МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ
РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ В.Ф. УТКИНА

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ПРИКЛАДНЫХ ИССЛЕДОВАНИЯХ

Межвузовский сборник научных трудов

Рязань 2024

УДК 004

Информационные технологии в прикладных исследованиях, межвузовский сборник научных трудов. – Рязань: ИП Коняхин А.В. (Book Jet), 2024. – 336 с.

ISBN 978-5-907811-32-4

Публикуются статьи о проблемах использования информационных технологий в науке, промышленности и образовании.

Сборник рассчитан на научных сотрудников, преподавателей, аспирантов и студентов высших учебных заведений.

Авторская позиция, орфография, пунктуация и стилистические особенности публикаций сохранены.

ISBN 978-5-907811-32-4

СОДЕРЖАНИЕ

УДК 004.415.2

АНИСИМОВ Д.В.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ПРИМЕНЕНИЕ MES-СИСТЕМ ДЛЯ АВТОМАТИЗАЦИИ ПРОИЗВОДСТВЕННОГО ПРОЦЕССА

Проводится исследование и критерии применения MES-систем для автоматизации производственного процесса. Описываются главные способы реализации и тенденции развития и применяемые решения подобных систем.

Оптимизация производства является ключевым аспектом эффективной работы любой промышленной компании. Большинство предприятий сталкиваются с высокими затратами на производство, длительными циклами поставок, избыточными запасами и другими проблемами, которые могут негативно сказываться на конкурентоспособности и прибыльности бизнеса. Для решения конкретной проблемы производства могут быть применены инструменты для её автоматизации, но, когда предприятие разрастается, сложность и количество исполняемых операций увеличивается в разы, вместе с тем растет количество отдельных автоматизированных операций. В этот момент встает задача интеграции процессов в единую систему, позволяющую управлять всем циклом производства или большей его частью.

Для решения данной проблемы применяются MES-системы. *MES (Manufacturing Execution System)* – производственная исполнительная система. Специализированное прикладное программное обеспечение, предназначенное для решения задач анализа, оптимизации, координации, синхронизации в контексте выпуска продукции предприятием [1].

MES-системы позволяют улучшить управление производством, давая возможность оперативно реагировать на изменения рынка и контролировать исполнение заказов. Это помогает предотвратить избыточные запасы, упростить планирование и повысить общую эффективность. Также важным аспектом MES-системы является сбор и анализ данных о производственных операциях, что в свою очередь дает возможность руководству следить за ходом производства в режиме реального времени, оптимизировать рабочие процессы, сократить время простоя оборудования, снизить затраты на энергию и сырье, улучшить качество и уменьшить процент брака. Кроме того, MES-системы помогают повысить прозрачность и контроль в

производственной среде. Они обеспечивают централизованное хранение информации, предоставляют доступ к данным на всех уровнях управления, что позволяет компании повысить надежность процессов и обеспечить соблюдение стандартов качества и безопасности.

В 1997 году Международная ассоциация решений для производственных предприятий, или MESA, определила 11 основных функций системы управления производством. Хотя модель MESA-11 со временем эволюционировала, эти первоначальные 11 основных функций обеспечивают основу для работы практически любого типа предприятия и являются неотъемлемой частью современных производственных систем управления. Это:

- Распределение ресурсов и их состояние: Использование данных в режиме реального времени для отслеживания и анализа состояния ресурсов, включая машины, материалы и рабочую силу, для внесения корректировок в распределение.
- Операции / подробное планирование: Оптимизация производительности за счет планирования, хронометража и последовательности действий на основе приоритетов и емкости ресурсов.
- Диспетчеризация производственных подразделений: Управление потоком производственных данных в режиме реального времени, для упрощенного ввода расчетных корректировок.
- Контроль документации: Управление документами, включая рабочие инструкции, чертежи, стандартные операционные процедуры, отчеты о пакетах и многое другое, и распространение их, чтобы обеспечить их доступность и возможность редактирования.
- Сбор данных: Отслеживание и сбор данных о процессах, материалах и операциях в режиме реального времени и использование их для принятия более обоснованных решений и повышения эффективности.
- Управление трудовыми ресурсами: Отслеживание графиков работы сотрудников, их квалификацию и разрешения, для оптимизации управление трудовыми ресурсами с меньшими затратами времени и ресурсов со стороны руководства.
- Управление качеством: Отслеживание отклонений от качества и исключений для улучшения управления контролем качества и документации.
- Управление процессами: Управление всем производственным процессом от получения заказа до выпуска готовой продукции.

- Управление техническим обслуживанием: Использование данных из MES для выявления потенциальных проблем с оборудованием до того, как они возникнут, и корректировка графиков технического обслуживания оборудования, инструмента и станков, чтобы сократить время простоя и повысить эффективность.
- Отслеживание продуктов и генеалогия: Отслеживание прогресса в разработке ваших продуктов и их генеалогию для принятия обоснованных решений. Наличие полной истории продукта чрезвычайно полезно производителям, которые должны соблюдать государственные или отраслевые нормативы.
- Анализ производительности: Сравнение результатов и цели, для выявление сильных и слабых стороны всего процесса и использование этих данных для повышения эффективности систем [2].

В то время, как модель MESA-11 сфокусировалась на основных функциях MES, Международное общество автоматизации (ISA) приняло решение о необходимости в согласованной терминологии и согласованной информационной модели для определения и интеграции действий между предприятием и системами управления – поэтому в конце 90-х они разработали стандарт ISA-95. Благодаря стандартизации терминологии ISA-95 упрощает эффективную коммуникацию между заинтересованными сторонами, такими как поставщики и производители. В свою очередь, согласованные модели снижают риск ошибок при интеграции производственных площадок с бизнес-системами.

ISA-95 определяет интерфейс между функциями управления и предприятия для создания уровней технологии и бизнес-процессов. Упрощенная модель этой иерархии помещает системы управления производством на третий уровень, между бизнес-планированием, логистикой и системами управления процессами:

- Уровень 4 – ERP: бизнес-планирование и логистика
- Уровень 3 – MES: управление производственными операциями
- Уровень 2 – Системы управления технологическими процессами: пакетный контроль
- Уровень 1 – Системы управления технологическими процессами: непрерывный контроль
- Уровень 0 – Системы управления технологическими процессами: дискретное управление.

Упрощенная модель систем управления производством представлена на рисунке 1.

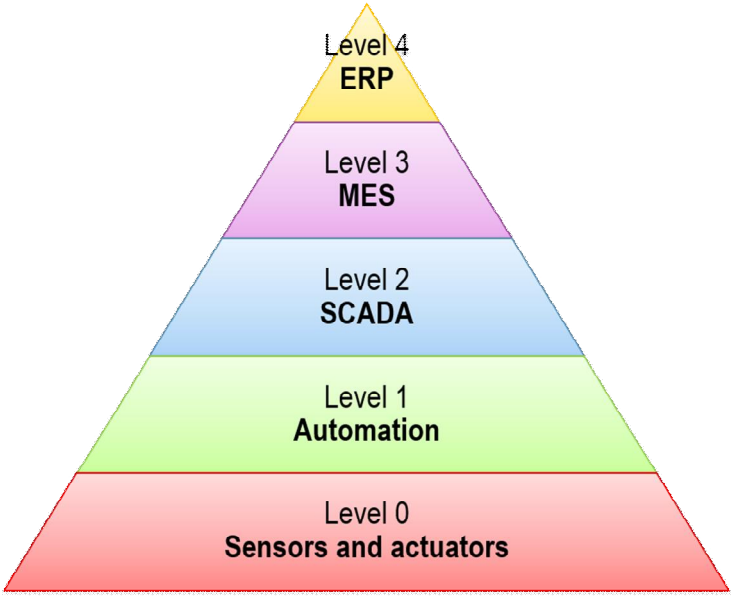


Рисунок 1 – упрощенная модель взаимодействия систем управления согласно стандарту ISA-95

MES-система предоставляет возможность расчёта ряда качественных показателей:

1. Оценка реальной эффективности. Программа собирает данные об объеме произведенных товаров пришедших на склад, оценивается алгоритм их обработки и рассчитываются индексы основных показателей.

2. Контроль работы персонала. MES собирает информацию о работе сотрудников: количество отработанного времени, выполненные задания, время простоя.

3. Измерение уровня качества. С помощью датчиков, установленных на оборудовании, происходит сбор информации о ходе производства. Система обрабатывает полученные данные, производит сравнение с заданными стандартами и выдает оценку интересующих параметров, что позволяет вносить изменения в процесс и наблюдать изменения результата.

4. Создание отчетов. Программа собирает информацию о различных качественных показателях и формирует на их основании

отчет, что позволяет уменьшить объём бюрократии и снизить документооборот.

5. Планирование. Система имеет возможность представлять данные в разрезе нескольких показателей, что облегчает процесс планирования.

6. Логистика. MES имеет возможность отслеживать расположение конкретной единицы продукции во всем цикле производства, хранения или отправки заказчику [3].

Внедрение и эксплуатация MES-системы могут быть сопряжены с определенными рисками и проблемами. Список самых распространённых приведен ниже:

Комплексное внедрение: Процесс внедрение системы MES является ресурсозатратным и трудоемким. Он требует значительного планирования, интеграции с существующими системами и настройки в соответствии с конкретными производственными процессами. Сложность внедрения может привести к задержкам, перерасходу бюджета и потенциальным сбоям на переходном этапе. Ассоциация решений для производственных предприятий (MESA) — это глобальное некоммерческое сообщество, призванное помогать компаниям-членам успешно внедрять и применять системы MES и связанные с ними методологии.

Интеграция данных: Интеграция системы MES с другими системами, такими как ERP, PLM или SCM, может оказаться сложной задачей. Обеспечение бесперебойного обмена данными и синхронизации между различными системами может потребовать обширных усилий по настройке, сопоставлению данных и интеграции. Несоответствия данных или ошибки во время интеграции способны приводить к неточной информации и препятствовать эффективному принятию решений.

