Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

ЭВРИСТИЧЕСКИЙ ПОДХОД К СОКРАЩЕНИЮ ДИАГНОСТИЧЕСКОЙ ИНФОРМАЦИИ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Афанасьева Георгия Михайловича

Научный руководитель
зав.каф.,к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2016

1 Общая характеристика работы

1.1 Актуальность темы исследования

В настоящее время все более активно развивается направление алгоритмов, основанных на природных принципах. Такие алгоритмы базируются на наблюдениях ученых за животными и насекомыми. Их представление в виде математических моделей и использование таких моделей для решения различного рода задач, дает хорошие результаты для постоянного применения.

1.2 Цели и задачи работы

Целью настоящей работы является использование эвристических подходов, основанных на принципах муравьиных алгоритмов, для решения задачи оптимизации сокращения диагностической информации.

Для достижения поставленной цели сформулированы следующие задачи:

- адаптировать механизм муравьиных алгоритмов к решению задачи сокращения диагностической информации;
- рассмотреть пути оптимизации полученного метода сокращения;
- реализовать адаптированный алгоритм и его возможные модификации;
- провести ряд экспериментов для выявления оптимальных значений параметров алгоритмов;
- провести сравнительный анализ разработанных подходов.

1.3 Структура работы

Работы состоит из введения, двух глав, заключения, списка использованной литературы и двух приложений. Основная часть исследования изложена на 50 страницах, текст проиллюстрирован графиками, построенными на основе таблиц из приложения.

2 Основное содержание работы

Первая глава «Применение муравьиных алгоритмов для сокращения диагностической информации» описывает основные понятия теории диагностирования, постановку задачи сокращения диагностической информации, а так же применения муравьиных алгоритмов для решения этой проблемы.

Техническая диагностика является одним из ключевых способов обеспечения и поддержания высокой надежности электронных устройств [1]. При диагностировании, первым делом, необходимо выяснить техническое состояние системы в текущий момент времени. Это значит, что необходимо проверить исправность функционирования системы, понять лежат ли значения параметров системы в нужных пределах. Проверить не отказала ли система, и правильно ли она выполняет назначенную функцию. Затем, следует выявить дефекты, нарушающие правильное функционирование системы.

Понятие **технической диагностики** задается как «область знаний, охватывающая теорию, методы и средства определения технического состояния объектов» [2, 3]. **Объектом диагностирования (ОД)** называется объект, состояние которого необходимо определить.

Множество технических состояний ОД образуют все исправные и неисправные технические состояния. **Техническое состояние объекта** — состояние, которое характеризуется в определенный момент времени, при определенных условиях внешней среды значениями параметров, установленных технической документацией на объект.

Диагностирование предполагает собой процесс изучения ОД, в результате получают заключения вида: ОД исправен, неисправен, в объекте есть такая-то неисправность.

Если объект работает неправильно, то для устранения неисправности сперва необходимо установить ее место. Поиск осуществляется при помощи выполнения диагностического теста над объектом и декодирования его результатов. Диагностическая проверка разделяется на несколько частей, каждая связана с поступлением на объект входного воздействия (тестового или рабочего) и определением выходной реакции объекта. Эти фрагменты диагностического эксперимента называются **элементарными проверками** [4–6].

Ответы объекта могут поступать как с основных выходов ОД, необходимых для применения ОД по назначению, так и с дополнительных выходов,

предусмотренных специально для диагностирования и не участвующих непосредственно в работе прибора. Основные и дополнительные выходы называют **контрольными точками (КТ)** или контролируемыми выходами. Параметры, измеряемые на них называют контролируемыми или **диагностическими параметрами**. С одной КТ можно снять измерения нескольких параметров. К примеру, при контроле сигнала синусоидальной формы можно одновременно измерить частоту и амплитуду сигнала [7].

Процесс диагностирования подразумевает собой неоднократную подачу на ОД специальных воздействий — **входных сигналов**, а так же анализ ответов на эти воздействия.

