинальные значения, которым они должны соответствовать и предельные отклонения от номинальных значений.

Таблица 3.2 - Определяемые показатели и точность их измерений

№	Наименование показателя	Ед. изм.	Номинальное значение	Предельные отклонения
1	Количество одновременно работающих пользователей	шт.	Не менее 3	0
2	Эффективность регистрации новых пользователей	Доля попыток успешной регистрации новых пользователей, %	100	5
3	Модульная структура ЭО ПБОД	Количество модулей, шт.	5	0
4	Эффективность отслеживания динамических характеристик системы	Доля ошибок при определении конечного адресата и передаче ему данных, %	5	5
5	Соответствие требования к техническим характеристикам	Доля выполненных требований, %	100	10
6	Состав ЭО ПБОД	Количество модулей, шт.	5	0
7	Корректность работы модуля аутентификации пользователей	Доля попыток успешной аутентификации, %	95	5
8	Корректность работы модуля авторизации пользователей	Доля неправильно авторизованных пользователей, %	0	0
9	Корректность работы модуля делегирования прав пользователю	Доля ошибочно назначенных прав пользователю, %	0	0
10	Корректность работы модуля определения динамического адресного пространства	Доля ошибок при определении конечного адресата и передаче ему данных, %	5	5
11	Пропускная способность	Кбит/с		
12	Задержка передачи	мс	10	30
13	Потеря/повторная передача данных	Частота потери/повторной передачи данных, %		
14	Количество модифицированного кода	% изменения сложности кода Количество внешних параметров, шт.	Не менее 2	

№	Наименование показателя	Ед. изм.	Номинальное значение	Предельные отклонения
15	Надежность пароля	Длина пароля, символов Хранение в защищенном виде	Не менее 6	
16	Количество хранимых данных	Количество групп, шт. Количество ролей, шт.	Не менее 2 Не менее 2	
17	Количество узлов	шт.	Не менее 3	0

Для показателей №3, 5 и 6 использовались номинальные значения и их нельзя использовать при моделировании функционирования программной системы. Оставшиеся 14 показателей были представлены в виде узлов ГЕРТ-сети моделирующей функционирование программных модулей ЭО ПБОД (см. рисунок 3.3).

При этом были получены следующие значения для представленных узлов сети (см. таблицу 3.3).

Таблица 3.3 - Моменты и производящие функции моментов

№	Тип распределения	$M_E(s)$	Математическое ожидание	Второй момент
1	Экспоненциальное	$\left(1-\frac{s}{a}\right)^{-1}$	3	0,16
2	Нормальное	$e^{sm+(1/2)s^2\sigma^2}$	100	0
4	Константа	n	5	0
7	Экспоненциальное	$\left(1-\frac{s}{a}\right)^{-1}$	95	10
8	Экспоненциальное	$\left(1-\frac{s}{a}\right)^{-1}$	0	0
9	Экспоненциальное	$\left(1-\frac{s}{a}\right)^{-1}$	0	0,1
10	Нормальное	$e^{sm+(1/2)s^2\sigma^2}$	5	0,05
11	Нормальное	$e^{sm+(1/2)s^2\sigma^2}$	1000	10
12	Нормальное	$e^{sm+(1/2)s^2\sigma^2}$	100	6

№	Тип распределения	$M_E(s)$	Математическое ожидание	Второй момент
13	Нормальное	$e^{sm+(1/2)s^2\sigma^2}$	1000	20
14	Нормальное	$e^{sm+(1/2)s^2\sigma^2}$	45	0,5
15	Нормальное	$e^{sm+(1/2)s^2\sigma^2}$	20	10
16	Константа	n	20	0
17	Экспоненциальное	$\left(1-\frac{s}{a}\right)^{-1}$	5	0,15

Данные 14 функциональных блоков входят в логическую структуру ЭО ПБОД (см. рисунок 3.3).

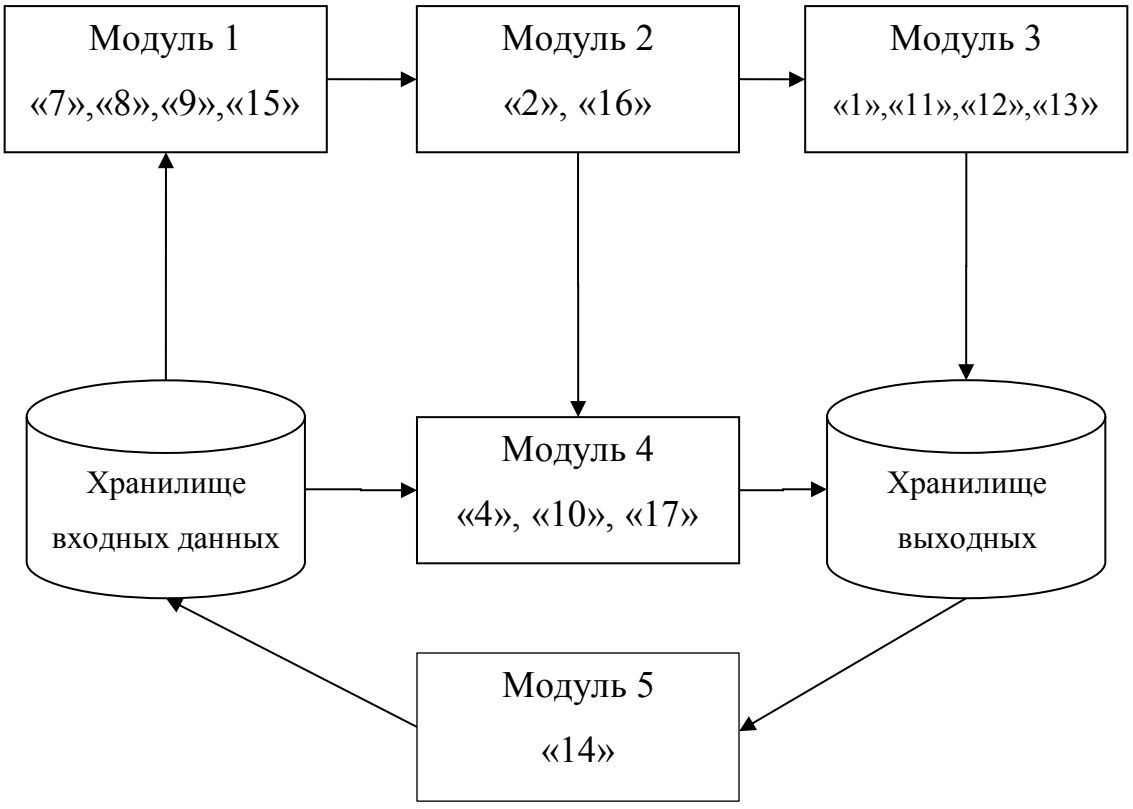


Рисунок 3.3 – Логическая структура программы со статистически оцененными компонентами

Используя алгоритм прямого и обратного расчета Герт-сети (см. раздел 1.4) были рассчитаны надежностные характеристики отдельных

программных модулей и ЭО ПБОД. Для этого процесс функционирования каждого модуля был представлен ГЕРТ-сетью. Для статистически оцененных компонентов программных модулей в алгоритме были использованы полученные функции плотностей распределения и их характеристики (см. таблицу 3.3), для остальных компонент были предложены экспертные оценки. Пример модели функционирования для программного модуля №5 представлен на рисунке 3.4.

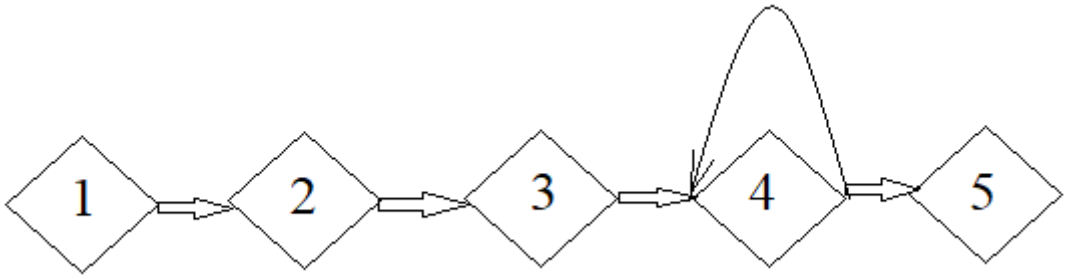


Рисунок 3.4 – Модель функционирования программного модуля №5 (1. Блок доступа к адресной информации, 2. Блок доступа к базе входных данных, 3. Входной интерфейс блока, 4. Блок проверки формата, 5. Блок модификации программного кода)

На рисунке 5й функциональный модуль программного модуля соответствует статистически рассчитанному в ходе проведенных испытаний показателю №14. При составлении топологического уравнения Мейсона [21] на месте узлов 1, 2, 3 и 4 были помещены константы, назначенные экспертом, за неимением статистически вычисленных оценок.

$$\mu_E = \frac{\partial}{\partial s}\left[\frac{1-0.95}{0.95\cdot 0.85\cdot 0.99\cdot e^{45s+0.5\cdot 0.5s^2}}\right]\Big|_{s=0}$$

Расчетное значение коэффициента надежности архитектуры ПО согласно методике, предложенной в п. 1.4 составило $R_5=0,7589$. Вычисление коэффициента надежности архитектуры ПО по формуле (1) для программного модуля №5 ($R=\sum_{i=1}^{N5} PU_{i5}\times R_{i5}$) дало результат R_5=0,75945375. Расхождение составило $E_5=0,00055375$. R_1=0,87256001, R_2=0,70564758,

R_3=0,90590065, R_4=0,65729975 Для оставшихся четырех программных модулей расхождения также составили незначительные величины (*E_1=0,00023374, E_2=0,00065225, E_3=0,00044870, E_4=0,00070024).* В дальнейшем будут использованы статистические данные для оценки надежности программный модулей.

По формуле (1.1) $R = \sum_{j=1}^{M} \sum_{i=1}^{Nj} PU_{ij} \times R_{ij}$, для которой M=5 – число программных модулей, *N1=6, N_2=6, N_3=6, N_4=7, N_5=5.* Был получен коэффициент надежности архитектуры ПО *R=0.62007891.* Трудозатраты для представленной программной архитектуры составили *T_s=92* человеко.часа.