Управление изменениями: Внедрение безбумажной системы MES часто требует изменений в бизнес-процессах, рабочих процессах и ролях в организации. Сопротивление изменениям со стороны сотрудников или заинтересованных сторон может создавать проблемы и влиять на принятие пользователями. Правильные стратегии управления изменениями, включая обучение и коммуникацию, имеют решающее значение для преодоления сопротивления и помогают обеспечить плавный переход и принятие системы MES.

Безопасность данных: Системы MES обрабатывают большие объемы конфиденциальных производственных данных, включая интеллектуальную собственность, параметры процесса и информацию

о качестве. Помощь в обеспечении безопасности данных и защита от несанкционированного доступа или киберугроз имеет первостепенное значение. Для снижения рисков должны быть приняты надежные меры безопасности, такие как шифрование данных, контроль доступа пользователей и сетевые гарантии.

Важно учитывать вышеуказанные проблемы при планировании внедрения MES-системы. Однако, следует отметить, что все преобразования сопряжены с подобными проблемами, и отсутствие системы MES может привести к неэффективности, снижению производительности и конкурентоспособности.

В последние годы эволюцию систем MES определяют несколько тенденций цифровой трансформации.

Облачная MES: Облачные вычисления набирают обороты в обрабатывающей промышленности, и программное обеспечение MES все чаще предлагается в качестве облачного решения. Облачные MES-системы обеспечивают такие преимущества, как масштабируемость, гибкость, снижение затрат на инфраструктуру и упрощение доступа к данным из нескольких местоположений. Они также облегчают интеграцию с другими облачными приложениями и обеспечивают совместную работу в режиме реального времени.

Интеграция IIoT: Интеграция систем MES с промышленным Интернетом вещей (IIoT) является важной тенденцией. Системы MES используют технологии IIoT для сбора данных с датчиков, машин и подключенных устройств в режиме реального времени. Данная интеграция обеспечивает улучшенную видимость, прогнозную аналитику, удаленный мониторинг и оптимизацию производственных процессов.

Аналитика больших данных: Системы MES используют возможности аналитики больших данных для обработки и анализа огромных объемов данных, генерируемых в ходе производственных операций. Для выявления закономерностей, корреляций и практических выводов применяются передовые алгоритмы аналитики и методы машинного обучения. Это помогает оптимизировать производство, повысить качество и обеспечить профилактическое обслуживание.

Мобильные приложения: Мобильные приложения все чаще интегрируются в системы MES, обеспечивая доступ к данным и функциям в режиме реального времени со смартфонов и планшетов. Операторы, супервайзеры и менеджеры имеют возможность отслеживать производственные процессы и управлять ими, просматривать производственные информационные панели и получать

уведомления на своих мобильных устройствах. Мобильные приложения MES повышают оперативность работы и позволяют принимать решения на ходу.

Интеграция с системами цепочки поставок: Системы MES расширяют свои возможности интеграции для подключения к системам цепочки поставок. Эта интеграция обеспечивает бесперебойный информационный поток между системами MES, ERP и SCM, улучшая видимость цепочки поставок, планирование спроса и синхронизацию, что, в свою очередь, способствует улучшению сотрудничества с поставщиками, заказчиками и партнерами по логистике.

Искусственный интеллект и машинное обучение: технологии искусственного интеллекта и машинного обучения применяются в системах MES для автоматизации принятия решений, оптимизации процессов и обеспечения возможностей прогнозирования. Алгоритмы искусственного интеллекта могут анализировать исторические данные, выявлять аномалии, прогнозировать производственные результаты и рекомендовать улучшения процессов. Эта тенденция позволяет системам MES предоставлять интеллектуальную информацию в режиме реального времени и поддерживать принятие решений на основе данных.

Усовершенствованные пользовательские интерфейсы и визуализация: Системы MES ориентированы на предоставление интуитивно понятных пользовательских интерфейсов и расширенных возможностей визуализации. Интерактивные панели мониторинга, 3D-визуализация, дополненная реальность и виртуальная реальность внедряются для улучшения взаимодействия с пользователем и облегчения лучшего понимания производственных данных. Это позволяет операторам и менеджерам быстро выявлять тенденции, аномалии и области, требующие улучшения.

Соответствие требованиям и возможности регулирования: В связи с ужесточением правил в различных отраслях системы MES включают в себя более надежные функции обеспечения соответствия требованиям и регулирования. Сюда входят функции электронного ведения записей, журналов аудита, управления документацией и соблюдения отраслевых стандартов и нормативных актов. Системы MES играют важную роль в обеспечении соответствия нормативным актам.

Эти тенденции отражают нацеленность отрасли на использование передовых технологий, возможностей подключения и анализа данных для повышения эффективности, гибкости и принятия

решений в производственных операциях. Внедрение этих тенденций в системы MES может предоставить производителям конкурентные преимущества.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Андреев Е.Б., Куцевич И.В., Куцевич Н.А. MES-системы: взгляд изнутри: Издательство «РТСофт» – «Космоскоп», 2015. – 240 с.

2. Ассоциация MES [Электронный ресурс]. URL:https://mesa.org/

3. Фролов Е. Б., Загидуллин Р. Р. MES-системы как они есть, или эволюция систем планирования производства // Металлообрабатывающее оборудование. – 2008.-№10. – с.31-37.

УДК 004.65

АРТЮХ Е.О.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

РАЗРАБОТКА МЕТОДОВ И АЛГОРИТМОВ ПРЕОБРАЗОВАНИЯ АУДИО ИНФОРМАЦИИ

Статья посвящена методам и алгоритмам преобразования аудио информации. В ней рассматриваются основные методы, такие как цифровая обработка сигналов, компрессия аудио данных, анализ аудио сигналов и синтез аудио сигналов. Также в статье рассматриваются основные алгоритмы преобразования аудио информации, такие как фильтрация, кодирование, компрессия и модуляция.

Введение

Преобразование аудио информации является одной из важных задач в области обработки аудио сигналов. Оно включает в себя различные методы и алгоритмы, которые позволяют преобразовывать, модифицировать и анализировать аудио данные. В данной статье мы рассмотрим некоторые из наиболее важных методов и алгоритмов, используемых в этой области.

Основные методы преобразования аудио информации

«Аудиоинформация» относится к данным или подробностям о звуке или аудиозаписях. Это может включать в себя различные аспекты, такие как формат аудиофайла, метод оцифровки звука, структуру данных и характеристики звукозаписи, такие как частота дискретизации и разрядность. Кроме того, аудиоинформация может также относиться к метаданным, связанным с аудиофайлами, которые

могут включать такие сведения, как исполнитель, альбом, номер дорожки и другую соответствующую информацию.

Цифровая обработка сигналов. Цифровая обработка сигналов (ЦОС) является основным методом преобразования аудио информации. Она позволяет преобразовывать аналоговые сигналы в цифровые и наоборот, а также модифицировать цифровые сигналы с помощью различных алгоритмов. ЦОС включает в себя такие методы, как фильтрация, интерполяция, децимация, корреляция, конволюция и т.д.

Фильтрация. Фильтрация является одним из основных методов ЦОС. Она позволяет удалять нежелательные частоты из аудио сигналов, такие как шумы или помехи. Существуют различные типы фильтров, такие как низкочастотные фильтры, высокочастотные фильтры, полосовые фильтры и т.д.

Интерполяция. Интерполяция является методом ЦОС, который позволяет увеличивать частоту дискретизации аудио сигналов. Это может быть полезно для улучшения качества аудио сигналов или для соответствия требованиям определенных приложений.

Децимация. Децимация является методом ЦОС, который позволяет уменьшать частоту дискретизации аудио сигналов. Это может быть полезно для уменьшения размера аудио файлов или для соответствия требованиям определенных приложений.

Корреляция. Корреляция является методом ЦОС, который позволяет определять схожесть двух сигналов. Это может быть полезно для идентификации аудио сигналов или для выделения полезной информации из аудио сигналов.

Конволюция. Конволюция является методом ЦОС, который позволяет комбинировать два сигнала для получения нового сигнала. Это может быть полезно для модификации аудио сигналов или для создания новых аудио сигналов.

Анализ аудио сигналов. Анализ аудио сигналов является важным методом преобразования аудио информации. Он позволяет извлекать полезную информацию из аудио сигналов, такую как частота, амплитуда, длительность и т.д. Анализ аудио сигналов может быть реализован с помощью различных методов, таких как Фурье-анализ, огибающая амплитуды, спектрограммы и т.д.

Фурье-анализ. Фурье-анализ является методом анализа аудио сигналов, который позволяет получить спектральную характеристику аудио сигнала. Это может быть полезно для определения частотного состава аудио сигнала.

Огибающая амплитуды. Огибающая амплитуды является методом анализа аудио сигналов, который позволяет получить форму аудио сигнала, не зависящую от его амплитуды. Это может быть полезно для определения длительности аудио сигнала.

Спектрограммы. Спектрограммы являются методом анализа аудио сигналов, который позволяет получить графическое представление спектральной характеристики аудио сигнала в зависимости от времени. Это может быть полезно для визуализации изменений частотного состава аудио сигнала во времени.

Синтез аудио сигналов. Синтез аудио сигналов является важным методом преобразования аудио информации. Он позволяет создавать новые аудио сигналы на основе существующих или на основе набора параметров. Синтез аудио сигналов может быть реализован с помощью различных методов, таких как синтез форм волн, синтез на основе таблиц, синтез на основе моделей и т.д.

Синтез форм волн. Синтез форм волн является методом синтеза аудио сигналов, который позволяет создавать новые аудио сигналы на основе различных форм волн, таких как синусоидальные волны, пилообразные волны, зубчатые волны и т.д.

Синтез на основе таблиц. Синтез на основе таблиц является методом синтеза аудио сигналов, который позволяет создавать новые аудио сигналы на основе набора значений амплитуды в зависимости от времени. Это может быть полезно для создания аудио сигналов с определенной формой.

Синтез на основе моделей. Синтез на основе моделей является методом синтеза аудио сигналов, который позволяет создавать новые аудио сигналы на основе моделей звукообразующих органов, таких как губы, нос, гортань и т.д. Это может быть полезно для создания аудио сигналов, имитирующих звуки человеческой речи.