Для функционирования систем диагностирования требуется заранее подготовить необходимые данные. Информация, полученная путем инструментального контроля или из документации образует совокупность диагностических признаков. Это множество используется для формирования исходной **диагностической информации (ДИ)** [8].

Исходная ДИ должна быть тщательно проанализирована. Такой анализ должен быть упорядочен. Для этого необходимо заранее составить диагностическую таблицу или так называемый **словарь неисправностей (СН)**. Число строк этой таблицы должно быть равно числу состояний ОД, а число столбцов — количеству элементарных проверок. Без помощи вычислительной техники невозможно быстро и качественно получить требуемые СН. Поэтому для формирования таких таблиц необходимо построение математических моделей.

Из вышесказанного вытекает основная **цель диагностирования** — требуется оценить выходные параметры системы и обнаружить источники их отклонения от заданных значений. При этом следует принять во внимание весь диапазон режимов работы системы а также условий ее эксплуатации, и вероятность изменения выходных параметров во времени, то есть учитывать параметрическую надежность.

Итоговые результаты тестовых воздействий задаются явной математической моделью. Особенно эффективным и удобочитаемым способом задания этой модели является **таблица функций неисправностей(ТФН)**. ТФН это матрица, в которой состояния ОД являются строками, а столбцами — приемлемые элементарные проверки. В ячейках находятся результаты, соответствующие j-ой элементарной проверке, находящейся в i-ом состоянии.

Таблица функций неисправностей хороша как универсальная математическая модель объекта диагностирования. Она содержит наглядную информацию, доступную для понимания не только компьютеру, но и человеку. Однако построение диагностических алгоритмов на основе ТФН редко заканчивается успехом, ввиду ее большого объема. Поэтому на практике вместо полных таблиц функций неисправностей применяются **Т-таблицы функций неисправностей (Т-ТФН)**. Т-ТФН это подтаблица ТФН, в которой строка, соответствующая состоянию s_0 отличается от всех других. Так же в этой таблице любые две строки различны между собой [9].

Основой муравьиных алгоритмов оптимизации является имитация жизнедеятельности колонии. Наряду с этим, колония рассматривается как система, в которой каждый муравей следует простым правилам [10–12]. Несмотря на простоту поведения каждого муравья в отдельности, поведение всей системы получается очень разумным. Такие методы являются вероятностной жадной эвристикой, где вероятности оцениваются, основываясь на информации о качестве решения, накопленной из предыдущих решений. Они используются как для статических, так и для динамических оптимизационных задач перебора. Всегда можно получить оптимальное решение, т.е сходимость гарантирована, тем не менее скорость сходимости невозможно предугадать.

Алгоритм основан на имитации поведения муравьиной колонии и отдельных ее особей. Которые маркируют более удачные пути большим количеством феромона. Выполнение алгоритма начинается с размещения муравьев в случайные вершины графа, а затем происходит передвижение муравьев — вероятностным методом определяется направление движения [13].

Так как в основе алгоритма лежит моделирование движения муравьев по разным путям, такой подход может стать эффективным способом поиска подходящих решений для задач оптимизации, использующих графовую модель представления. Ряд опытов показал, что эффективность муравьиных алгоритмов возрастает с ростом размерности решаемых задач оптимизации.

Из-за того, что муравьиные алгоритмы крайне эффективны при задачах поиска. Поэтому, как предлагается в [14,15], для решения поставленной задачи сокращения диагностической информации необходимо представить ее в виде задачи поиска.

Моделирование поведения муравьев связано с распределением феромона

на вершинах графа. При этом вероятность включения вершины в маршрут отдельного муравья пропорциональна количеству феромона на этой вершине, а так же привлекательности данной вершины в рамках разбиения множества неисправностей на классы эквивалентности. Количество откладываемого феромона равно приросту обнаруживаемых неисправностей, относительно предыдущей вершины. Чем больше классов эквивалентности породит переход, тем больше феромона будет отложено на его вершине, следовательно, большее количество муравьев будет включать ее в синтез собственных маршрутов [16].