Количество альтернативных вариантов программного комплекса, с учетом возможности введения множества версий для каждого функционального компонента, очень велико. Никакими переборными методами такая задача решена быть не может. В [123] была предпринята попытка использовать для проектирования информационных систем нейросетевой подход, однако в данном исследовании показана эффективность применения генетического алгоритма.

3.3 Проектирование надежного варианта программной системы

Полученные в предыдущем разделе значения надежности для разработанной программной системы являются недостаточными. Поэтому был применен разработанный в данном исследовании модифицированный генетический алгоритм для задач многокритериальной оптимизации, с целью увеличить показатели надежности программной системы и не превысить ограничения по трудозатратам для ее создания. Генетический алгоритм за счет применения различных вариантов обеспечения программной избыточности будет предлагать различные варианты надежной программной

архитектуры для обеспечения функционала, заложенного в данную программную систему.

При решении данной задачи было учтено то, что для программной системы уже создана архитектура, от которой достаточно сложно отказаться, в силу того, что функции, реализуемые в программной системе выполняются не автономно, а при участии и по инициативе пользователя. Поэтому в генетическом алгоритме в решения выбирались для зафиксированного количества уровней программной архитектуры M=5 и количества программных компонент на каждом уровне N_j. Таким образом генетический алгоритм решал задачу повышения надежности и снижения трудозатрат, выбирая количество версий для каждого программного компонента Z_{ij} и метод обеспечения программной избыточности.

Мощность пространства поиска составила $2^{175} \approx 4 \times 10^{52}$. Очевидно, что никакими переборными методами задачу такого объема решить невозможно.

Была выполнена программная реализация генетических алгоритмов MGAGD и MGAIGD в среде *Embarcadero C++Builder* 2010 [3].

В результате многократных запусков генетического алгоритма были получены альтернативные существующей архитектуры программной системы. В таблице 3.4 представлены значения характеристик надежности (коэффициент надежности архитектуры ПО *R)* и трудозатрат T_s. Решения алгоритма, представленные в таблице, являются элементами множества Парето (недоминируемых решений) данной задачи, аппроксимация которого была получена в ходе многократного ее решения алгоритмом. Наилучшие результаты показал алгоритм MGAGD с настройками: 200 индивидов в популяции, низкая мутация.

Таблица 3.4 - Характеристики оптимальных вариантов модернизации

№	**Коэффициент готовности системы**	**Трудозатраты на модернизацию, чел-час.**
1	*0.92007891*	489
2	*0.87100579*	367

3	*0.95411304*	523
4	*0.96486553*	597
5	*0.90338740*	454

Интерпретация полученных решений не представляет сложности. Так решению №4 из таблицы 3.4 соответствует программная архитектура со следующими характеристиками (см таблицу 3.5).

Таблица 3.5 - Интерпретация решения генетического алгоритма

№ уровня программной архитектуры	№ компонента	Кол-во версий	Способ обеспечения избыточности
1	1 1	1	Нет
1	1 2	3	*RB*
1	1 3	3	*RB*
1	1 4	2	*NVP*
1	1 5	1	Нет
1	1 6	1	Нет
2	2 1	2	*NVP*
2	2 2	3	*NVP*
2	2 3	4	*NVP*
2	2 4	2	*NVP*
2	2 5	2	*NVP*
2	2 6	1	Нет
3	3 1	1	Нет
3	3 2	2	*NVP*
3	3 3	3	*NVP*
3	3 4	3	*NVP*
3	3 5	3	*NVP*
3	3 6	1	Нет
4	4 1	1	Нет
4	4 2	3	*NVP*
4	4 3	4	*RB*
4	4 4	4	*NVP*
4	4 5	2	*NVP*
4	4 6	2	*NVP*
4	4 7	1	Нет
5	5 1	1	Нет
5	5 2	4	*NVP*
5	5 3	4	*RB*
5	5 4	2	*RB*
5	5 5	3	*NVP*

Анализ полученных решений позволяет убедиться, что генетический алгоритм действительно находит решения, не только удовлетворяющие критерию надежности, но и уменьшающие трудозатраты. Так для варианта программной системы, представленного в виде структуры в таблице 3.5, отсутствуют способы обеспечения избыточности для компонентов, которые носят детерминированных характер. То есть для тех программных модулей, для которых коэффициент надежности архитектуры, заданный экспертом или полученных в результате анализа статистических данных, R_{ij}=1.

Выводы

В главе был рассмотрен процесс проектирования реальной информационной системы с позиций подходов и методов, предложенных в данной диссертации. В результате была не только составлена модель процесса функционирования разрабатываемого программного комплекса, но и предложены варианты его архитектуры, значительно повышающие его надежность. Для достижения данного результата были решены следующие задачи:

- Составлена структурная модель разрабатываемого программного комплекса;
- Составлена модель состава функциональных блоков для программных модулей разрабатываемого комплекса;
- Проведены исследовательские испытания в соответствии с программной испытаний, в результате чего были получены объективные данные о результатах работы отдельных функциональных блоков разрабатываемого комплекса;
- Проведена статистическая оценка результатов проведенных экспериментов;

- Рассчитаны коэффициенты надежности архитектуры тех функциональных компонентов программного комплекса, с которыми происходили испытания;
- По рассчитанным оценкам и экспертным оценкам надежности (для оставшихся функциональных компонентов) по формуле (1) были рассчитаны значения коэффициенты надежности архитектуры для отдельных программных модулей и всего разрабатываемого комплекса в целом.
- Достоверность расчетов была подтверждена оценками, полученными в результате моделирования отдельных программных модулей в виде ГЕРТ-сети;
- Для повышения надежности программного комплекса был применен разработанный генетический алгоритм;
- В результате было получено множество альтернативных вариантов архитектуры разрабатываемого программного комплекса, отличающихся от исходного высокими значениями коэффициентов надежности архитектуры.

ЗАКЛЮЧЕНИЕ

В процессе диссертационного исследования были получены следующие результаты:

1. Был проведен анализ подходов оценки надежности программного обеспечения;

2. Предложена модифицированная модель оценки надежности функционирования программной системы, учитывающая как экспертные оценки надежности отдельных компонентов, так и оценки, полученные в результате статистического анализа экспериментальных данных;

3. Модифицирована модель оценки надежности функционирования программной системы на основе оценок надежности функционирования ее компонентов;

4. Разработаны стохастические алгоритмы многокритериальной условной оптимизации, позволяющие находить надежные варианты архитектуры ПО с учетом трудозатрат на реализацию ПО и возможностью использования различных подходов к обеспечению отказоустойчивости;

5. Предложена новая схема оценивания решений в многокритериальном генетическом алгоритме, отличающаяся от известных учетом одновременно всего множества критериев и позволяющая избегать преждевременной сходимости алгоритма;

6. Разработанные модели и алгоритмы применены при решении реальных задач проектирования архитектуры ПО. Получены варианты архитектуры программного комплекса, отличающиеся повышенной надежностью.

Таким образом, в диссертации разработаны стохастические алгоритмы моделирования и оптимизации, позволяющие повысить обоснованность принятия решений при выборе надежных вариантов программного обеспечения, что имеет существенное значение для теории и практики разработки систем обработки информации.

СПИСОК ЛИТЕРАТУРЫ

1. Алимханов, А.М. Использование программной системы поддержки для повышения доступности ресурсов корпоративной СУБД / А.М. Алимханов, С.В. Савин, Р.В. Юнусов; Вестник НИИ СУВПТ: Сб. научн. трудов/ Под общей ред. профессора Н.В. Василенко; Красноярск: НИИ СУВПТ.- 2003. Выпуск 11.- С. 107-112
2. Алимханов, А.М. Мультиверсионное формирование программно-информационных технологий корпоративных интегрированных структур/ А.М. Алимханов, И.В. Ковалев, Р.В. Юнусов; Материалы V-й Всероссийской конференции с международным участием молодых ученых и аспирантов «Новые информационные технологии. Разработка и аспекты применения», 28 ноября 2002 г. Таганрог: ТРТУ. 2002. С. 162-164
3. Архангельский А.Я. Программирование в C++ builder - М.:ООО"Бином-Пресс", 2010 г. - 896с. (1230с.) 7-е изд.
4. Благодатских, В.А. Стандартизация разработки программных средств [Текст] / В.А. Благодатских, В.А. Волнин, К.Ф. Поскакалов ; под ред. О.С. Разумова. – М.: Финансы и статистика, 2006. – 288 с.
5. Богатырев, В.А. К повышению надежности вычислительных систем на основе динамического распределения функций / В.А. Богатырев // Изв. Вузов. Приборостроение. № 4, 1981.
6. Василенко, Н. В. Модели оценки надежности программного обеспечения / Н. В. Василенко, В. А. Макаров // Вестник Новгор. гос.ун-та. – 2004. – № 28. – С. 126 – 132.
7. Вихарев, С. М. Надежность автоматизированных систем :учеб. пособие / С. М. Вихарев, А. А. Баринов. – Кострома : Изд-во КГТУ, 2007. – 80 с. – ISBN 978-5-8285-0305-6.
8. Вороновский Г.К., и др. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности. Х.: ОСНОВА,

1997.