Основные алгоритмы преобразования аудио информации:

Фильтрация. Фильтрация является одним из основных алгоритмов преобразования аудио информации. Она позволяет удалять нежелательные частоты из аудио сигналов, такие как шумы или помехи. Существуют различные типы фильтров, такие как низкочастотные фильтры, высокочастотные фильтры, полосовые фильтры и т.д.

Низкочастотный фильтр. Низкочастотный фильтр является типом фильтра, который позволяет пропускать низкие частоты и блокировать высокие частоты аудио сигнала. Он может быть использован для удаления шума из аудио сигнала.

Высокочастотный фильтр. Высокочастотный фильтр является типом фильтра, который позволяет пропускать высокие частоты и блокировать низкие частоты аудио сигнала. Он может быть использован для удаления баса из аудио сигнала.

Полосовой фильтр. Полосовой фильтр является типом фильтра, который позволяет пропускать частоты в определенном диапазоне и блокировать частоты вне этого диапазона. Он может быть использован для удаления определенных частот из аудио сигнала.

Кодирование. Кодирование является важным алгоритмом преобразования аудио информации. Оно позволяет преобразовывать аудио сигналы в цифровой формат, который может быть легко передан или храниться. Кодирование может быть реализовано с помощью различных методов, таких как pulse-code modulation (PCM), differential pulse-code modulation (DPCM), adaptive differential pulse-code modulation (ADPCM) и т.д.

PCM. PCM является методом кодирования аудио сигналов, который позволяет преобразовывать аналоговые сигналы в цифровые сигналы путем дискретизации амплитуды сигнала в определенные моменты времени. Это позволяет сохранять точность аудио сигналов, но может привести к значительному увеличению размера аудио файлов.

DPCM. DPCM является методом кодирования аудио сигналов, который позволяет преобразовывать аналоговые сигналы в цифровые сигналы путем дискретизации изменений амплитуды сигнала в определенные моменты времени. Это может позволить уменьшить размер аудио файлов по сравнению с PCM, но может привести к незначительному ухудшению качества аудио сигналов.

ADPCM. ADPCM является методом кодирования аудио сигналов, который позволяет преобразовывать аналоговые сигналы в цифровые сигналы путем дискретизации изменений амплитуды сигнала в определенные моменты времени с использованием адаптивной коэффициент дискретизации. Это может позволить уменьшить размер аудио файлов по сравнению с DPCM, но может привести к незначительному ухудшению качества аудио сигналов.

Компрессия. Компрессия является важным алгоритмом преобразования аудио информации. Она позволяет уменьшить размер аудио файлов, сохраняя при этом их качество. Компрессия может быть с потерями или без потерь. Компрессия с потерями является методом компрессии аудио данных, при котором уменьшается размер аудио файлов за счет удаления части информации. Это может привести к ухудшению качества аудио сигналов, но может позволить значительно

уменьшить размер аудио файлов. Компрессия без потерь является методом компрессии аудио данных, при котором уменьшается размер аудио файлов за счет использования специальных алгоритмов, не удаляющих ни одной информации. Это позволяет сохранять точность аудио сигналов, но может привести к меньшему уменьшению размера аудио файлов.

Модуляция. Модуляция является важным алгоритмом преобразования аудио информации. Она позволяет изменять параметры аудио сигналов, такие как частота, амплитуда или фаза. Модуляция может быть реализована с помощью различных методов, таких как амплитудная модуляция (AM), частотная модуляция (FM) и фазовая модуляция (PM).

Амплитудная модуляция (AM). Амплитудная модуляция является методом модуляции, которые позволяет изменять амплитуду аудио сигнала в зависимости от модулирующего сигнала. Это может быть полезно для передачи информации в аудио канале.

Частотная модуляция (FM). Частотная модуляция является методом модуляции, который позволяет изменять частоту аудио сигнала в зависимости от модулирующего сигнала. Это может обеспечить лучшую чёткость и стабильность аудио сигнала по сравнению с AM.

Фазовая модуляция (PM). Фазовая модуляция является методом модуляции, который позволяет изменять фазу аудио сигнала в зависимости от модулирующего сигнала. Это может обеспечить лучшую чёткость и стабильность аудио сигнала по сравнению с AM и FM.

Заключение

Преобразование аудио информации является важной задачей в области обработки аудио сигналов. В данной статье мы рассмотрели некоторые из наиболее важных методов и алгоритмов, используемых в этой области. Цифровая обработка сигналов, компрессия аудио данных, анализ аудио сигналов и синтез аудио сигналов являются некоторыми из основных методов преобразования аудио информации. Фильтрация, кодирование, компрессия и модуляция являются некоторыми из основных алгоритмов преобразования аудио информации. Хотя мы ограничились только описанием этих методов и алгоритмов, реализация их может потребовать использования сложных математических формул и алгоритмов, а также специализированных программных и аппаратных средств.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Smith S. Digital signal processing: a practical guide for engineers and scientists. – Newnes, 2003.

2. Proakis J. G. Digital signal processing: principles, algorithms, and applications, 4/E. – Pearson Education India, 2007.

4. Тропченко А. Ю., Тропченко А. А. Цифровая обработка сигналов. Методы предварительной обработки. Учебное пособие по дисциплине" Теоретическая информатика" //СПб: СПбГУ ИТМО. – 2009. – Т. 100.

5. Оппенгейм А., Шафер Р. Цифровая обработка сигналов. – Litres, 2022.

6. Рабинер Л.Р., Шафер Р. В. Цифровая обработка речевых сигналов. – Рипол Классик, 1981.

УДК 004.65

БАСТРЫЧКИН А.С., МЕЛИХОВ А.Ю., ЛАЗАРЕВ И.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ИССЛЕДОВАНИЕ РАБОТЫ БРОКЕРОВ СООБЩЕНИЙ ДЛЯ АСИНХРОННОЙ ПЕРЕДАЧИ НА ПРИМЕРЕ РЕАЛИЗАЦИИ APACHE KAFKA

Проводится исследование внутреннего устройства работы брокера сообщений для асинхронной передачи Apache Kafka. Описываются главные структурные элементы брокера и их функции.

На сегодняшний день можно заметить тенденцию к усложнению программных систем: они становятся все более масштабными и состоят из множества отдельных компонентов. Внутри системы необходимо обеспечивать взаимодействие между компонентами, и для этого существует разнообразие методов коммуникации. Однако особое место среди них занимают брокеры сообщений.

Брокер сообщений представляет собой форму архитектуры, при которой элементы системы взаимодействуют друг с другом через посредника. Это позволяет разгрузить веб-сервисы, так как они освобождаются от необходимости управлять пересылкой сообщений - эту функцию выполняет брокер.

Можно сказать, что в работе брокера сообщений используются две основные роли: "продюсер" (создатель сообщений) и "консьюмер" (потребитель/подписчик). Первый создает и отправляет сообщения

второму. Существует также промежуточная точка - файловая система, где хранятся сообщения от продюсера в процессе отправки.

Для чего нужны брокеры сообщений?

1. Для организации связи между службами, даже если одна из них временно неактивна. Продюсер может отправлять сообщения, независимо от активности потребителя.

2. Для повышения производительности системы благодаря асинхронной обработке задач.

3. Для обеспечения надежной доставки сообщений: брокеры обычно предоставляют механизмы повторной отправки сообщений в случае неудачи и соответствующую маршрутизацию недоставленных сообщений.

Недостатки брокеров сообщений:

1. Усложнение системы и зависимость от надежности распределенной сети.

2. Возможность ошибок из-за асинхронной и распределенной природы системы.

3. Сложность освоения таких систем, требующая значительного времени.

Когда брокеры сообщений могут быть полезны:

1. При выполнении длительных и ресурсоемких задач, не требующих мгновенного результата.

2. В микросервисной архитектуре для координации сервисов.

3. В мобильных приложениях для использования push-уведомлений.

4. В транзакционных системах, где каждый этап обработки состоит из отдельных шагов, выполняемых различными элементами системы.

Рассмотрим брокер сообщений Apache Kafka.

Apache Kafka, открывший двери в мир обработки потоковых данных, стал неотъемлемой частью современной архитектуры данных и приложений. Эта платформа, созданная в LinkedIn и открытая для общественности, предоставляет эффективные решения для передачи, обработки и анализа данных в режиме реального времени.

Ключевые характеристики Apache Kafka

Масштабируемость и Отказоустойчивость

Одной из главных привлекательных особенностей Kafka является её способность масштабироваться и обеспечивать высокую отказоустойчивость. Кластер Kafka состоит из нескольких брокеров (серверов), способных автоматически обрабатывать сбои в случае

отказа одного из узлов. Это гарантирует непрерывность работы системы даже в условиях нештатных ситуаций.

Потоковая Обработка

Kafka работает на основе потоковой обработки данных, что позволяет ей эффективно передавать и обрабатывать данные в режиме реального времени. Эта архитектура идеально подходит для приложений, где требуется мгновенная реакция на изменения данных, таких как системы мониторинга, аналитические приложения и др.

Producer-Consumer Модель

Ключевой концепцией в Kafka является модель "производитель-потребитель" (Producer-Consumer). Producer отвечает за размещение событий (или записей) в Kafka, в то время как Consumer подписывается на эти события и обрабатывает их. Эта модель обеспечивает гибкость в построении приложений, где различные компоненты могут эффективно обмениваться данными (рисунок 1).

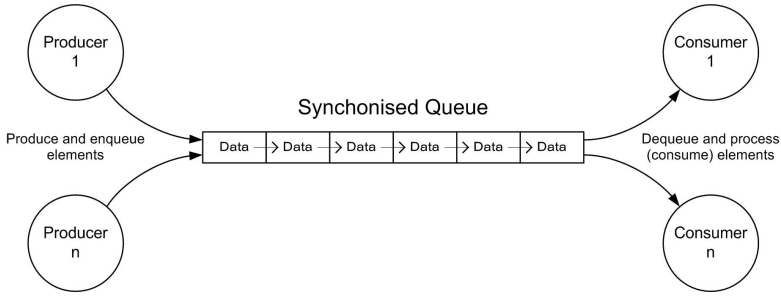


Рисунок 1 – Пример модели Producer-Consumer

1. Брокеры (Brokers):

Брокеры в Apache Kafka представляют собой серверы, отвечающие за обработку, хранение и передачу данных. Брокеры действуют в составе кластера, обеспечивая горизонтальную масштабируемость и отказоустойчивость системы. В кластере Kafka может быть несколько брокеров, и они автоматически согласовывают свою работу для обеспечения целостности данных.