Так же, во избежание преждевременной сходимости, необходимо включить в алгоритм моделирование отрицательной обратной связи в виде испарения феромона. Однако стоит учитывать, что если феромон испаряется быстро, это приведет к потере памяти колонии и забыванию хороших решений, с другой стороны, большее время испарения приведет к устойчивому локальному оптимальному решению.

Алгоритм поиска является вариацией стандартного муравьиного алгоритма, описанного выше [17]. Множество рабочих муравьев содержит всех муравьев, которые в данный момент задействованы в поиске решения. Множество муравьев уже нашедших решение, состоит из особей которые уже нашли оптимальный по их мнению маршрут.

Алгоритм начинает выполняться в цикле. Условием выхода из которого является достижение заданного количества поколений (итераций по графу). Через определенное количество итераций к множеству рабочих муравьев добавляется заданное число новых муравьев. Каждый новый муравей помещается в случайную вершину.

Далее все рабочие муравьи делают переход в другую вершину. Вершина для переходя выбирается исходя из количества феромона и показателя эффективности разбиения множества состояний. Эти показатели влияют на вероятность выбора этой вершины муравьем,

Чем больше классов эквивалентности породит переход в новую вершину, тем быстрее найдется решение. Решением для отдельного муравья является разбиение всего множества состояний на классы эквивалентности, содержащих по 1 элементу. Другими словами муравей находит решение, когда комбинация элементарных проверок позволяет однозначно определить любую неисправность в диагностической таблицы. Если после перехода в новую вер-

шину муравей нашел решение, то он исключается из списка рабочих муравьев и добавляется к списку нашедших решение муравьев.

После того, как в текущем поколении все муравьи совершат переход, количество феромона на всех вершинах уменьшается. А затем к вершинам, в которых побывали муравьи, добавляется суммарное количество феромона от всех посетивших данную вершину муравьев.

К концу выполнения итерации увеличивается счетчик поколений и начинается выполнение новой. Алгоритм заканчивает работу, когда количество поколений достигнет максимального количества поколений.

На выходе алгоритм выводит список муравьев, которые нашли решение. Список отсортирован по длине пути каждого муравья, чем короче путь, тем лучше, так как он содержит меньшее число вершин графа, соответствующих элементарным проверкам. Исходя из списка пройденных вершин графа, можно построить СПР, содержащий только те проверки, которые были выбраны в качестве решения. Таким образом алгоритм позволяет сократить объем диагностической информации.

Для сравнения производительности и эффективности адаптированного муравьиного алгоритма, были два следующих алгоритма:

Первый алгоритм основан на случайном выборе вершин. В данной реализации муравьи ничем не руководствуются при выборе очередной вершины, они просто случайно перемещаются из одной в другую пока не найдут решение.

Второй алгоритм представляет собой объединение случайный алгоритма и первоначального муравьиного. Суть данного метода заключается в том, что муравьи выбирают случайную вершину, руководствуясь только количеством феромона на ней. А значение феромона, откладываемое муравьем после перехода в эту вершину, зависит от того как много новый классов эквивалентности породил текущий переход.

Вторая глава «Реализация муравьиного алгоритма для задачи оптимизации сокращения диагностической информации» содержит описание структуры реализованного приложения а так же функционирования основных программных модулей. Так же здесь приводятся примеры работы с интерфейсом программы и сравнение результатов исследований, полученных в ходе работы.

Для реализации описанных выше алгоритмов, было написано приложе-

ние, которое обладает следующим функционалом:

1. выбор текстового файла, содержащего диагностическую таблицу;
2. отображение выбранного файла, а так же путь к нему;
3. возможность найти и отобразить сокращенную диагностическую информацию;
4. сохранить полученные результат в текстовом формате;
5. вывод сведений о количестве найденных столбцов из первоначального файла;
6. вывод сведений о количестве затраченного времени.