9. Головкин, Б.А. Расчет характеристик и планирование параллельных вычислительных процессов / М.: Радио и связь, 1983. – 272 с
10. Громов, Ю. Ю. Надёжность информационных систем : учеб. пособие / Ю. Ю. Громов [и др.]. – Тамбов : Изд-во ТГТУ, 2010. – 160 с. – ISBN 978-5-8265-0911-1.
11. Гудман, С. Введение в разработку и анализ алгоритмов / С. Гудман, С. Хидетниеми. – Пер. с англ. – М.: Мир, 1981. – 366 с
12. Гуменникова А.В., Адаптивные поисковые алгоритмы для решения сложных задач многокритериальной оптимизации – диссертация на соискание ученой степени кандидата технических наук. – М.,: Красноярск, 2006
13. Дегтерев, А.С. Архитектурная надежность программного обеспечения информационно-телекоммуникационных технологий / А.С. Дегтерев, М.А. Русаков// Сборник материалов всероссийской заочной электронной конференции «Приоритетные направления развития науки, технологий и техники» (Москва, март 2005 г.).– М.: РАЕН, 2005. С. 161-162.
14. Дилон, Б. Инженерные методы обеспечения надежности систем / Б. Дилон, И. Сингх. – М.: Мир, 1984. – 318 с.
15. Емельянов С.В., Ларичев О.И. Многокритериальные методы принятия решений. – М.: Знание, 1985. – 32 с. – (Новое в жизни, науке, технике. Сер. «Математика, кибернетика»; № 10).
16. Зайцева, Е. Н. Оценка вероятности отказа невосстанавливаемой системы с использованием методов алгебры логики / Е. Н. Зайцева, Ю. В. Поттосин // Информатика. – 2007. – № 2. – С. 77 – 85.
17. Ильенкова, С. Д. Управление качеством. Учебник [Текст] / С.Д. Ильенкова, Н.Д. Ильенкова, С.Ю. Ягудин и др.; Под ред. С.Д. Ильенковой. – М.: ЮНИТИ, 1998. – 198 с.
18. Калайда В.Т., Романенко В.В. Технология разработки программного

обеспечения: Учебное пособие. — Томск: Томский межвузовский центр дистанционного образования, 2007. — 257 с.

19. Канатников А.Н., Крищенко А.П. Аналитическая геометрия: Учеб. для вузов. 2-е изд. / Под ред. В.С. Зарубина, А.П. Крищенко. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2000. – 388 с.
20. Кини Р.Л., Райфа Х. Принятие решений при многих критериях: предпочтения и замещения: Пер. с англ./Под ред. И.Ф. Шахнова. – М.: Радио и связь, 1981. – 560 с. ил
21. Ковалев, И.В. Автоматизированные системы управления: Учебное пособие для лекционных занятий для студентов специальности 210200 «Автоматизация технологических процессов и производств», всех форм обучения / И.В. Ковалев, Г.В. Волкова – Красноярск: СибГТУ. – 2006. – 179с.
22. Ковалев, И.В. Многоатрибутивная модель формирования гарантоспособного набора проектов мультиверсионных программных систем / И.В. Ковалев, Р.Ю. Царев; Вестник НИИ СУВПТ. – Вып.7. – Красноярск: НИИ СУВПТ, 2001. – С. 129-137.
23. Ковалев И.В. Управление развитием надежных кластерных структур информационных систем / И. В. Ковалев, Н. Н. Джиоева, А. В. Прокопенко, Р. Ю. Царев //Программные продукты и системы. 2010. № 2 (90).С. 68–71.
24. Ковалев, И.В. Модели поддержки многоэтапного анализа надежности программного обеспечения автоматизированных систем управления/ И.В. Ковалев, Р.Ю. Царев, М.А. Русаков, М.Ю. Слободин// Проблемы машиностроения и автоматизации.– № 2, 2005.– С. 73-81.
25. Ковалев, И.В. Мультиверсионный метод повышения программной надежности информационно-телекоммуникационных технологий в корпоративных структурах / И.В. Ковалев, Р.В. Юнусов; Телекоммуникации и информатизация образования. 2003. №2, С. 50-55.
26. Ковалев, И.В. Надежностная оценка информационных технологий при

комплексной автоматизации управления предприятием/ И.В. Ковалев, М.А. Русаков, М.Ю. Слободин // Татищевские чтения: актуальные проблемы науки и практики: Сб. материалов международной научной конференции/ ВУиТ. – Тольятти, 2005. – С. 131-136.

27. Ковалев, И.В. Надежность архитектуры программного обеспечения телекоммуникационных технологий / И.В. Ковалев, Н.В. Василенко, Р.В. Юнусов; Международная научная конференция Telematica'2001, Санкт-Петербург, 2001– С. 23-24.
28. Ковалев, И.В. Оптимальное проектирование мультиверсионных систем управления / И.В. Ковалев, А.А. Попов, А.С. Привалов. – Доклады НТК с международным участием «Информационные технологии в инновационных проектах» . – Ижевск: ИжГТУ, 2000. – С. 24-29.
29. Липаев, В. В. Надежность программных средств / Липаев В. В. – М. : СИНТЕГ, 1998. – 232 с. – (Информатизация России на пороге XXI века). – ISBN 5-89638-008-9.
30. Липаев, В.В. Анализ и сокращение рисков проектов сложных программных средств. Москва: НПО Синтег, 2005.
31. Липаев, В.В. Методы обеспечение качества крупномасштабных программных средств. М.: РФФИ. СИНТЕГ, 2003.
32. Липаев, В.В. Системное проектирование сложных программных средств для информационных систем. Изд. Второе переработанное и дополненное. М.: СИНТЕГ, 2002.
33. Липаев, В.В. Сопровождение и управление конфигурацией сложных программных средств. – М.: СИНТЕГ, 2006, 372 с.
34. Липаев, В.В. Технико-экономическое обоснование проектов сложных программных средств. М.: СИНТЕГ, 2004.
35. Липаев, В.В. Экономика производства сложных программных продуктов. М.:СИНТЕГ, 2008. 432 стр.
36. Майерс, Г. Надежность программного обеспечения [Текст] / Г. Майерс; перевод с англ. Ю.Ю. Галимова; под ред. В.Ш. Кауфмана. – М.: Мир,

1980. – 360 с.

37. Мамиконов, А.Г. Типизация разработки модульных систем обработки данных / А.Г. Мамиконов, В.В. Кульба, С.А. Косяченко. – М.: Наука, 1989. – 165 с
38. Мамиконов, А.Г. Синтез оптимальных модульных систем обработки данных / А.Г. Мамиконов, В.В. Кульба. – М.: Наука, 1986
39. Матвеевский, В. Р. Надежность технических систем : учеб. пособие / В. Р. Матвеевский ; Моск. гос. ин-т электроники и математики. – М., 2002. – 113 с.
40. Михалевич, В.С. Вычислительные методы исследования и проектирования сложных систем / В.С. Михалевич, В.Л. Волкович. – Наука, 1982. – 286 с
41. Монахов, Ю. М. Функциональная устойчивость информационных систем. В 3 ч. Ч. 1. Надежность программного обеспечения : учеб. пособие / Ю. М. Монахов ; Владим. гос. ун-т. – Владимир : Изд-во Владим. гос. ун-та, 2011. – 60 с. – ISBN 978-5-9984-0189-3.
42. Морозов, В.А. Модификация алгоритма голосования NVP-CV для одного вида матриц согласования [Текст] / В.А. Морозов // Вестник СибГАУ. – 2007. – № 1. – С.81-87.
43. Новой, А.В., Система анализа архитектурной надежности программного обеспечения.: Дис. канд. техн. наук: Красноярск, 2011 – 131 с.
44. Орлов, С.А., Технологии разработки программного обеспечения: Учебник / С. Орлов. – СПб.: Питер, 2002. – 464 с
45. Осокин А.Н. Теория информации: учебное пособие / А.Н. Осокин, А.Н. Мальчуков; Томский политехнический университет.– Томск: Изд-во Томского политехнического университета, 2014. – 208 с.
46. Панфилов И.А. Модели и алгоритмы выбора эффективной конфигурации многопроцессорных систем обработки информации и

управления - диссертация на соискание ученой степени кандидата технических наук, Красноярск, 2006

47. Панфилова Т. А. Анализ вероятностно-временных характеристик отказоустойчивого программного обеспечения распре деленных вычислительных систем / Р. Ю. Царев, А. В. Штарик, Е. Н. Штарик, М. А. Кочергина, Т. А. Панфилова // Вестник СибГАУ. 2012. № 4 (44). С. 64–70.
48. Панфилова Т.А. Прямой и обратный алгоритм расчета стохастических сетей / Царев Р.Ю., Штарик А.В., Штарик Е.Н., Хасанова Е.Р., Панфилова Т.А. // Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева. 2013. № 1 (47). С. 91-96.
49. Панфилова Т.А. Формализация задачи выбора надежного варианта программного обеспечения /Панфилов И.А., Панфилова Т.А. // Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева. 2008. № 2 (19). С. 26-28.
50. Письман Д. М., Дегтерев А. С. GERT-сетевой анализ времени выполнения задачи на неспециализированном гетерогенном кластере // Фундаментальные исследования. 2005. № 4. С. 79–80.
51. Письман Д. М., Шабалин С. А. Алгоритм расчета модифицированной ГЕРТ-сети // Успехи соврем. естествознания. 2005. № 11. С. 36–37.
52. Подиновский В.В., Ногин В.Д. Парето-оптимальные решения многокритериальных задач. – М.: Наука. Главная редакция физико-математической литературы, 1982. – 256 с.
53. Рыжкин, А. А. Основы теории надежности : учеб. пособие / А. А. Рыжкин, Б. Н. Слюсарь, К. Г. Шучев. – Ростов н/Д : Издат. центр ДГТУ, 2002. – 182 с. – ISBN 5-7890-0209-9.
54. Семенкин Е.С., Семенкина О.Э., Коробейников С.П. Оптимизация технических систем. Учебное пособие. – Красноярск: СИБУП, 1996. 284 с.