2. Топики (Topics):

Топики в Kafka представляют собой категории, в которых группируются записи или сообщения. Producer направляет свои сообщения в конкретные топики, а Consumer подписывается на топики для чтения и обработки данных. Топики помогают организовать данные и обеспечивают логическое разделение информации.

3. Партиции (Partitions):

Партиции представляют собой фрагменты топика, которые распределяются по брокерам в кластере. Каждая партиция содержит упорядоченный и неделимый лог записей. Распределение данных на партиции обеспечивает параллельную обработку сообщений и увеличивает пропускную способность системы.

4. Producer (Производитель, Издатель):

Producer ответственен за создание и отправку сообщений в топики. Он формирует записи и направляет их в соответствующий топик в брокеры Kafka. Producer может отправлять сообщения в различные партиции топика, что обеспечивает распределение нагрузки и повышает производительность.

5. Consumer (Потребитель, Подписчик):

Consumer подписывается на топики и обрабатывает поступающие оттуда сообщения. Consumer может читать данные из нескольких партиций одновременно, что позволяет распараллеливать обработку. Он следит за смещением (offset) в каждой партиции, чтобы обеспечивать надежность доставки и сохранение порядка сообщений.

6. Zookeeper:

Zookeeper используется для управления координацией и обнаружением брокеров в кластере Kafka. Он отслеживает состояние брокеров, хранит метаданные, такие как информация о топиках и их партициях, а также обеспечивает выбор лидера для каждой партиции. Zookeeper обеспечивает целостность и согласованность кластера. (рисунок 2).

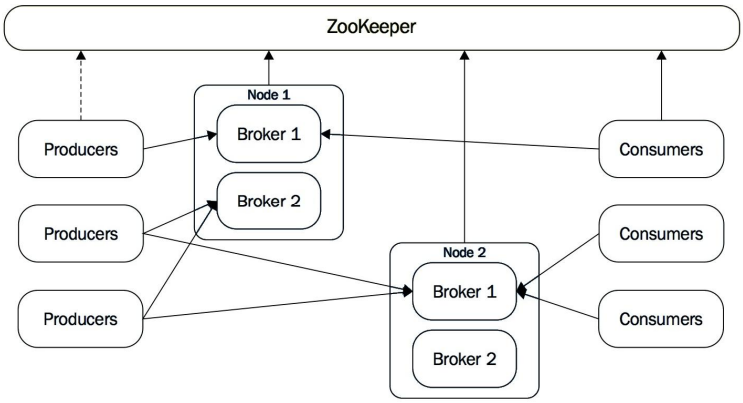


Рисунок 2 – Модель взаимодействия Zookeeper’а

1. Масштабируемость и Отказоустойчивость:

Kafka обеспечивает горизонтальную масштабируемость. Кластер может включать сотни и тысячи брокеров для обработки высоких нагрузок. В случае сбоя одного брокера, другие брокеры автоматически принимают на себя его обязанности, гарантируя непрерывную работу системы.

2. Потоковая Обработка:

Kafka Streams API предоставляет возможность разработки потоковых приложений, позволяя анализировать и преобразовывать данные в режиме реального времени. Это полезно для создания приложений, обрабатывающих потоки событий.

3. Гарантии Доставки и Порядка:

Kafka обеспечивает строгие гарантии доставки сообщений (at-least-once и exactly-once semantics) и сохранение порядка сообщений внутри топиков. Это особенно важно для приложений, где сохранение порядка и гарантии доставки являются критическими аспектами.

4. Репликация и Хранение Данных:

Кластер Kafka поддерживает репликацию топиков для обеспечения отказоустойчивости. Данные хранятся на диске, что позволяет Kafka эффективно обрабатывать как потоковые, так и хранимые данные.

Процесс работы Apache Kafka можно описать следующим образом, уделяя внимание каждому этапу:

1. Запуск и Конфигурация Брокеров:

- Каждый узел в кластере Kafka представляет собой брокер. Процесс запускает брокеры с учетом их конфигурации.

- Конфигурация брокеров включает в себя параметры, такие как идентификатор брокера, адреса других брокеров, порты для прослушивания входящих соединений, а также настройки хранения данных и другие параметры.

2. Создание и Конфигурация Топиков:

- Разработчик определяет топики, в которых будут храниться данные. Это может быть выполнено с помощью команды Kafka CLI или API администрирования.

- При создании топика указываются параметры, такие как количество партиций, репликации, а также настройки хранения сообщений.

3. Отправка Сообщений Producer'ом:

- Producer создает сообщение и отправляет его в один из топиков. Producer может явно указать партицию для отправки или доверить системе автоматический выбор партиции.

- Producer может использовать различные стратегии отправки, например, синхронную или асинхронную, в зависимости от требований приложения.

4. Хранение и Распределение Данных:

- Kafka хранит данные в виде лога (log) для каждой партиции топика. Каждое сообщение в партиции имеет уникальный номер, называемый смещением (offset).

- Система распределяет сообщения между различными брокерами и их партициями для обеспечения масштабируемости и отказоустойчивости.

5. Подписка и Чтение Consumer'ом:

- Consumer подписывается на топики и начинает читать сообщения из партиций. Он сохраняет смещение, чтобы знать, какие сообщения уже были обработаны.

- Consumer может работать в рамках группы, где каждый потребитель в группе обрабатывает свои уникальные партиции, обеспечивая балансировку нагрузки.

6. Zookeeper и Координация:

- Zookeeper отслеживает состояние брокеров, хранит метаданные топиков, а также управляет выбором лидера для каждой партиции.

- В более новых версиях Apache Kafka, зависимость от Zookeeper постепенно снижается, и Kafka самостоятельно управляет своей метаданными.

7. Обработка Событий в Реальном Времени:

- Kafka обеспечивает обработку событий в реальном времени, что позволяет быстро реагировать на поступающие данные и обрабатывать их сразу после отправки.

8. Гарантии Доставки и Масштабирование:

- Kafka гарантирует доставку сообщений с определенной степенью надежности (at-least-once или exactly-once).

- Система масштабируется горизонтально, позволяя добавлять новые брокеры и расширять кластер для обработки растущей нагрузки (пример изображен на рисунке 3).

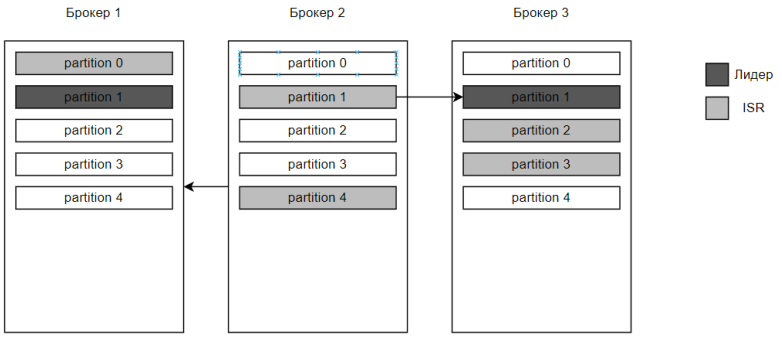


Рисунок 3 – Пример масштабирования

Процесс работы Kafka предоставляет эффективные средства для обработки потоков данных в реальном времени с гарантированными свойствами порядка, надежности и масштабируемости.

Изучив внутреннее устройство работы Apache Kafka можно заключить, что данная реализация брокера сообщений с асинхронной передачей является довольно надёжной системой, с применением которой можно создавать такие же надёжные сервисы.

Создавая системы с передачей сообщений по Kafka можно быть уверенным, что система обработает их все.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Дилан Скотт, Виктор Гамов, Дейв Клейн. С44 Kafka в действии / пер. с англ. А. Н. Киселева. – М.: ДМК Пресс, 2022. – 310 с.

УДК 004.65

БОЩЕНКО А.Р.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина.

ТИПЫ И КОМПОНЕНТЫ АРХИТЕКТУР СОВРЕМЕННЫХ ВЕБ-ПРИЛОЖЕНИЙ

Рассматриваются перспективные разновидности к подходу организации архитектуры веб-приложений. Описывается структура и все её компоненты с примерами различных технологий разработки.

Веб-приложение – это клиент-серверное приложение с веб-

интерфейсом, в котором есть браузер (клиент) и веб-сервер. Логика веб-приложения распределена между сервером и клиентом, есть канал для обмена информацией и хранилище данных, расположенное локально или в облаке. Удобство такого приложения обусловлено тем, что такую системы легче администрировать, реализовывать в ней хранение данных, управлять доступами к той или иной информации. Пользователи приложения могут подключиться с любой точки мира через сеть интернет, можно создать закрытую систему только для пользователей, подключенный к локальной сети. Отличительной особенностью веб-приложения от веб-сайта являются аутентификация, интеграция и интерактивность.

Каким стеком технологий нужно обладать, чтобы написать свое веб-приложение? Первое, что нужно изучить, это технологии front-end разработки. Язык гипертекстовой разметки HTML, именно его понимают браузеры интерпретируя в интерфейс. Каскадная таблица стилей CSS как дополнение к коду HTML, имеет тот же синтаксис, но предназначена для красивого оформления нашей страницы.

Язык программирования JavaScript, это специально созданный язык для реализации действий (скриптов) на HTML-страницах. В эпоху, когда было уже недостаточно просто статичной текстовой страницы в сети интернет, которую можно было просто почитать, сообщество программистов решило, что нужен инструмент, который будет слушать действия пользователя и реагировать на него, изменяя каким-либо образом внешний вид сайта. В последние года набирает популярность надстройка над языком JavaScript – TypeScript. Этот язык был разработан корпораций Microsoft в 2012 году. Он расширяет возможности JavaScript добавлением в язык явной статической типизации, в нем объекты имеют вид классических объектов объектно-ориентированного программирования, также появилась поддержка подключения модулей. React – JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

Знание сетевого протокола передачи данных в сети-интернет HTTP или защищенный протокол передачи данных HTTPS. Знание методов запроса к ресурсу таких, как GET-запрос, PUT-запрос, DELETE-запрос, POST-запрос.