Исходный код приложения был написан на языке java, и содержит 8 классов:

— `Table.java` — позволяет оперировать с файлами и представляет их в качестве таблицы, необходимой для построения графа или поиска решения в алгоритме.
— `Graph.java` — необходим для представления таблицы в виде графа;
— `States.java` — задает представление множества неисправностей, а так же разбиения его на классы эквивалентности
— `Ant.java` — содержит все свойства, характерные для конкретного муравья, а так же действия по выбору и переходу между вершинами графа;
— `ProbEntry.java` — сущность для хранения промежуточных значений расстояния между двумя вершинами;
— `ProbEntryArrayWithSum.java` — список всех промежуточных значений $d_{i,j}$ а так же их сумма, для конкретного муравья на конкретной итерации;
— `Colony.java` — содержит граф и таблицу для перехода, а так же сами алгоритмы поиска пути;
— `Program.java` — класс необходимый для связи пользовательского интерфейса и самой программой, не нуждается в детально описании;
— `App.java` — класс реализующий пользовательский интерфейс, не требует подробного объяснения.

Рассмотрим подробнее каждый класс и его реализацию.

Класс `Table.java` содержит следующие поля:

— `rowCount` — количество строк (неисправностей) в таблице;
— `colCount` — количество столбцов (элементарных проверок);
— `name` — имя файла, на основе которого была построена таблица;

— table — сама таблица, представленная в виде списка списков.

Класс Table.java содержит следующие функции:

— loadTable — для загрузки таблицы из файла;
— readBitsFromLine — для считывания последовательности из строки;
— stripDuplicatesFromFile — для исключения из файла всех повторяющихся строк.

При старте программы создается экземпляр класса Table.java, и с помощью функции loadTable происходит загрузка и разбиение таблицы на столбцы.

Класс Graph.java содержит следующие поля:

— dryOutIntensive — интенсивность испарения феромона;
— nodes — список вершин графа, с феромоном на них;
— nodesCount — количество вершин в графе.

Класс Graph.java содержит следующие функции:

— reducePhero — имитация испарения феромона ;
— getNodePheromone — получить значение феромона на конкретной вершине;
— addAntsPheromone — добавить феромоны от всех муравьев, на посещенные на текущей итерации вершины.

Класс States.java содержит следующие поля:

— table — ссылка на объект таблицу, статическое поле одинаковое для всех экземпляров класса;
— sizeOfOneStates — количество классов эквивалентности размерности 1;
— listOfListOfStates — список классов эквивалентности.

Класс States.java содержит следующие функции:

— getListOfListOfStates — получить список классов эквивалентности;
— addState — добавить класс эквивалентности к списку ;
— probabilitySplitStates — разбить список классов эквивалентности в зависимости от последовательности элементарных проверок, но не вносить изменений в sizeOfOneStates и listOfListOfStates. Необходимо для вычисления длины пути от одной вершины до другой;
— splitStates — то же самое, что и probabilitySplitStates, но с изменением списков классов sizeOfOneStates и listOfListOfStates. Вызывается конкретным муравьем, после перехода в очередную вершину;
— splitStatesForRandomStep — то же самое, что и splitStates, но для

случайного алгоритма.

Класс `Ant.java` содержит следующие поля:

— `table` — ссылка на объект таблицу, статическое поле одинаковое для всех экземпляров класса;
— `graph` — ссылка на объект графа, статическое поле одинаковое для всех экземпляров класса;
— `count` — счетчик для подсчета количества созданных экземпляров, увеличивается на единицу при создании нового муравья;
— `alpha` — параметр альфа, статическое поле одинаковое для всех экземпляров класса;
— `betta` — параметр бета, статическое поле одинаковое для всех экземпляров класса;
— `visitedNodes` — список пройденных вершин графа, для конкретного муравья;
— `states` — разбиение неисправностей на классы эквивалентности;
— `id` — идентификатор муравья, присваивается ему при создании экземпляра класса, исходя из значения `count`;
— `currentNodeId` — номер текущей вершины муравья.