55. Смагин, В. А. Расчет вероятностно-временных характеристик пребывания задач в сетевой модели массового обслуживания / В. А. Смагин, В. П. Бубнов, Г. В. Филимонихин // Известия вузов. Приборостроение. – 1989. – № 2. – Т. XXXII.
56. Смагин, В. А. Стратегия последовательных приближений для повышения надежности функционирования сложных программных комплексов / В. А. Смагин // Надежность и контроль качества. – 2001. – № 7. – С. 35 – 43.
57. Соммервилл, И. Инженерия программного обеспечения [Текст] / И. Соммервилл, 6-е издание. : пер. с англ. – М. : Издательский дом "Вильямс", 2002. 624 с.
58. Стюгин М.А., Паротькин Н.Ю., Золотарев В.В. Технология защиты информации с применением движущейся цели в задачах безопасной адресации узлов компьютерной сети / Информационная безопасность в свете стратегии «Казахстан-2050», Астана, 2015. – с. 320-324.
59. Тарасенко Ф.П. Прикладной системный анализ : учебное пособие / Ф.П. Тарасенко. — М. : КНОРУС, 2010. — 224 с.
60. Терсков, В.А, Цели и проблемы синтеза вычислительных систем, работающих в реальном масштабе времени. Международный симпозиум «Непараметрика 2000». – Красноярск: Сибирская аэрокосмическая академия, 2000. с. 63-70.
61. Фокс, Дж. Программное обеспечение и его разработка/ Пер. с англ. – Под ред. Д.Б. Подшивалова. – М.: Мир, 1985. – 268 с
62. Хорошевский, В.Г. Инженерный анализ функционирования вычислительных машин и систем / М.: Радио и связь, 1987. – 256 с.
63. Царев Р.Ю. Применение сетей петри при моделировании программ с блоком восстановления /Царев Р.Ю., Тынченко С.В., Гриценко С.Н.// Современные наукоемкие технологии. 2016. № 6-2. С. 310-314.
64. Царев Р.Ю. Оценка транзакционной надежности современных систем управления и обработки информации /Р. Ю. Царев, А. В. Штарик, Е. Н.

Штарик, О. И. Завьялова // Приборы и системы. Управление, контроль, диагностика. 2012. № 6. С. 29–32.

65. Царев, М. Ю., Царев Р. Ю., Шевчук С. Ф. Модификация ГЕРТ-сети для анализа временных характеристик сетевых моделей // Вестник СибГАУ. 2009. № 1 (22). Ч. 2. С. 74–78.
66. Черкесов, Г.Н. Надежность аппаратно-программных комплексов.– СПб.: Питер, 2005.
67. Черноруцкий, И.Г. Методы оптимизации и принятия решений: Учебное пособие / И.Г. Черноруцкий.– СПб: Лань, 2001.– 384 с.
68. Шабалин, Н. Г. Автоматизированная система управления качеством технологических процессов на железнодорожном транспорте (АСУ КТП). М.: Железнодорожные технологии, 2004. 348 с.
69. Шеенок, Д.А. Программная надежность автоматизированных систем управления технологическими процессами на железнодорожном транспорте // Транспортная инфраструктура Сибирского региона: Материалы второй межвузовской научно-практической конференции, 16-18 мая 2011 г. Иркутск. – том 1. – 2011. – С. 244-248.
70. Шеенок Д.А. Многокритериальная оптимизация отказоустойчивой программой архитектуры специализированными эволюционными алгоритмами - диссертация на соискание ученой степени кандидата технических наук : 05.13.01 / СибГАУ им. М.Ф. Решетнева. Красноярск, 2013.
71. Юнусов Р.В. Система модельно-алгоритмической поддержки многоэтапного анализа надежности программных средств - диссертация на соискание ученой степени кандидата технических наук / Красноярск, 2003.
72. Юнусов, Р.В. Анализ надежности аппаратно-программного информационно-управляющего комплекса / Вестник НИИ СУВПТ: Сб. научн. трудов/ Под общей ред. профессора Н.В. Василенко; Красноярск: НИИ СУВПТ.2003. Выпуск 11. С. 103-106

73. Юнусов, Р.В. Оценка надежности программного обеспечения клиент-сервер на примере комплексной системы управления предприятием «Галактика» / Вестник НИИСУВПТ; Красноярск: НИИСУВПТ.-2001.-Вып.7. С.107-112
74. Юнусов, Р.В. Оценка надежности и гарантоспособная модель архитектуры программного обеспечения / Вестник НИИСУВПТ; Красноярск: НИИСУВПТ.2001.Вып.8. С.194-208
75. Юнусов, Р.В. Моделирование программных архитектур автоматизированных систем управления / Управляющие и вычислительные системы. Новые технологии: Материалы всероссийской электронной научно-технической конференции. Вологда: ВоГТУ, 2001. С. 60-61
76. Юнусов, Р.В. Модель формирования гарантоспособной архитектуры программного обеспечения / Решетневские чтения: Тезисы докладов 4 Всероссийской Научно-практической конференции студентов, аспирантов и молодых специалистов. Красноярск: САА, 2000. С. 172-174
77. Andrews, J.D. Fault Tree and Markov Analysis Applied to Various Design Complexities [Электронный ресурс]. – URL: http://www.fault-tree.net/papers/ericson-fta-vs-markov.pdf (дата обращения: 10.08.2013).
78. Avizienis, A. The N-Version Approach to Fault-Tolerant Software [Text] / A. Avizienis // IEEE Trans. Soft. Eng. – 1985. – Vol. SE-11 (12). – P. 1511-1517.
79. Avzienis, A. Fundamental Concepts of Dependability [Text] / A. Avizienis, J.-C. Laprie and B. Randell // Research Report N01145, LASS-CNRS, April 2001.
80. Back, Hoffmeister, Schwefel. A Survey of Evolution Strategies. / Proc. 4th International Conf. on Genetic Algorithms, 1991.
81. Baluja S. The Equilibrium Genetic Algorithm and the Role of Crossover. 1993.

82. Beasley, Bull, Martin. An Overview of Genetic Algorithms: Part2, Research topics. / University Computing, 15(4), 1993. p. 170-181
83. Boehm, B.W. The COCOMO 2.0 Software Cost Estimation Model. – American Programmer. – 2000. – 586 p.
84. Boehm, B.W. Information Processing / B.W. Boehm, A.C. Haile. Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCPI – 1985), Vol. I: Highlights, Report SAMSO/XRS-71-1, U.S. Air Force Systems Command (NTIS: AD 900031L), Los Angeles, CA, 1985.
85. Boehm, B.W. Software engineering economics. – Prentice-Hall. – 1981. – 320 p.
86. Berman, O. Choosing an Optimal Set of Libraries / O. Berman, M. Cutler.; IEEE Transaction on reliability. Vol. 45, No 2, June 1996, Pp. 303-307.
87. Clasen, U. Eine Moeglichkeit der numerischen Behandlung von zeitlich-stochastischen Netzplaenen / In: "Operations Research Proceedings", Springer Verlag Berlin-Heidelberg, 1994. – P. 46-51
88. Cohon, J.: Multiobjective Programming and Planing, John Wiley, New York (1978).
89. De Jong K.A., Spears W.M. An Analysis of the Interacting Role of Population Size and Crossover in Genetic Algorithms.
90. El-Ghazali Talbi Metahheuristics from design to implementation – Wiley J. & Sons, 2009,
91. Fonseca C.M., Fleming P.J. Multiobjective optimization and mul-tiple constraint handling with evolutionary algorithms - Part II: Ap-plication example. Technical report 565, University of Sheffield, Sheffield, UK, January 1995.
92. Fonseca C.M., Fleming P.J. Multiobjective optimization and mul-tiple constraint handling with evolutionary algorithms - Part I: A unified formulation. Technical report 564, University of Sheffield, Sheffield, UK, January 1995.

93. Fourman, M.P. Compaction of symbolic layout using genetic algorithms. In J.J. Grefenstette (Ed.), Genetic Algorithms and Their Applications^ Proceedings of the First International Conference on Genetic Algorithms. Hillsdale, NJ^ Lawrence Erlbaum, 1985. P.p. 141-153.
94. Goldberg, D.E. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesly, 1989.
95. Grams T. The Poverty of Reliability Growth Models/ FastAbstract ISSRE Copyright 1999
96. Horn, J., Nafpliotis N., Goldberg D. E. A niched Pareto genetic al-gorithm for multiobjective optimization. In Proceedings of the First IEEE Conference on Evolutionary Computation, Vol. 1, Pisca-taway, 1994. - P. 82-87.
97. Hui-Qun, Z. A New Method for Estimating the Reliability of Software System Based on Components / Z. Hui-Qun, S. Jing, G. Yuan; FastAbstract ISSRE and Chillarege Corp. Copyright 2001
98. Kaszycki,. G. Using Process Metrics to Enhance Software Fault Prediction Models/ FastAbstract ISSRE Copyright 1999
99. Koski J., Oscyczka A. Multi-criteria Design Optimization. Springer-Verlag, 1990.
100. Kursawe F. Breeding ES - first results // Seminar "Evolutionary algorithms and their applications", 1996.
101. Lyu, M.R. Handbook of Software Reliability Engineering / Edited by Michael R. Lyu Published by IEEE Computer Society Press and McGraw-Hill Book Company, 1996, 819 p
102. Lyu, M.R. Software Fault Tolerance / Edited by Michael R. Lyu Published by John Wiley & Sons Ltd, 1996
103. Lyu, M. Handbook of Software Reliability Engineering [Text] | M. Lyu . – New York : McGraw-Hill and IEEE Computer Society Press. – 1966. – 851 P.
104. Misra, P. N. Software Reliability analysis [Text] / P.N. Misra // IBM

System Journal. – 1983. – Vol. 22. NO 3. – P.262-270.

105. Muhlenbein H., Voigt H.-M. Gene Pool Recombination in Genetic Algorithms. In Proc. Of the Metaheuristics Inter. Conf., 1995.

106. Pham, H. Handbook of reliability engineering [Text] / H. Pham. – London : Springer. – 2003. – 696 P.

107. Pham, H. System Software Reliability [Text] / H. Pham. – London : Springer. – 2006. – 456 P.

108. Randell, B. System Structure for Software Fault-Tolerance [Text] / B. Randell // IEEE Trans. Soft. Eng. – 1975. – Vol. SE-1. – P. 220-232.

109. Randell, B. The Evolution of the Recovery Block Concept [Text] / B. Randell, J. Xu : John Wiley & Sonc Ltd. – 1995. – P. 1-21.

110. Rosenberg, L. Software Metrics and Reliability / L. Rosenberg, T. Hammer, J. Shaw; Software reliability engineering was presented at the 9-th International Symposium, “Best Paper” Award, November, 1998

111. Schaffer, J. D. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algorithms and Their Applica-tions, Pittsburgh, PA, 1985. - P. 93-100.