Знание интерфейса API. API – протокол договоренностей о том, как между собой будут общаться программы. Содержит в себе протокол передачи данных, формат данных и модель данных. Есть несколько типов построения API:

– *SOAP* (Simple Object Access Protocol) – обмен структурированными сообщениями между веб-сервисами, формат

данных XML;

– *RPC (Remote Procedure Call)* – метод взаимодействия между компонентами распределённой системы, позволяющий вызывать функции на сервере, как будто они являются локальными, формат данных XML, JSON и бинарные;

– *WebSocket API* – двунаправленный протокол связи, позволяет установить постоянное соединение между клиентом и сервером для обмена данными, формат данных может быть текстовым или бинарным, в том числе XML и JSON;

– *Rest (Representational State Transfer)* – одна из самых популярных в последние годы подходов к написанию запросов. Работает поверх HTTPS, эффективно используя все его свойства.

Для блока серверной части приложения нужно знать любой язык программирования и фреймворки, которые могут облегчить сборку и запуск нашего приложения на сервере. Рассмотрим более подробно Java-технологии. Конечно, любой разработчик должен отлично знать Java Core. Также важно владеть и понимать спецификацию, предложенную комьюнити Java, под названием Servlet API. Данная спецификация основана на Java-технологии, если точнее, это Java-класс, скомпилированный в байт-код. Servlet API состоит из 2 пакетов, а именно, javax.servlet javax.servlet.http. Стек фреймворка Spring, то есть все его модули, такие как Core, Web, Data Access, Test. А также проекты Spring Data, Spring Boot, Spring Batch. Данный фреймворк по сути отвечает за внедрение зависимостей с некоторыми удобными фишками, как взаимодействие с СУБД, прокси, аспектно-ориентированное программирование, веб-инфраструктура MVC)

Необходимо понимание работы СУБД, знание языка SQL, необходимо иметь представление о технологии JDBC для доступа к базам данных из программ на языке Java.

Архитектура веб-приложения – это высокоуровневая структура, описывающая, как различные компоненты веб-приложения взаимодействуют друг с другом с целью повышения функциональности и результативности продукта для бизнеса. На этапе выбора архитектуры важно не потеряться в огромном разнообразии их типов, ведь неправильный выбор может нанести ущерб бизнесу.

Современные веб-приложения используют концепцию трехуровневой архитектуры, которая разделяет приложения на уровень представления, уровень приложений и уровень данных. Трехуровневая схема архитектуры веб-приложения более безопасна, поскольку клиент не имеет прямого доступа к данным. В рамках трехуровневой архитектуры веб-приложения каждый уровень работает

на своей собственной инфраструктуре и может разрабатываться параллельно разными командами. Такая структура позволяет обновлять и масштабировать каждый уровень по мере необходимости, не затрагивая другие уровни.

Типы архитектур веб-приложений

Рендеринг на стороне сервера (Server-Side Rendering) – это метод отрисовки веб-страницы на сервере, а не в браузере. Сервер отдает сгенерированную HTML-страницу пользователю, пока основное веб-приложение загружается и запускается. Рендеринг на стороне сервера Vue.js Приложение также можно считать "изоморфным" или "универсальным" в том смысле, что большая часть кода вашего приложения выполняется как на сервере, так и на клиенте. Статическая веб-страница, используя JavaScript, отправляет запрос серверу (API), сервер возвращает клиенту HTML-страницу. В случае если на новой странице есть только несколько элементов, отличных от текущей, браузер всё равно запросит всю страницу. Для кэширования страницы используется NGINX. Этот способ позволяет значительно ускорить процесс загрузки страницы, не обращаясь к бэкэнду.

Если данный тип сравнивать с SPA (клиентское одностраничное приложение), то у него есть ряд преимуществ:

– время обработки контента более низкое, данное свойство хорошо заметно, когда очень медленный доступ в интернет или медленное устройство;

– унифицированная ментальная модель данных. То есть не нужно будет метаться от языка к языку, на котором устроен ваш внешний интерфейс и внутренний сервер;

– Улучшение SEO: поисковые роботы будут видеть полностью отрисованную страницу напрямую.

Рендеринг на стороне клиента (Client-Side Rendering) – страница отображается на компьютере в браузере. При рендеринге на стороне клиента (CSR) получается простой HTML-документ с информацией о стилях CSS и файлом JavaScript, который будет отображать остальную часть сайта с помощью браузера. Это радикально отличается от использования SSR. Сервер отвечает только за загрузку HTML и CSS, всё остальное обрабатывается библиотекой JavaScript на стороне клиента. Имейте в виду, что CSR может повлиять на SEO. Некоторые поисковые роботы могут не выполнять JavaScript и, следовательно, видеть только начальное состояние пустоты или загрузки вашего приложения. Это также может привести к проблемам с производительностью у пользователей с более медленными интернет-соединениями или устройствами, поскольку им нужно дождаться

загрузки и запуска всего JavaScript, прежде чем они смогут увидеть всю страницу целиком.

Статическая генерация сайта (Static Site Generation) – предоставляет пользователю через API по запросу сгенерированную HTML-страницу, расположенную на CDN или любом другом сервере. Не требует пересоздания страницы, т.к. используется статический шаблон. Генерирует сайт на основе необработанных данных и набора шаблонов. Т.к. страницы заранее созданы, это приводит к значительному ускорению загрузки в браузере.

К плюсам можно отнести высокую производительность, разгруженный сервер, т.к. статические страницы не занимают много памяти. К минусам явно можно определить такие факты, как наличие только нескольких шаблонов (вообще отсутствие такового), тесть разработчику придется потратить время вначале на создание страниц, со стороны пользователя сложнее публиковать контент с помощью генератора статических сайтов, скорее всего придется прибегать к помощи разработчиков.

Одностраничное приложение (Single Page Application) – при работе загружает только одну страницу, а когда нужно поменять содержимое обращается к API JavaScript для отображения нового содержимого. В данном типе проектирования весь код, состоящий из HTML, CSS, JS, либо добавляется по мере необходимости, либо извлекается браузером. Это позволяет пользователю взаимодействовать с веб-приложением не загружая целые новые страницы с сервера, что потенциально повышает производительность.

*Прогрессивное веб-приложение (Progressive W*eb *App)* – строится на логике одностраничного веб-приложения, но с сервисами, которые запускаются в браузере. В данном типе архитектуры нужно учитывать, что браузер и операционная система поддерживает набор указанных стандартов. Со стороны пользователя такое веб-приложение просит добавить себя на экран операционной системы. Приложение автоматически добавляется на устройство, когда пользователь дает свое согласие в сплывающем окне браузера. За счет такого подхода обеспечивается автономность приложения, фоновая синхронизация и допуск push-уведомлений.

Интерфейс MICRO (micro front-end) – технология подразумевает разделение монолитного веб-приложения на микросервисы, которые запущены на отдельных серверах. Каждое мини-приложение отвечает за определенный функционал. Благодаря такому подходу реализовывается распределение нагрузки. Микросервисы могут

обращаться друг к другу посредством HTTP-запросов. Для пользователя все они находятся на одной страницы.

Описание структуры современного веб-приложения

Уровень представления (Front-end) – всё с чем взаимодействует пользователь приложения. В современном мире есть два основных вида фронтенда: веб-интерфейсы и мобильные приложения. Первый вид реализуются в основном с использованием языка программирования JavaScript и фреймворка React. Мобильные приложения под iOS пишутся на языке Swift, а под Android в основном используется язык Kotlin.

Язык Swift был специально создать для устройств Apple. Поддерживает и функциональное программирование и объектно-ориентированное программирование, защищен от ошибок строгой типизаций (в отличие от его предшественника Objective-C), работа с памятью, когда часть кода (объект) перестает быть нужным, Swift автоматически освобождает память.

Язык Kotlin разработанный компанией JetBrains на базе JVM с полной компиляций в Java-байт код (с недавнего времени также компилируется в JavaScript). Достаточно прост в использовании и изучении Java-разработчикам. Используется на данный момент в разработке приложений под устройства Android вместо Java, но, конечно, не только в данной сфере разработки.

Уровень приложения (Back-end) – приложение запускается на сервере и является ключевым компонентом схемы архитектуры веб-приложения, который получает запросы пользователей, выполняет бизнес-логику и доставляет данные в СУБД. Бекенд может реализовываться на разных языках программирования, например, программная платформа Node.js, основанная на движке V8 (компилирующем JavaScript в машинный код); Java и фреймворки Spring, Python, PHP Laravel, Go, платформа .NET, Ruby.

Для того, чтобы обеспечить надежность большой системы, применяют технологию микросервисов, а также технологию Replication (запуск каждого из микро-приложений на нескольких отдельных серверах). Микросервисы обрели свою популярность, когда проблема развития большого монолитного приложения стала слишком велика, а также стало рискованными обновлять приложение, т.к. в монолитной структуре есть вероятность возникновения проблем в других местах. Внедрение новых функций тоже становится проблематичным, т.к. придется переписывать весь код, а это замедлит процесс разработки. Переход на микросервисную структуру решают эти и многие другие проблемы. Преимущества этого подхода:

– микросервисы состоят из отдельных компонентов, которые не влияют на другие, что предоставляет разработчик гибкость в управлении и развитии приложения, также есть возможность выбора технологий, более подходящих для каждого отдельного сервиса.

– каждый микросервис может масштабироваться отдельно от других, позволяет эффективно управлять нагрузками;

– каждый микросервис представляет собой отдельную единицу, и его обновление может быть проведено независимо от других служб. Это означает, что развертывание новых версий службы не требует остановки всего приложения. Это минимизирует время простоя системы и улучшает общую доступность приложения.

При внедрении данного типа архитектуры используют кросс-платформенное программное обеспечение для размещения и поддержки веб-сервера. Одно из самых популярных это веб-сервер Apache. Благодаря ему устанавливается соединение между пользователем и сервером. Он ловит данные, отправляемые пользователем, направляет их на сервер, работает с динамическими PHP страницами, отвечает за распределение нагрузки сервера, повышает отказоустойчивость. Соответственно также работает в обратную сторону, от сервера клиенту.