Класс `Ant.java` содержит следующие функции:

— `getCurrentNodeId` — получить текущую вершину, в которой находится муравей;
— `getPathSize` — получить длину пути, пройденного муравьем;
— `getVisitedNodes` — получить список посещенных вершин;
— `getUnvisitedNodes` — получить список не посещенных вершин;
— `isPathFound` — проверить нашел ли муравей решение;
— `calcProbAndDenominatorForUnvisitedNodes` — вычислить вероятность перехода, для всех вершин в которые можно перейти из текущей и выдать список интервалов вероятности для каждой из них. Используется для муравьиного алгоритма;
— `calculateProbability` — генерировать случайное число из промежутка $[0, 1]$ и на основе полученных из `calcProbAndDenominatorForUnvisitedNodes` интервалов, выбрать следующую вершину. Используется для муравьиного алгоритма;
— `calcComboProbAndDenominatorForUnvisitedNodes` — то же самое, что и

calcProbAndDenominatorForUnvisitedNodes только для комбинированного алгоритма;

— calculateComboProbability — то же самое, что и calculateProbability только на основе функции calcComboProbAndDenominatorForUnvisitedNodes, используется для комбинированного алгоритма;
— chooseRandomNodeId — выбрать случайную вершину для перехода;
— makeStep — сделать переход в вершину, выбранную с помощью calculateProbability;
— makeRandomStep — сделать переход в вершину, выбранную с помощью chooseRandomNodeId;
— makeComboStep — сделать переход в вершину, выбранную с помощью calculateComboProbability;
— clearUnnecesaryFields — очистить список классов эквивалентности, необходимо для сокращения объема оперативной памяти, вызывается когда муравей нашел решение.

Класс Colony.java содержит следующие поля:

— finishedAnts — список муравьев нашедших решение;
— workingAnts — список муравьев, которые ищут решение;
— runtime — доступ к среде исполнения JVM, для мониторинга памяти;
— averageMemory — средний расход памяти приложения;
— genCounter — счетчик поколений.

Класс Colony.java содержит следующие функции:

— addMemory — добавить заданное количество памяти к averageMemory;
— getAverageMemory — получить среднее значение затрат по памяти за время выполнения всех итераций;
— parallelFindPath — параллельный вариант муравьиного алгоритма;
— parallelFindRandomPath — параллельный вариант случайного алгоритма;
— parallelFindComboPath — параллельный вариант комбинированного алгоритма;
— getSortedFinishedAnts — получить список, нашедших решение муравьев, отсортированный в порядке увеличения качества решения;
— getMinPathSize — получить длину минимального пути;

— `printResult` — вывести результат муравьиного алгоритма;
— `printRandomResult` — вывести результат случайного алгоритма;
— `printComboResult` — вывести результат комбинированного алгоритма;

Для работы алгоритма необходимо загрузить таблицу из файла. Она состоит из `rowCount` строк и `colCount` столбцов. При загрузке таблицы, ее столбцы группируются по `nBits` штук. Одна такая последовательность затем представляется в качестве вершины графа — `graph`. Каждая вершина графа несет в себе определенное количество феромона — `nodePheromone`, который изначально задается единицей.

3 Результаты работы

В ходе работы были реализованы три алгоритма поиска пути. Чтобы определить эффективность работы каждого, были проведены измерения с целью выявления оптимальных параметров. В качестве входных данных для сокращения, использовалась диагностическая информация для устройств из каталога ISCAS’85 [18].

По результатом проведенных исследований можно сделать вывод о том, что для получения наилучшего решения необходимо:

- использовать комбинированный алгоритм;
- производить поиск по всем столбцам таблицы, не группировать их;
- не перегружать алгоритм лишними муравьями, во избежание существенных затрат по времени и памяти, иначе поиск будет нецелесообразен;
- подобрать минимальное значение периода рождаемости, так чтобы оно позволяло найти решение, тогда алгоритм не будет делать порождать лишних муравьев, и будет работать значительно быстрее;
- подобрать оптимальное количество итераций, примерно 1/3 от общего количества вершин графа, это позволит сократить время работы алгоритма без потери уже найденных решений.