112. Shooman, M. L. Reliability of Computer and Networks: Fault Tolerance, Analysis, and Design [Text] / M.L. Shooman – New York : John Wiley & Sonc ltd. – 2002. – 546 P.

113. Shooman, M.L.. Software Reliability for Use During Proposal and Early Design Stages / FastAbstract ISSRE Copyright 1999

114. Srinivas, Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. / Evolutionary Computation, vol. 2 (3), 1995.

115. Steuer, R.E.: Multiple Criteria Optimization. John Wiley, New York, (1986).

116. Styugin, M. Multilevel decentralized protection scheme based on moving targets / M. Styugin, N. Parotkin / International J. of Security and Its Applications, vol. 10, issue 1, pp. 45-54, 2016.

117. Torres-Pomales, W. Software Fault Tolerance: A Tutorial [Text] / W.

Torres-Pomales . – Virginia : CASI. – 2000. 66 p.

118. Whitley D. Modeling Hybrid Genetic Algorithms.(1995).

119. Zitzler E., Thiele L. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach // IEEE Transactions on Evolutionary Computation, 1999.

120. ГОСТ 27.002-89. Надежность в технике. Основные понятия. Термины и определения" (утв. Постановлением Госстандарта СССР от 15.11.1989 N 3375)

121. ГОСТ 28195-99 «Оценка качества программных средств» - Межгосударственный Совет по стандартизации, метрологии и сертификации, Минск.

122. University of Essex: сайт - М.: Информационный интернет-портал «www.essex.ac.uk», 2014. – URL: http://dces.essex.ac.uk/staff/zhang/MOEAcompetition/cec09testproblem0904.pdf.pdf

123. Панфилова Т.А. Система поддержки принятия решений на основе методов нейросетевого моделирования / Л.В. Липинский, С.Ю. Кузин, А.С. Егоров, Т.А. Панфилова, В.С. Тынченко. – М.: РОСПАТЕНТ.– 2009.–Свидетельство о государственной регистрации программы для ЭВМ № 2009611013 от 16.02.2009.

ПРИЛОЖЕНИЕ А. Протоколы экспериментальных исследований

ПРОТОКОЛ

экспериментальных исследований № _1_ *от 12.06.2016 г.*

1. Объект исследования: ЭО ПБОД в целом.

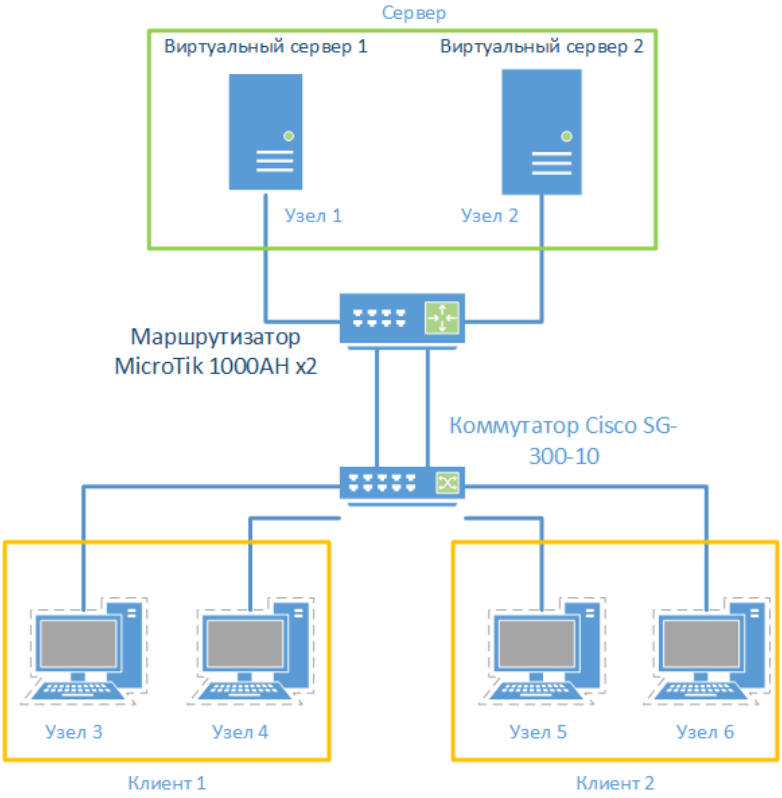
2. Цель исследования: проверка соответствия объекта исследования требованиям пункта № 4.2.1 ТЗ.

3. Дата начала исследования: 08.06.2016 г.

4. Дата окончания исследования: 12.06.2016 г.

6. Результаты исследования

Тестирование проводилось на скорости доступа портов 100 Мбит/с. Схема сети, используемая в тесте приведена на рисунке. В качестве сервера использовалось две виртуальных машины, у каждой из которых был отдельный сетевой интерфейс, одна из которых выполняла роль сервера (узел 1), другая клиента (узел 2). Аналогично на клиентских ПК запущены по 2 виртуальные машины с отдельными физическими сетевыми интерфейсами.



Тестирование проводилось на скорости доступа портов 100 Мбит/с согласно п. 6.2 методики исследований при этом число пар ограничивалось числом виртуальных узлов - 6, т.е. максимальное возможное количество одновременно взаимодействующих пользователей составляло 5 штук с учетом того, что 1 узел мог установить соединение для передачи данных одновременно с 2 другими узлами. В таблице приводятся результаты определения количества взаимодействующих узлов для 50 итераций теста.

№ итерации	Число узлов	№ итерации	Число узлов
1	5	26	5
2	5	27	5
3	5	28	5
4	5	29	5
5	5	30	5
6	5	31	5
7	5	32	5
8	3	33	5
9	5	34	5
10	5	35	5
11	5	36	5
12	5	37	5

13	5	38	5
14	5	39	3
15	5	40	5
16	5	41	5
17	5	42	5
18	5	43	5
19	5	44	5
20	5	45	5
21	5	46	4
22	2	47	5
23	5	48	5
24	5	49	5
25	5	50	5
		Среднее	**4,84**

7. Замечания и рекомендации

Отсутствуют

8. Выводы

По результатам экспериментальных исследований было установлено, что разработанная реализация ЭО ПБОД обеспечивает на приемлемом уровне одновременную работу в среднем 4,84 пользователей (п. 4.2.1), с обменом ресурсами двух разных типов (п. 4.3.1.1). Во время выполнения теста были найдены все требуемые адресаты и переданы им необходимые данные, поскольку в ЭО ПБОД отсутствует поле контроля целостности и данные повторно отправляются только в случае отсутствия подтверждения, то было 5 итераций с частичной потерей передаваемых данных, когда передаваемый файл разбивался на несколько UDP дейтаграмм, и они не все были доставлены по назначению. В результате полученный бинарный файл не соответствовал исходному файлу, что проявилось в изменении его хэша. Таким образом можно говорить о достижении задачи из п. 4.2.1 ТЗ «обеспечивать возможность одновременной работы не менее 3 пользователей».

ПРОТОКОЛ

экспериментальных исследований № _2_ *от 30.05.2016 г.*

1. Объект исследования: подсистема регистрации новых пользователей ЭО ПБОД.

2. Цель исследования: проверка соответствия объекта исследования требованиям пункта № 4.2.2 ТЗ.

3. Дата начала исследования: 27.05.2016 г.

4. Дата окончания исследования: 30.05.2016 г.

6. Результаты исследования

В соответствии с методом испытаний п. 6.3 были зарегистрированы через web-интерфейс 50 новых пользователей. Их учетные данные представлены в таблице. Для каждой записи случайно назначалась роль и группа. После чего проводилась попытка авторизации для данной учетной записи. Ее результат отражен в столбце «Проверка записи».

№ итерации	Логин	Хэш пароля	Роль	Группа	Проверка записи
1	user1	b3d79a7e2e767727285a524f8297dcbc	2	3	Успех
2	user2	d747e48825d8e7c9690529c6ab348697	2	2	Успех
3	user3	d58b6190494856ed0f7da1520324e88a	2	2	Успех
4	user4	eb67a603d4aeafb4516a42dfdc281840	3	2	Успех
5	user5	bd38afdcbc80f59b34b8910b6552e35a	3	2	Успех
6	user6	38ae3fdd219ad6b84605e8f4b77358c3	1	3	Успех
7	user7	ef3e8d141915e3945a12ea95129f04ed	2	1	Успех