Помимо популярного Apache существуют и другие решения, их достаточно много, но можно выделить самых значимых конкурентов. Рассмотрим программу для балансировки нагрузки (Load Balancer), под названием NGINX. Его главное преимущество перед Apache является преодоление барьера в 10 000 одновременных соединений. В ней можно зарегистрировать все компьютеры реплики для определенного сервиса, в момент, когда приходит запрос от пользователя в конкретный микросервис, load balancer будет принимать решение на какую конкретно машину-сервер будет направлен запрос, а также может самостоятельно равномерно распределять нагрузку. Внутри есть информация под какими IP-адресами в интернете находятся те или иные сервера. Таким образом фронтенд обращается теперь только к балансировщику. Некоторые хостинги используют на серверах связку Nginx+Apache. Nginx отдает статический контент, а Apache подключается позже и отвечает за обработку динамического контента.

Еще один конкурент – Tomcat, разработанный компанией, которая выпустила Apache, это контейнер сервлетов с открытым исходным кодом, который также выполняет функцию веб-сервера. Он используется для работы приложений, написанных на Java.

Для обеспечения более корректной работы между микросервисами применяют программный компонент под названием брокер сообщений. В его работе используются две основные сущности: producer (отправитель) и consumer (потребитель/подписчик). Каждый компонент системы, который хочет отправить или получить данные, подписывается на топик, используя специальный протокол (MQTT). В зависимости от используемого протокола обмена данными, брокеры могут поддерживать различные типы сообщений, такие как текстовые, бинарные данные или даже графические изображения. Когда отправитель передает сообщение, его принимает брокер и размещает в топике. Затем программный компонент сообщает о наличии нового письма всем подписчикам данного топика. Каждый подписчик получает его копию, которую может обработать или проигнорировать в зависимости от потребностей. Этот процесс называется публикацией-подпиской и используется в различных системах, таких как интернет-чаты, социальные сети, системы мониторинга и управления ресурсами и т.д. Брокер сообщений гарантирует, что сообщение, которое ему отправили, не будет утеряно. Например, ApacheKafka.

Прикладной уровень (API) – интерфейс программирования приложения. Инструмент, который позволяет приложениям взаимодействовать друг с другом. Веб API доступен через интернет по протоколу HTTP. Создается с использованием либо Java, либо .NET.

Уровень данных – базы данных, нужны для сохранения и структурирования любых данных, чтобы позже по запросу быстро выдать ответ серверу, а он в свою очередь клиенту. Приложение на сервере в свою очередь должно уметь обращаться к базам данных. Самыми популярными на данный момент в мире являются SQL-базы данных. Это базы данных, которые используют язык запросов SQL для формирования команд к ним на вставку, получение, обновление, удаление данных и тд. БД множество разнообразных, например, PostgreSQL, MySQL, Oracle DB, Microsoft SQL Server и тд.

Кроме баз данных используется кэш, отдельное хранилище, в котором находятся данные, которые редко меняются, к которым часто обращается приложение и данные, не относящиеся к критически важным, которые часто меняются. Такой подход эффективно использует скорость работы с данными, характерную для памяти, и смягчает нагрузку центральной базы данных приложения, связанную с обработкой разного вида данных. Например, такие базы данных, как Redis или Memcached.

Redis (Remote Dictionary Service) — это опенсорсный сервер баз данных типа ключ-значение. Можно сказать, что Redis – это сервер структур данных. Уникальные особенности сервера Redis стали основной причиной его популярности и того, что он применяется во множестве реальных проектов. Первое время Redis использовали практически так же, как Memcached. Но, по мере развития Redis, эта система управления базами данных (СУБД) нашла применение и во многих других ситуациях. В частности — в реализациях механизма издатель/подписчик, в задачах потоковой обработки данных, в системах, где нужно работать с очередями. Redis поддерживает такие типы данных как: строка (String), битовый массив (Bitmap), битовое поле (Bitfield), хеш-таблица (Hash), список (List), множество (Set), упорядоченное множество (Sorted set), геопространственные данные (Geospatial), структура HyperLogLog (HyperLogLog), поток (Stream). Ещё одна важная особенность Redis заключается в том, что эта СУБД размывает границы между кешем и хранилищем данных. Тут важно понять то, что чтение данных из памяти и работа с данными, находящимися в памяти, гораздо быстрее чем те же операции, выполняемые традиционными СУБД, использующими обычные жёсткие диски (HDD) или твердотельные накопители (SSD).

В случае, если приложение подразумевает постоянный поток большого количества данных, и есть вероятность, что на машине, где запущена SQL-БД закончится память на жестком диске, используют NoSQL-базы данных. NoSQL применяется к системам, в которых делается попытка решить проблемы масштабируемости и доступности за счёт полного или частичного отказа от требований атомарности и согласованности данных. Такие базы данных позволяют легко распределяют данные по нескольким компьютерам, объединенных в кластер. Также они не предоставляют ACID-гарантии, только гарантии BASE (Basically Available, Soft State, Eventual Consistency). Например, MongoDB, Apache Cassandra, Neo4j и тд.

Для хранения файлов любого типа и объема используются object storage хранилища. Объектное хранилище, часто называемое объектно-ориентированным хранилищем, представляет собой архитектуру хранилища данных, идеально подходящую для хранения, архивирования, резервного копирования больших объемов статических неструктурированных данных и управления ими — надежно, эффективно и по доступной цене. Они предоставляют высокий уровень надежности и доступности. Принцип работы с S3-хранилищем сводится к созданию контейнеров и добавлению туда необходимых файлов, которые представляются в виде объектов. Всё,

что попадает в контейнер, можно просматривать, перемещать или удалять.

Поиск ошибок и сбоев в работе системы, выявление вредоносной активности, сбор статистики посещения веб-ресурса выполняется с помощью сбора Log-ов. Для более удобной работы с логами существует такая база данных, как Elasticsearch. Удобный пользовательский интерфейс для работы с этой базой данных предоставляет Kibana.

Elasticsearch – это распределённая RESTful-система на основе JSON, которая сочетает в себе функции NoSQL-базы данных, поисковой системы и аналитической системы. Elasticsearch может использоваться для сбора логов и аналитики журналов, так как он обладает следующими возможностями:

– быстро обрабатывать большие объёмы журналов;
– индексировать системные логи по мере поступления;
– выполнять запросы к ним в режиме реального времени.

Все компоненты, перечисленные выше, составляют архитектуру веб-приложения. Реализовать такую структуру можно на большом количестве серверов, которые нужно приобрести, разместить и настроить, а можно воспользоваться сервисом по аренде серверов, например, AWS (Amazon Web Services), Google Cloud, Heroku, Microsoft Azure и тд.

Amazon Web Services — это набор облачных сервисов от компании Amazon. На единой платформе пользователи могут заказать вычислительные ресурсы, хранилище, инфраструктуру, сервисы с готовыми для использования инструментами. Если серверы Амазон перестанут работать, это сразу станет заметно. Услугами платформы пользуются крупнейшие корпорации. Согласно результатам исследования Synergy Research Group, AWS занимает треть мирового рынка облачных решений. На второй строчке рейтинга — Microsoft Azure с долей в 19%. Но конкурентов много: Google Cloud, IBM Cloud, timeweb.cloud и другие площадки тоже предлагают клиентам удобные решения для бизнеса.

Управлять сервисами можно тремя способами: AWS Console в браузере; AWS CLI — утилита для управления через командную строку. Отличное решение для автоматизации рутины; SDK — комплект приложения разработчика. SDK доступны для разных языков программирования, от C++ и Java до Python и PHP. Даже если нужного языка нет в списке, можно найти библиотеку, которую поддерживает сообщество. Или написать свой набор инструментов. Выбор способа

зависит от квалификации администратора и задач, которые нужно решить.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Суханов В.И. Разработка веб-приложений на платформе Spring: учебно-методическое пособие / В.И. Суханов; М-во науки и высшего образования РФ. – Екатеринбург: Изд-во Урал. ун-та, 2023. – 180 с.

2. Кулаков К.А. Архитектура и фреймворки веб-приложений: учебное электронное пособие / К. А. Кулаков, В. М. Димитров; М-во науки и высшего образования Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования Петрозавод. гос. ун-т. – Электрон. дан. – Петрозаводск : Издательство ПетрГУ, 2020.

УДК 004.921

БУРЦЕВА С.Н.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ОСОБЕННОСТИ РАЗРАБОТКИ ГРАФИЧЕСКОГО РЕДАКТОРА НА ЯЗЫКЕ C++ QT

В статье рассматриваются основные принципы и возможности использования библиотеки QTC++ для разработки графического редактора.

Разработка графического редактора является интересным и захватывающим проектом, который может быть реализован с использованием различных технологий. Одним из популярных инструментов для создания графических приложений на C++ является библиотека Qt. В даннойстатье будет рассмотрено, как использовать Qt для создания графического редактора.

1. Почему Qt?

Qt – это мощная кроссплатформенная библиотека, которая предоставляет обширный набор инструментов для разработки графических приложений. Её основные преимущества включают в себя:

Кроссплатформенность: Приложения, созданные с использованием Qt, могут быть легко импортированы на различные операционные системы без изменения исходного кода.

Обширная функциональность: Qt предоставляет широкий набор классов и компонентов для создания различных типов приложений, включая графические редакторы.

Простота использования: Qt предоставляет интуитивно понятный API и удобные инструменты разработки, что делает процесс создания приложений быстрым и эффективным.

2. Основные компоненты графического редактора

Перед тем, как начать разработку графического редактора на Qt, необходимо определить основные компоненты, которые будут включены в приложение. Некоторые из основных компонентов могут включать:

Холст: Область, на которой пользователь будет создавать и редактировать изображения.

Инструменты рисования: Инструменты для создания различных элементов, таких как линии, прямоугольники, эллипсы и т. д.

Панели инструментов: Панели с кнопками для выбора инструментов рисования и настройки их параметров.

Панель свойств: Панель для отображения и редактирования свойств выбранных элементов.