В ходе выполнения дипломной работы были описаны и реализованы три алгоритма сокращения диагностической информации. Приведен сравнительный анализ их быстродействия, затрат по памяти и эффективности сжатия диагностических таблиц. Проведено исследование различных параметров и их влияние на результаты работы алгоритмов. Сделаны выводы по полученной информации и даны рекомендации по применению их на реальных данных, с целью улучшения качества и быстродействия. Кроме того реализовано пользовательское приложение, которое позволяет быстро задать необходимые данные и сохранить результаты работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Карибский, В. В.* Основы технической диагностики. Модели объектов, методы и алгоритмы диагноза / В. В. Карибский, П. П. Пархоменко, Е. С. Согомонян, В. Ф. Халчев; Под ред. П. П. Пархоменко. — Москва: Энергия, 1976. — С. 464.

2 ГОСТ 20911-89. Техническая диагностика. Термины и определения[Текст]. – Взамен ГОСТ 20911-75; введ. 1991-01-01. — Москва: Изд-во стандартов, 2009. — С. 11.

3 ГОСТ-27.002-89 Надежность в технике. Основные понятия. Термины и определения [Текст]. — Москва: Изд-во стандартов, 2002. — С. 24.

4 *Скобцов, Ю. А.* Логическое моделирование и тестирование цифровых устройств / Ю. А. Скобцов, В. Ю. Скобцов. — Донецк: ИПММ Украины, ДонНТУ, 2005.

5 *Малышенко, Ю. В.* Техническая диагностика / Ю. В. Малышенко. — URL:`http://abc.vvsu.ru/books/teh_diag/Default.asp` (дата обращения 05.04.2016). Загл. с экр. Яз. рус.

6 *Барашко, А. С.* Моделирование и тестирование дискретных устройств / А. С. Барашко, Ю. А. Скобцов, Д. В. Сперанский. — Киев: Наукова думка, 1992.

7 *Чжен, Г.* Диагностика отказов цифровых вычислительных систем / Г. Чжен, Е.Мэннинг, Г.Метц. — Москва: Мир, 1972. — С. 232.

8 *Богомолов, А. М.* Аналитические методы в задачах контроля и анализа дискретных устройств / А. М. Богомолов, Д. В. Сперанский. — Саратов: Изд-во Саратовского ун-та, 1986.

9 *Миронов, С. В.* Разработка методов сокращения диагностической информации [Рукопись]. Дис . . . канд. физ.-мат. наук / С. В. Миронов. — Саратов, 2008.

10 *Штовба, С. Д.* Муравьиные алгоритмы / С. Д. Штовба // *Математика в приложениях*. — 2003. — № 4.

11 *Скобцов, Ю. А.* Эволюционные вычисления / Ю. А. Скобцов, Д. В. Сперанский. — Москва: Национальный Открытый Университет «ИНТУИТ», 2015. — С. 331.

12 *Deneboug, J. L.* Collective patterns and decision making / J. L. Deneboug, S. Goss // *Ethology Ecology and Evolution*. — 1989. — Vol. 1, no. 4. — Pp. 295 – 311.

13 *Dorigo, M.* An introduction to ant colony optimization / M. Dorigo, K. Socha // *MIT Press*. — 2006.

14 *Миронов, С. В.* Генетические алгоритмы для сокращения диагностической информации / С. В. Миронов, Д. В. Сперанский // *Автоматика и телемеханика*. — 2008. — № 7. — С. 146–156.

15 *Сперанский, Д. В.* Введение в генетические алгоритмы: Учеб.пособие / Д. В. Сперанский, В. Г. Самойлов, О. В. Емельянова; Под ред. Д. В. Сперанского. — Саратов: Изд-во Саратовского ун-та, 2006.

16 *Гладков, Л. А.* Генетические алгоритмы / Л. А. Гладков, В. В. Курейчик, В. М. Курейчик. — 2-е изд., испр. и доп изд. — Москва: ФИЗМАТЛИТ, 2006. — С. 320.

17 *Сперанский, Д. В.* Муравьиный алгоритм синтеза тестов для цифровых устройств // Компьютерные науки и информационные технологии : Материалы Международной научной конференции. — Саратов: Издат. центр «Наука», 2014. — С. 323–325.

18 *Brglez, F.* A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran // Proceedings of 1985 International Symposium on Circuits and Systems. — Kyoto, Japan: June 1985.