№ итерации	Логин	Хэш пароля	Роль	Группа	Проверка записи
8	user8	a18b1077198f6cf1a43acd97131daf93	2	2	Успех
9	user9	da290b0c1f76f55ea8a6a0b2bc3bd394	3	3	Успех
10	user10	89675d75ecdf194fd8fa8126b8f9296d	2	3	Успех
11	user11	188b2b4e17403656ab28313407c3a616	3	3	Успех
12	user12	c97a404b3ba2d26a4d38b3b3724d79ab	1	3	Успех
13	user13	9dbe7df3a9721f46e4c8444e3abc96ec	2	3	Успех
14	user14	0d6f124ecdc7683422a5fcccb1517a0c	2	1	Успех
15	user15	279693457ea7a1873829ad1b72990045	1	2	Успех
16	user16	e72085541c03603d0103b91b322b466d	2	3	Успех
17	user17	09a68e912d595f62e563f853c421fe07	3	3	Успех
18	user18	757d6cf5beab916fe99712be0e89dd16	1	1	Успех
19	user19	8008673935852d30789f345703ac3fbb	2	3	Успех
20	user20	915b74cb2172a58c5a9d7139a64465ee	2	3	Успех
21	user21	85c038bd6206dc94f89624280eaa1d60	3	2	Успех
22	user22	5d0e7d17a9da2b9af95b4c70c4d28076	2	2	Успех
23	user23	c0eef316f0bb075fa01fd8fb89aa926f	3	3	Успех
24	user24	7e3010c1a8fb340b6c46ceb75dabb3ff	2	1	Успех
25	user25	3b46ac74c92fea0a9d812484f3581e9d	3	3	Успех
26	user26	8739d6aa27308d1da730e12ced740ad9	3	3	Успех
27	user27	06979296cb4e7cdd249aa296081b1ac6	2	3	Успех
28	user28	20842dc65fb1c992b7d63b7a06dcaafe	3	2	Успех
29	user29	92242f59a23794386093abb3b54911be	1	1	Успех
30	user30	492df2a1fe6c0954abc0cc5ff16eceec	1	3	Успех
31	user31	4314aa605a76b1652795901ce12ee0e8	2	1	Успех
32	user32	d13d1ce7714d9ca8fb05dbdf2db0fdfd	2	2	Успех
33	user33	5bd5519fdce8a409f5bdc796d4974b55	1	2	Успех
34	user34	61c9abbb4b58407091324b386c77bc7e	3	1	Успех
35	user35	205f77eb91fa6e56b6645ddef6d513ce	2	2	Успех
36	user36	93ae7e4f67b074720b7a4ffb34750a23	2	3	Успех
37	user37	1b612838bbc27d208160d85e56a1175a	3	1	Успех
38	user38	3cb7c74c499b3f43114614eaf9eaf226	1	2	Успех
39	user39	b892939b750c9914705e41cb1a909513	2	3	Успех
40	user40	9f10fc2dd65b5e3a923ffe8345c7e74a	2	3	Успех
41	user41	3e0f263580da34de39c17a068ee3a4a0	1	2	Успех
42	user42	8abcb6ffa3364600ae8bd18bd007527b	1	2	Успех
43	user43	a87147b49f3761488f195c879538d50e	2	2	Успех
44	user44	2650ed375dd8c79e7312b14fea6f18ca	1	2	Успех
45	user45	7a53d484715da1839844cb2c92a34177	2	2	Успех
46	user46	e29e54f39e62d9a27cb1511d0056d37c	1	1	Успех
47	user47	6a31391bba08349a0aa720507ea3611c	3	1	Успех
48	user48	4cc6a41a4a7777af74b229665c1b514e	3	3	Успех
49	user49	f2f7b279a54fa769a3a58dc835e8e33e	1	1	Успех
50	user50	e9534c1d1743795f381ff8ee7c9403e3	1	1	Успех
				Итого	**100%**

7. Замечания и рекомендации

Отсутствуют

8. Выводы

По результатам экспериментальных исследования было установлено, что разработанная реализация ЭО ПБОД 100% корректную регистрацию, сохранение и предоставление учетных данных новых пользователей. Более детальная проверка эффективности модулей авторизации/аутентификации должна быть проведена при одновременном обращении нескольких пользователей согласно пп. 4.7 – 4.9 методики исследований. Таким образом можно говорить о достижении задачи из п. 4.2.1 ТЗ «обеспечивать возможность регистрации новых пользователей, назначения им ролей и отнесения к группам».

ПРОТОКОЛ

экспериментальных исследований № _3_ *от 16.06.2016 г.*

1. Объект исследования: структура ЭО ПБОД.

2. Цель исследования: проверка соответствия объекта исследования требованиям пункта № 4.2.3 ТЗ.

3. Дата начала исследования: 13.06.2016 г.

4. Дата окончания исследования: 16.06.2016 г.

6. Результаты исследования

На основе анализа исходных кодов модулей ЭО ПБОД, предоставленных на регистрацию в Федеральную службу по интеллектуальной собственности (свидетельство о регистрации № 2016614108, Дата регистрации: 14.04.2016), и описания протокола ЭО ПБОД было установлено наличие совместно функционирующих 5 модулей, реализованных в виде отдельных классов и функций со стандартизированными форматами ввода-вывода данных:

1) модуль аутентификации пользователя по логину и паролю;

2) модуль авторизации пользователя с использованием принадлежности пользователя к группам и его ролям в них;

3) модуль делегирования прав пользователя, полученных на основе аутентификации и авторизации, связанным с пользователем программным компонентам;

4) модуль определения динамического адресного пространства в распределенной информационно-вычислительной системе;

5) модуль защиты от исследования программных компонентов распределенной информационно-вычислительной системы.

Результаты выполнения пунктов программы при единичном тестировании функционала модулей после их последовательной обфускации модулем защиты от исследований программных компонентов представлены в таблице.

п. программы / изменяемый модуль	Аутентификации	Авторизации	Делегирования прав	Определения адресного пространства	Защиты от исследований
4.7	Выполнено частично	Выполнено	Выполнено	Выполнено	Выполнено
4.8	Выполнено	Выполнено частично	Выполнено	Выполнено	Выполнено
4.9	Выполнено	Выполнено частично	Выполнено	Выполнено	Выполнено
4.10	Выполнено	Выполнено	Выполнено	Выполнено частично	Выполнено
4.14	Выполнено	Выполнено	Выполнено	Выполнено	Выполнено

Частичное выполнение пункта программы испытаний означает возникновение случаев превышение усредненного времени выполнения операций модулем, следовательно, и клиентским приложением, от усредненных показателей работы модулей в обычном режиме при проведении испытаний.

7. Замечания и рекомендации

Отсутствуют

8. Выводы

По результатам исследований было установлено наличие в ЭО ПБОД 5 взаимосвязанных модулей, взаимодействующих через стандартизированные программные интерфейсы. Изменение их внутренней структуры с сохранением форматов ввода-вывода не приводит к потере функциональности ЭО ПБОД, следовательно, можно отметить, что выполнены требования пункты 4.2.3 ТЗ «ЭО ПБОД должен быть разработан по модульному принципу и обеспечивать возможность доступа через открытый программный интерфейс (API)».

ПРОТОКОЛ

экспериментальных исследований № _*4*_ *от 21.06.2016 г.*

1. Объект исследования: модуль определения динамического адресного пространства ЭО ПБОД.

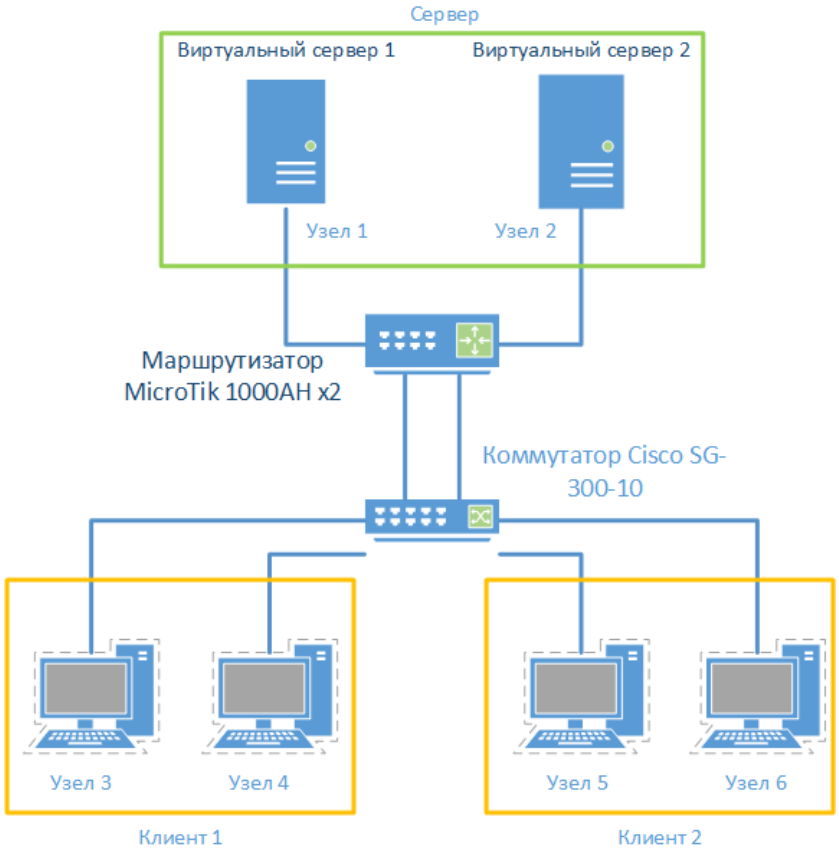
2. Цель исследования: проверка соответствия объекта исследования требованиям пункта № 4.2.4 ТЗ.

3. Дата начала исследования: 17.06.2016 г.

4. Дата окончания исследования: 21.06.2016 г.

6. Результаты исследования

Тестирование проводилось на скорости доступа портов 100 Мбит/с. Схема сети, используемая в тесте приведена на рисунке. В качестве сервера использовалось две виртуальных машины, у каждой из которых был отдельный сетевой интерфейс, одна из которых выполняла роль сервера (узел 1), другая клиента (узел 2). Аналогично на клиентских ПК запущены по 2 виртуальные машины с отдельными физическими сетевыми интерфейсами.



Тестирование проводилось на скорости доступа портов 100 Мбит/с. Для вычисления хэш-значения использовался алгоритм md5. В таблицах представлены сравнения хэш значений отправленных и полученных объектов для двух пар взаимодействующих узлов для каждой итерации.