Меню и панели инструментов: Меню для доступа к функциям приложения и панели инструментов для быстрого доступа к основным командам.

3. Создание графического редактора с использованием Qt

Для создания графического редактора на Qt можно использовать следующие шаги:

Создание нового проект Qt с помощью QtCreator или любой другой среды разработки.

Определение интерфейса графического редактора, включая холст, панели инструментов и свойств.

Создание классов для основных компонентов графического редактора, таких как холст, инструменты рисования и панели инструментов.

Использование классов из модуля QtWidgets для создания пользовательского интерфейса редактора.

Реализация функциональности редактора, включая рисование элементов, выбор инструментов, редактирование свойств и т. д.

4. Пример написания графического редактора

Произведём реализацию графического редактора с помощью QT, который будет позволять загружать и сохранять изображения в формате PNG, а также рисовать при помощи основных инструментов (фигур, примитивов) графики.

Создание виджета с формой образуется на основе класса QMainWindow, также требуется в область centralWidgetпоместить контейнер типа QGroupBox, находящегося в списке Conatiners, далее требуется установка его свойств выравнивания alignment, равных AlignLeft, AlignTop. В этом контейнере должны разместить кнопки инструментов (чтобы не было трудностей, обычные PushButton), установить им равные значения minSizeи MaxSize.

Почти целую часть окна должен занять контейнер QScrollArea, в которой нужно поместить обычную метку QLabel, назначенную для него виджетом, которая будет непосредственно отображать изменения и редактирования в рисунке стандартным методом setPixmap.

При создании кнопок, у них будет имя по умолчанию“pushButton”, в последствии требуется переименовывать каждый созданный компонент понятными именами, чтобы не было затруднений в читаемости кода.

Кнопки инструментов подключены к слотам clicked() в форме редактора сигналов и слотов. Сам процесс рисования осуществляется через обработку события mouseMoveEvent (с отображением координат текущей точки в строке состояния, если активен процесс рисования) и трех других событий от мыши.

Типичный цикл работы приложения выглядит следующим образом:

При нажатии кнопки выбора инструмента вызывается слот, подключенный к этой кнопке (например, pen, setColor), который сохраняет выбранное состояние в переменной instrument и инициирует изменение панели инструментов с помощью метода repaintButtons.

При возникновении события нажатия кнопки мыши (mousePressEvent) мы вычисляем позицию текущей точки current по координатам клика.

При обработке события движения курсора (mouseMoveEvent) с учетом зажатой левой кнопки мыши, вычисляются координаты следующей точки next и выводятся в строку состояния окна. В конце обработчика инициируется перерисовка рабочей области с помощью метода repaint.

При отпускании кнопки мыши (mouseReleaseEvent) снова вычисляется следующая точка next, и если выбран инструмент рисования, вызывается метод draw.

Событие mouseDoubleClickEvent используется только при рисовании полигона (инструмент 6). Оно устанавливает флаг isDblClick (был ли двойной щелчок), и если выбран инструмент

"полигон", соединяет текущую точку current с currentTmp, запомненной при первом вызове метода drawPolygon.

Сам процесс рисования управляется из обработчика события paintEvent, который выполняет отрисовку на изображении (img) для карандаша и ластика или на его копии (imgTmp) для линии, эллипса, прямоугольника и полигона. Рисование происходит с учетом координат двух точек current и next.

Методы рисования с именами draw проверяют, отличаются ли точки current и next, и если да, выполняют окончательную отрисовку с учетом выбранного цвета инструмента.

Методы обработки пунктов меню "Файл" и события закрытия формы closeEvent выполняют стандартные действия по созданию, открытию и сохранению файла. При учебных целях предотвращается закрытие приложения до сохранения текущего файла.

Инструмент "вывод текста" не был включен в проект, но может быть реализован, например, в виде класса-наследника от QGraphicsitem.

Разработка примитивного графического редактора (рисунок 1):

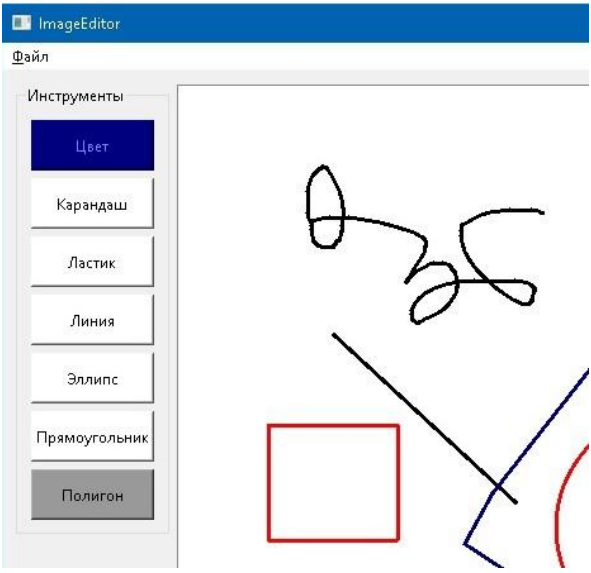


Рисунок 1 – Примитивный графический редактор

Разработка графических редакторов самостоятельно на C++ с использованием Qt является очень полезным занятием по следующим причинам:

1.Глубокое понимание принципов работы графических приложений: Создание графического редактора с нуля позволяет глубоко понять принципы работы графических приложений, включая взаимодействие с пользователем, обработку событий мыши и клавиатуры, а также рисование на холсте.

2. Развитие навыков программирования: Работа над графическим редактором позволяет развивать навыки программирования на C++ и использования библиотеки Qt. Это включает в себя работу с классами, объектами, наследованием, полиморфизмом и другими концепциями объектно-ориентированного программирования.

3. Понимание архитектуры приложений: Создание графического редактора требует понимания архитектуры приложений, включая разделение кода на модули, обработку событий и обновление пользовательского интерфейса.

4.Контроль над функциональностью и дизайном: Самостоятельное создание графического редактора дает полный контроль над его функциональностью и дизайном. Вы можете реализовать любые функции и интерфейсные решения, которые соответствуют вашим потребностям и предпочтениям.

5.Улучшение карьерных перспектив: Навыки разработки графических приложений на C++ с использованием Qt являются востребованными на рынке труда. Они могут улучшить ваше резюме и открыть двери для новых карьерных возможностей в области разработки программного обеспечения.

В целом, самостоятельная разработка графических редакторов на C++ с использованием Qt представляет собой отличную возможность для углубленного изучения программирования, повышения квалификации и создания качественных приложений.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Бланшетт, Джасмин, и Марк Саммерфилд. Программирование графического интерфейса пользователя на C++ с Qt 5. – PearsonEducation, 2016. – 832 с.

2. Миллер, Майкл, и Николь Хауптманн. "Программирование на C++ с использованием библиотеки Qt." – ДМК Пресс, 2015. – 352 с.

3. Макаров, Сергей. "Qt. Программирование для профессионалов." – БХВ-Петербург, 2017. – 608 с.

УДК 004.514.6

БУРЦЕВА С.Н.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

РАЗРАБОТКА РЕДАКТОРА ДРАКОН-СХЕМ

В статье рассматривается идея построения ДРАКОН-схем и принципы разработки графического редактора на C++ QT.

Язык Дракон

Язык ДРАКОН (DRAKON) был создан при участии Федерального космического агентства России (НПЦ АП Н.А. Пилюгина, Москва) и Российской академии наук (ИПМ им. М.В. Келдыша, Москва). На основе ДРАКОНа была разработана автоматизированная технология проектирования алгоритмов и программ под названием "ГРАФИТФЛОКС", которая успешно применяется во многих крупных проектах, включая "Ангара", "Фрегат", "Морской старт" и другие.

Визуальный язык ДРАКОН обладает уникальными характеристиками в эргономике. В отличие от обычных блок-схем, ДРАКОН-схемы сохраняют ясность даже при сложных алгоритмах, а новый способ изображения логических операций при помощи инфографики делает их более прозрачными и удобными для понимания.

ДРАКОН-схемы представляют собой графический язык программирования, предназначенный для проектирования и визуализации алгоритмов.

Существует несколько причин, почему ДРАКОН-схемы могут быть полезны:

1) *Наглядность: ДРАКОН*-схемы представляют алгоритмы в виде графических элементов, что обеспечивает высокую степень наглядности. Это упрощает понимание структуры программы и последовательности её выполнения.

2) *Простота использования:* Создание ДРАКОН-схем не требует специальных навыков программирования. Этот графический язык достаточно интуитивно понятен и может быть использован как программистами, так и людьми без опыта в программировании.

3) *Анализ и отладка:* ДРАКОН-схемы могут использоваться для анализа алгоритмов и выявления потенциальных проблем до начала

программирования. Они помогают выявить логические ошибки и неясные моменты в алгоритме.

4) *Обучение программированию:* ДРАКОН-схемы могут быть полезны для обучения основам программирования. Они помогают начинающим программистам лучше понять основные концепции, такие как последовательность, условия и циклы.

5) *Документация:* ДРАКОН-схемы могут служить в качестве документации к программному коду. Они помогают другим разработчикам лучше понять структуру программы и её логику.

6) *Планирование проекта:* ДРАКОН-схемы могут использоваться для планирования и проектирования программных проектов. Они помогают разработчикам лучше представить структуру программы и определить последовательность выполнения задач.

Дракон-схемы и их составляющие

Дракон-схемы представляют собой совокупность действий. Под действием понимается нечто, что имеет конечную продолжительность и приводит к желаемому и совершенно определённому результату.

Использование ДРАКОН-схем может значительно улучшить понимание и эффективность разработки программных проектов, особенно на ранних стадиях проектирования.