№ итерации	Номера узлов	Хэш объекта перед отправкой	Хэш объекта после получения	Совпадение хэшей
1	5-6	06fe9147e55d31003b38ef30e1705aff	06fe9147e55d31003b38ef30e1705aff	да
2	4-3	f7a46fc0720c5130f99c05e9f191cc53	f7a46fc0720c5130f99c05e9f191cc54	да
3	6-3	7a32f62e8d82ced4dc953ed74bfdbb1f	7a32f62e8d82ced4dc953ed74bfdbb1f	да
4	6-3	da292fc5333edf544cee7e5e4e5e60ed	da292fc5333edf544cee7e5e4e5e60ed	да
5	4-2	0a40e4fb2aef8e950cc4b28fb4564d61	0a40e4fb2aef8e950cc4b28fb4564d62	да
6	6-4	4e3cc3b452de378f42dc95878181f92c	4e3cc3b452de378f42dc95878181f92c	да
7	6-2	732d901d729e35bbb0b40eff37e1d507	732d901d729e35bbb0b40eff37e1d508	да

№ итерации	Номера узлов	Хэш объекта перед отправкой	Хэш объекта после получения	Совпадение хэшей
8	6-4	186907f0b1097fe24c476f87d10a4deb	186907f0b1097fe24c476f87d10a4deb	да
9	2-2	d38b36daf02d0471f378c8f9e4a61c25	d38b36daf02d0471f378c8f9e4a61c26	да
10	5-2	47c9999df59056de01f15fded688fad2	47c9999df59056de01f15fded688fad3	да
11	3-5	8b3b963f516de8afee9de695ff0704da	8b3b963f516de8afee9de695ff0704da	да
12	3-5	1b8a14d46ce3ae834a3a3ea3ee0b0279	1b8a14d46ce3ae834a3a3ea3ee0b0280	да
13	3-6	d0ac04b75c78479da9a1c942bc194f3f	a7ea1a948548c1b4d9611daacd46b97a	нет
14	2-6	4224e33c1146b8ff4cb5e06f54ad2d52	4224e33c1146b8ff4cb5e06f54ad2d53	да
15	6-5	080171908942b0b959d343ac6214c261	080171908942b0b959d343ac6214c262	да
16	5-5	f4b5b24c74a981f2312a0e274431c5cb	f4b5b24c74a981f2312a0e274431c5cb	да
17	4-5	33460a2db5c0849f2822e21b3d19179f	33460a2db5c0849f2822e21b3d19179f	да
18	2-6	41f81f67a0b0803c15a1bb1f344a5f5d	41f81f67a0b0803c15a1bb1f344a5f5d	да
19	2-6	c5c9e9bddabcc6b50abb08c57960d40a	c5c9e9bddabcc6b50abb08c57960d40a	да
20	5-2	388ed9c809d60bf274ef507b63d7c143	388ed9c809d60bf274ef507b63d7c144	да
21	5-6	2be1338770db80307480f06db4151c0e	2be1338770db80307480f06db4151c0e	да
22	4-2	816e14b0845411306f2a304177875405	816e14b0845411306f2a304177875405	да
23	4-5	874e2af7539fdd2144f77940d0eb8419	874e2af7539fdd2144f77940d0eb8420	да
24	3-4	e76c5826f9c55df6f9a8675ab9858225	e76c5826f9c55df6f9a8675ab9858226	да
25	5-3	2f443a886b2bf5e89e98bf2c3ed45e16	2f443a886b2bf5e89e98bf2c3ed45e17	да
26	4-3	7908ccab27c4ebbd784deb5ac80a1c50	7908ccab27c4ebbd784deb5ac80a1c51	да
27	4-5	a15a006693e30a8ef9d8ac6625b84cbe	a15a006693e30a8ef9d8ac6625b84cbe	да
28	2-6	b115e27f72371f1932c15789fd60b14b	b115e27f72371f1932c15789fd60b14b	да
29	6-2	870c19ab481c0540c81a5f8385af505a	870c19ab481c0540c81a5f8385af505a	да
30	4-5	2f0c7b25208a75fa10dc6f9d1c41b91a	2f0c7b25208a75fa10dc6f9d1c41b91a	да
31	3-6	5a97ca2559b5d79c59b5900da1b115a7	5a97ca2559b5d79c59b5900da1b115a8	да

№ итерации	Номера узлов	Хэш объекта перед отправкой	Хэш объекта после получения	Совпадение хэшей
32	5-4	0ad18a2a1f22debc2f707ae16ff26d70	0ad18a2a1f22debc2f707ae16ff26d71	да
33	3-4	9207dc28aca7056f986fbdb07d5a9986	ea523c11a7b139e67953e27bbe7525f8	нет
34	3-3	ca2c021c8c60ddc43522987a45b7ca5f	ca2c021c8c60ddc43522987a45b7ca5f	да
35	2-6	2ffaf1c5b2b32c7d0fb0bafad9e44506	2ffaf1c5b2b32c7d0fb0bafad9e44507	да
36	6-3	19776d2252e64ec524e3059e81c61f2a	19776d2252e64ec524e3059e81c61f2a	да
37	4-6	c632887b2c73a41a69f23c95cdfccdff	c632887b2c73a41a69f23c95cdfccdff	да
38	3-3	a61417480fc50ba686d8da6c470fc81d	a61417480fc50ba686d8da6c470fc81d	да
39	2-6	ea22f54d6304da2baec2e712d456c364	ea22f54d6304da2baec2e712d456c365	да
40	4-3	b1d77206d800c3679de9ba8ad50eb506	b1d77206d800c3679de9ba8ad50eb507	да
41	2-3	6a637de857e7c01cee69d5a57d4c1aaf	6a637de857e7c01cee69d5a57d4c1aaf	да
42	4-6	ec6afc41e62e4cfe0139503f7af41371	ec6afc41e62e4cfe0139503f7af41372	да
43	2-4	8d058bf9647facb1237b72edd5e1bef6	8d058bf9647facb1237b72edd5e1bef7	да
44	5-4	0aff0af98266ba964d8ba09514c134c7	0aff0af98266ba964d8ba09514c134c8	да
45	6-3	c99bdc213b88ebbbeff071ced99af5b1	c99bdc213b88ebbbeff071ced99af5b2	да
46	3-5	715199c8ca821836274ef6afe87de517	715199c8ca821836274ef6afe87de518	да
47	3-5	172e89f04d3921c4e753b70aac6e599b	172e89f04d3921c4e753b70aac6e599b	да
48	4-5	5ea7c8f8491a8965d0c2b44c71d39c86	5ea7c8f8491a8965d0c2b44c71d39c87	да
49	3-4	1d6f82d4d651e3e03e73267f67f62731	1d6f82d4d651e3e03e73267f67f62732	да
50	6-2	240e895d291e0395f8d3bb8a5b97b0b9	240e895d291e0395f8d3bb8a5b97b0b10	да
			Доля ошибок	**4%**

№ итерации	Номера узлов	Хэш объекта перед отправкой	Хэш объекта после получения	Совпадение хэшей
1	4-6	847c2de710c70f83447f32ede79bb1e1	847c2de710c70f83447f32ede79bb1e2	да
2	4-2	086d55399a07327b5c0ac2e9bd4210c4	086d55399a07327b5c0ac2e9bd4210c5	да

№ итерации	Номера узлов	Хэш объекта перед отправкой	Хэш объекта после получения	Совпадение хэшей
3	5-6	a670859e513f47e334b0da14a819d372	a670859e513f47e334b0da14a819d373	да
4	2-5	4bdd4b6de1cddb16c30f689bf699e019	4bdd4b6de1cddb16c30f689bf699e020	да
5	3-5	ab267e4d0c75fefcb6eee1f934eb851f	ab267e4d0c75fefcb6eee1f934eb851f	да
6	2-3	a63d58ec7fb9a41c059f42da42896f20	a63d58ec7fb9a41c059f42da42896f21	да
7	2-6	25e3c64a4e69f58c8a6126dc417b36f0	25e3c64a4e69f58c8a6126dc417b36f1	да
8	5-3	3f845426fc25d28026612d2e12211455	3f845426fc25d28026612d2e12211456	да
9	2-5	ae048de1ae313260ad07d6c6c19b1c88	ae048de1ae313260ad07d6c6c19b1c89	да
10	6-5	3a4f63e5bf27bae13993ccba3ebe9ba0	3a4f63e5bf27bae13993ccba3ebe9ba1	да
11	5-6	64339e5fba0a03149cdb148f75f967bf	64339e5fba0a03149cdb148f75f967bf	да
12	3-6	53a1d4f3b37de0325d69b74edfff468d	53a1d4f3b37de0325d69b74edfff468d	да
13	2-6	892ae161a36993ad91d811769be40305	cd246a4b061c097cee1ccbd65692da69	нет
14	4-6	ab0344131d4f8ba0092e7ca236c48df6	ab0344131d4f8ba0092e7ca236c48df7	да
15	6-3	51b2f35ffd7b6a42df25b20dc51c5fd3	51b2f35ffd7b6a42df25b20dc51c5fd4	да
16	5-3	efac8a91796f5361db502cb7870c55dd	efac8a91796f5361db502cb7870c55dd	да
17	5-6	fb0c2dea60d01ad64394dbfd74fdf7fc	fb0c2dea60d01ad64394dbfd74fdf7fc	да
18	6-3	09b8bf2dd0ccce805afec0efa91a18e8	09b8bf2dd0ccce805afec0efa91a18e9	да
19	4-6	559ff449b18bfa26d68d386861a04684	559ff449b18bfa26d68d386861a04685	да
20	2-6	4bcb3e767a67bc52fbadc73bbaaf09d5	4bcb3e767a67bc52fbadc73bbaaf09d6	да
21	5-3	80199e5b33f23540044cfa99ce7083e3	80199e5b33f23540044cfa99ce7083e4	да
22	4-3	e1904f530216eef02437a2d43fa82bfe	e1904f530216eef02437a2d43fa82bfe	да
23	2-5	21c00f1eed746a19a46ac8b1151d411b	21c00f1eed746a19a46ac8b1151d411b	да
24	5-6	c5cc69e38637e6e5928fc15d43855f3c	c5cc69e38637e6e5928fc15d43855f3c	да
25	4-2	11360f7109d26e230409f9472a760f12	11360f7109d26e230409f9472a760f13	да
26	4-6	072dbbba36a3fb7a5c244f662872ab95	072dbbba36a3fb7a5c244f662872ab96	да

№ итерации	Номера узлов	Хэш объекта перед отправкой	Хэш объекта после получения	Совпадение хэшей
27	5-2	d4d0ddc7a8e377043bfa295908639da3	d4d0ddc7a8e377043bfa295908639da4	да
28	6-3	3f41ae9e4ce4eb11aa32d3658173a745	3f41ae9e4ce4eb11aa32d3658173a746	да
29	2-2	31c42b8347c33011718c4b8440a5141c	31c42b8347c33011718c4b8440a5141c	да
30	6-5	b14015e0cd8d328e4fb5116e37180957	b14015e0cd8d328e4fb5116e37180958	да
31	3-2	ccd31f4d51083cc64b42c6ebcb91dcd6	ccd31f4d51083cc64b42c6ebcb91dcd7	да
32	6-5	ef3f5c591ba827e10576f874f790a0d4	ef3f5c591ba827e10576f874f790a0d5	да
33	4-5	274458fe8e8d631a2d61ca78d3dad5ac	274458fe8e8d631a2d61ca78d3dad5ac	да
34	2-4	0ea6157f25cbcc1b7d26538ca73b4f93	0ea6157f25cbcc1b7d26538ca73b4f94	да
35	3-4	4885785c93653e339db4b6c8387a265d	4885785c93653e339db4b6c8387a265d	да
36	3-6	c92b9475992979c98cb21dc1568a6a59	c92b9475992979c98cb21dc1568a6a60	да
37	3-5	b168aeb7c9c67fed769eca512f1c2cc3	b168aeb7c9c67fed769eca512f1c2cc4	да
38	6-4	48d71cd4d444347f00361432d7e137e7	48d71cd4d444347f003614332d7e137e8	да
39	3-5	7cc1f90027121d86ba2b3222d477e51d	7cc1f90027121d86ba2b3222d477e51d	да
40	6-2	d2372b5c8a0f099e7d9149521a2d9b4a	9ba69a96cf69217ac512434f6eded0b6	нет
41	5-3	51b777996ecad4a2a7f4ed8937003a1f	51b777996ecad4a2a7f4ed8937003a1f	да
42	6-5	2d2c136b70c2dab510d7ea2acf80583c	2d2c136b70c2dab510d7ea2acf80583c	да
43	4-3	25d9bf6c878457bbaed4ef86775970bc	25d9bf6c878457bbaed4ef86775970bc	да
44	6-2	9a3f78fca7a626cc4ea10f4d5ac03d96	9a3f78fca7a626cc4ea10f4d5ac03d97	да
45	4-2	21e047f781da8a18bb0a46c2d9047f5e	21e047f781da8a18bb0a46c2d9047f5e	да
46	5-4	b11c1d1b656ffa6621175ad59c5f01b7	b11c1d1b656ffa6621175ad59c5f01b8	да
47	6-2	b272c3eee57c3be9ef0232bf3932e282	e32c34248f284f2b58c1ad837aaa8245	нет
48	3-2	1ea60022f5dc4546ef4567b71f03620a	1ea60022f5dc4546ef4567b71f03620a	да
49	6-3	1c59731cc791ebcf695cccddbebfd3af	1c59731cc791ebcf695cccddbebfd3af	да
50	6-5	d27d4906f14bc62c1dc7e69cccc4702f	d27d4906f14bc62c1dc7e69cccc4702f	да

№ итерации	Номера узлов	Хэш объекта перед отправкой	Хэш объекта после получения	Совпадение хэшей
			Доля ошибок	**6%**

Усредненная величина ошибки при определении конечного адресата и передаче данных ему: 5%.

7. Замечания и рекомендации

Отсутствуют

8. Выводы

По результатам экспериментальных исследований было установлено, что разработанная реализация ЭО ПБОД обеспечивает на приемлемом уровне одновременную работу не менее 3 пользователей (п. 4.2.1), доступ к 3 ресурсам двух разных типов (п. 4.3.1.1). Во время выполнения теста были найдены все требуемые адресаты и переданы им необходимые данные, поскольку в ЭО ПБОД отсутствует поле контроля целостности и данные повторно отправляются только в случае отсутствия подтверждения, то было 5 случаев частичной потери передаваемых данных, когда передаваемый файл разбивался на несколько UDP дейтаграмм, и они не все были доставлены по назначению. В результате полученный бинарный файл не соответствовал исходному файлу, что проявилось в изменении его хэша. Таким образом можно говорить о 5% ошибке эффективности отслеживания динамических характеристик системы и достижении задачи из п. 4.2.4 ТЗ «обеспечивать возможность отслеживания динамических характеристик распределенной информационно-вычислительной системы, изменяемых в соответствии с технологией движущейся цели».

ПРОТОКОЛ

экспериментальных исследований № _5_ *от 15.06.2016 г.*

1. Объект исследования: технические характеристики ЭО ПБОД.

2. Цель исследования: проверка соответствия объекта исследования требованиям пункта № 4.3.1 ТЗ.

3. Дата начала исследования: 13.06.2016 г.

4. Дата окончания исследования: 15.06.2016 г.

6. Результаты исследования

Согласно протоколу № 1 ЭО ПБОД в среднем обеспечивает одновременную работу 4,84 пользователя.

Согласно перечню средств измерений, приведенном в таблице, и реализованном в лабораторном стенде выполняются требования подпунктов 4.3.1.2 и 4.3.1.3, в частности процедура использования вэб-сервера, вэб-браузера и реляционной БД в ЭО ПБОД подробно представлена в научном отчете в части описания протокола.

Наименование, тип и марка	Кол-во	Основные характеристики
ПК IBM PC	2	ПК на базе Intel core i5 4 ядра на частоте 3,2 ГГц 8 Гб ОЗУ 500 Гб HDD 2 сетевых интерфейса Gigabit Ethernet
Операционная система	2	Lubuntu 15
Web-браузер	2	Firefox 44
Сервер	1	Сервер на базе Intel Xeon E3 4 ядра по 3,2 ГГц 16 Гб ОЗУ 500 Гб HDD 3 сетевых интерфейса

		Gigabit Ethernet
Операционная система для сервера	1	Linux Debian Server 8.x
Web-сервер	1	Apache 2.4
СУБД	1	MySQL
Коммутатор	1	Коммутатор L2+ Cisco SG-300-10 10 портов Gigabit Ethernet
Маршрутизатор	1	MikroTik RouterBOARD 1100AHx2 12 портов Gigabit Ethernet, RAM 2 Gb, частота процессора 1ГГц
Сетевой кабель	8	UTP cat 6

Анализ представленных исходных кодов ЭО ПБОД позволяет дать заключение о использовании в качестве языка программирования C++, что также подтверждается свидетельством о государственно регистрации программы для ЭВМ № 2016614108, Дата регистрации: 14.04.2016.

Использование формата JSON и конфигурационных файлов в формате ключ-значение осуществляется при взаимодействии пользователя по протоколу аутентификации с сервером. Подробно данная процедура и форматы данных представлены в описании протокола аутентификации научного отчета. Следовательно, соответствие подпункт 5 п. 4.5 программы исследований частичное, поскольку формат JSON не применяется в явном виде для передачи данных между пользователями, а только для конфигурационных сообщений.

7. Замечания и рекомендации

Отсутствуют

8. Выводы

По результатам экспериментальных исследований было установлено, что разработанная реализация ЭО ПБОД соответствует на 95% требованиям пункта 3.1.1 ТЗ. Неполное соответствие требованиям обусловлено выбором более лучшего формата представления данных при передаче, основанного на позиционном размещении данных без указания идентификатора поля при передаче.

ПРОТОКОЛ

экспериментальных исследований № _6_ *от 16.06.2016 г.*

1. Объект исследования: структура ЭО ПБОД.

2. Цель исследования: проверка соответствия объекта исследования требованиям пункта № 4.3.2 ТЗ.

3. Дата начала исследования: 13.06.2016 г.

4. Дата окончания исследования: 16.06.2016 г.

6. Результаты исследования

На основе анализа исходных кодов модулей ЭО ПБОД, предоставленных на регистрацию в Федеральную службу по интеллектуальной собственности (свидетельство о регистрации № 2016614108, Дата регистрации: 14.04.2016), и описания протокола ЭО ПБОД было установлено наличие совместно функционирующих 5 модулей, реализованных в виде отдельных классов и функций со стандартизированными форматами ввода-вывода данных:

1) модуль аутентификации пользователя по логину и паролю;

2) модуль авторизации пользователя с использованием принадлежности пользователя к группам и его ролям в них;

3) модуль делегирования прав пользователя, полученных на основе аутентификации и авторизации, связанным с пользователем программным компонентам;

4) модуль определения динамического адресного пространства в распределенной информационно-вычислительной системе;

5) модуль защиты от исследования программных компонентов распределенной информационно-вычислительной системы.

Результаты выполнения пунктов программы при единичном тестировании функционала модулей после их последовательной обфускации модулем защиты от исследований программных компонентов представлены в таблице.

п. программы / изменяемый модуль	Аутентификации	Авторизации	Делегирования прав	Определения адресного пространства	Защиты от исследований
4.7	Выполнено частично	Выполнено	Выполнено	Выполнено	Выполнено
4.8	Выполнено	Выполнено частично	Выполнено	Выполнено	Выполнено
4.9	Выполнено	Выполнено частично	Выполнено	Выполнено	Выполнено
4.10	Выполнено	Выполнено	Выполнено	Выполнено частично	Выполнено
4.14	Выполнено	Выполнено	Выполнено	Выполнено	Выполнено

Частичное выполнение пункта программы испытаний означает возникновение случаев превышение усредненного времени выполнения операций модулем, следовательно, и клиентским приложением, от усредненных показателей работы модулей в обычном режиме при проведении испытаний.

7. Замечания и рекомендации

Отсутствуют

8. Выводы

По результатам исследований было установлено наличие в ЭО ПБОД 5 взаимосвязанных модулей, взаимодействующих через стандартизированные программные интерфейсы. Изменение их внутренней структуры с сохранением форматов ввода-вывода не приводит к потере функциональности ЭО ПБОД, следовательно, можно отметить, что выполнены требования пункты 4.2.3 ТЗ «В состав ЭО ПБОД должны входить:

1) модуль аутентификации пользователя по логину и паролю;

2) модуль авторизации пользователя с использованием принадлежности пользователя к группам и его ролям в них;

3) модуль делегирования прав пользователя, полученных на основе аутентификации и авторизации, связанным с пользователем программным компонентам;

4) модуль определения динамического адресного пространства в распределенной информационно-вычислительной системе;

5) модуль защиты от исследования программных компонентов распределенной информационно-вычислительной системы.».

Исследование проводили

Доцент		Н.Ю. Паротькин
Доцент		В.В. Золотарев
Инженер		Т.А. Панфилова
Инженер		О.А. Сопова