Пример ДРАКОН-схемы из научных публикаций (рисунок 1):

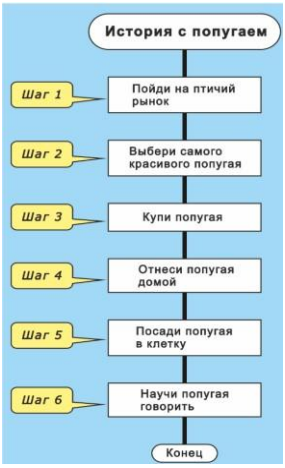


Рисунок 1 – Пример ДРАКОН-схемы “История с попугаем”

Пример ДРАКОН-схемы в разработанном графическом редакторе (рисунок 2):

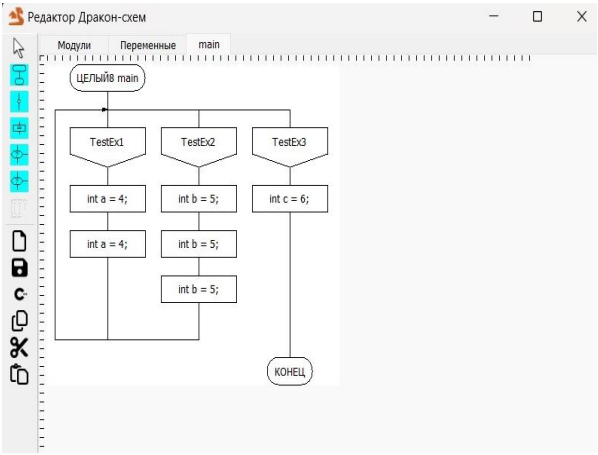


Рисунок 2 – Пример Дракон-схемы

QT и причины использования данной библиотеки для разработки графического редактора

В Qt C++ удобно создавать графический интерфейс для рисования ДРАКОН-схем по нескольким причинам:

- *Графические элементы:* Qt предоставляет широкий набор графических элементов, таких как QGraphicsView, QGraphicsScene, QGraphicsItem и другие, которые могут быть использованы для создания интерактивного рисования ДРАКОН-схем.

- *Гибкость и настраиваемость:* Qt позволяет создавать пользовательские элементы управления и настраивать их внешний вид и поведение в соответствии с вашими потребностями. Это позволяет создавать удобные инструменты для рисования ДРАКОН-схем с учетом специфических требований пользователя.

- *Событийная модель:* Qt предоставляет мощную событийную модель, которая позволяет легко обрабатывать события мыши, клавиатуры и другие пользовательские действия. Это позволяет создавать интерактивные редакторы ДРАКОН-схем с поддержкой функций, таких как выделение, перемещение и масштабирование элементов.

- *Многоплатформенность:* Приложения, разработанные с использованием Qt, могут быть легко импортированы на различные операционные системы, включая Windows, macOS и Linux. Это обеспечивает удобство использования и доступность редактора ДРАКОН-схем для широкого круга пользователей.

Отрисовка компонентов ДРАКОН-схемы

Можно рассмотреть отрисовку компонентов на примере метода drawRoundedRect().

Метод drawRoundedRect() в Qt C++ используется для отрисовки прямоугольника с закругленными углами. Его сигнатура:

void QPainter::drawRoundedRect(constQRectF&rect, qrealxRadius, qrealyRadius, Qt::SizeMode mode = Qt::AbsoluteSize)

Этот метод принимает следующие параметры:

- rect: прямоугольник (QRectF), который нужно нарисовать.
- xRadius: радиус закругления по горизонтали.
- yRadius: радиус закругления по вертикали.
- mode (необязательный): режим размеров закругления. По умолчанию используется абсолютный размер (Qt::AbsoluteSize), но также можно использовать относительные размеры с помощью Qt::RelativeSize.

Пример кода, используемый в коде при написании графического редактора для отрисовки компонента “заголовок” ДРАКОН-схемы:

```
//Заголовок
Qp.drawRoundedRect(headX, headY, TITLE_LEN, 2*ROUND_D, ROUND_D, ROUND_D, Qt::AbsoluteSize);
QFontMetricsfm(qp.font());
QString txt;
txt = type + “ “ + name;
QRectF a = fm.boundingRect(txt);
While(a.width() > TITLE_LEN){
  txt = txt.left(txt.lenght() – 1);
  a = fm.boundingRect(txt + “…”);
}
qp.drawText(QRect(headX, headY, TITLE_LEN, 2*ROUND_D), QT::AlignCenter, txt);
```

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Пароджанов В.Д. Алгоритмы и жизнеритмы на языке ДРАКОН. Разработка алгоритмов. Безошибочные алгоритмы. – М., 2019. – 374 с.

2. Ковальчук М., Михайленко Л. Алгоритмическая культура как компонент алгоритмической деятельности // Knowledge, Education, Law, Management 2018 № 1 (21)

3. Филатова Л.Ю., Филатова А.С. Развитие алгоритмического стиля мышления при обучении студентов вуза // Наука ЮУрГУ: материалы 67-й научной конференции. Секции естественных наук. – С. 469-472.

4. Грис Д. Наука программирования. – М.: Мир, 1984. – С. 303.

УДК 004.054

ВАСИЛЬЕВА Т.С., САПРЫКИН А.Н.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

РАЗРАБОТКА ЭФФЕКТИВНЫХ СТРАТЕГИЙ ТЕСТИРОВАНИЯ ВЕБ-СТРАНИЦ С ПОМОЩЬЮ АВТОМАТИЗИРОВАННЫХ ИНСТРУМЕНТОВ

Рассматриваются уровни автоматизации, выбор и использование специализированных инструментов, а также демонстрация их работы для тестирования веб-приложений.

В современном мире веб-приложения играют ключевую роль в повседневной жизни людей и успешной деятельности компаний, вопрос обеспечения их надежности и качества становится все более актуальным. Высокое качество веб-приложения обеспечивается эффективной стратегией автоматизации тестирования.

Стратегия тестирования веб-приложений – это спланированный и организованный подход к проверке и гарантированию качества приложений. Она включает в себя определение целей тестирования, выбор методов и инструментов, разработку тестовых сценариев, выполнение тестов и анализ результатов с целью выявления дефектов и обеспечения соответствия разрабатываемого продукта требованиям заказчика [4].

В автоматизированном тестировании веб-приложений все этапы подготовки тестовых данных, запуска и выполнения шагов для проверки качества осуществляются с использованием инструментов автоматизации тестирования. Эти инструменты эмулируют поведение пользователя при взаимодействии с пользовательским интерфейсом приложения [3].

Интерес к данному вопросу возникает в свете растущей потребности в автоматизации тестирования пользовательского интерфейса, что делает его особенно значимым и актуальным. Это позволит увеличить покрытие тестирования [1], сэкономить время на тестирование старого функционала, сократить время обнаружения ошибок и ускорить процесс их исправления.

Уровни автоматизации

В контексте автоматизации процессов тестирования, можно выделить несколько ключевых уровней:

- Модульное тестирование (Unit Tests Layer) – фокус на тестировании отдельных компонентов веб-приложения, выполняемых

разработчиками, что обеспечит правильную работу каждого элемента приложения в отдельности [3].

- Функциональное тестирование (Functional Test Layer) – тестирование веб-приложения посредством доступа к функциональному слою, обходя пользовательский интерфейс, что позволит проверить работу функций приложения на корректность [3].

- Тестирование пользовательского интерфейса (GUI Test Layer) – доступна проверка не только GUI, но и выполнения функционального тестирования, взаимодействуя с бизнес-логикой веб-приложения [3].

Позиция тестирования пользовательского интерфейса в ходе автоматизированных проверок

Выбранная стратегия автоматизации основывается на уровне тестирования пользовательского интерфейса, что обусловлено нарастающей популярностью подхода автоматизации тестирования программного обеспечения через графический пользовательский интерфейс (GUI) [4]. Этот подход позволяет проводить тестирование в том же режиме, в котором оно будет использоваться конечными пользователями, что обеспечивает высокую степень имитации реального использования приложения.

Для полноты пояснения выбора данной стратегии будут рассмотрены некоторые преимущества и недостатки. Автоматизированного тестирование на реальных устройствах обладает рядом достоинств, включая проведение тестирования на девайсах с разными конфигурациями, операционными системами (ОС) и браузерами. Отметим важность этого подхода для тестирования линейки поддерживаемых мобильных устройств. Однако, несмотря на его значимость, имеется ряд недостатков: длительность тестирования, требования к поддержке и нестабильность тестов.

Инструменты для автоматизации тестирования пользовательского интерфейса

Автоматизация тестирования пользовательского интерфейса осуществляется с помощью специализированных инструментов, таких как Selenium, которые позволяют взаимодействовать с интерфейсом тестируемого приложения.

При выборе инструментов для автоматизации тестирования веб-страниц, следует учитывать критерии: поддержка разных платформ и приложений, обеспечение запуска тестов, интегрированная среда разработки, настройка тестовых сценариев, совместное выполнение тестов и его настройка, создание репортов.

В процессе автоматизации приложение добавляет дополнительную библиотеку, например, Selenium WebDriver для

Python, в код приложения. Selenium WebDriver является популярным средством для автоматизированного тестирования пользовательского интерфейса, обеспечивая высокую производительность и надежность. Он позволяет тестировать веб-приложения на различных браузерах и ОС, что делает его идеальным выбором для автоматизации тестирования пользовательского интерфейса. Это позволяет разработчикам тестов получить доступ к элементам интерфейса, изменить информацию о них и имитировать действия пользователя, такие как клики мышью, нажатия клавиш на клавиатуре, что обеспечит высокое качество приложения.

Кроме того, Python, в сочетании с Selenium, обеспечивает мощную платформу для автоматизации тестирования, благодаря богатому набору библиотек и инструментов, таких как BeautifulSoup для парсинга HTML и XML документов, что может быть полезно для анализа и взаимодействия с веб-страницами в рамках тестирования.

Демонстрация работы специализированных средств

На рисунках 1-2 представлена реализация рассмотренных примеров.

Пример 1. Код открывает веб-страницу с помощью Selenium, получает HTML-код страницы, парсит его с помощью BeautifulSoup [2], находит все ссылки на странице и проверяет их активность с помощью библиотеки requests.

```
from selenium import webdriver
from bs4 import BeautifulSoup
import requests
# Инициализация драйвера Selenium
driver = webdriver.Chrome()
# Открываем веб-страницу
driver.get("https://www.example.com")
# Получаем содержимое страницы
page_source = driver.page_source
# Используем BeautifulSoup для парсинга HTML
soup = BeautifulSoup(page_source, "html.parser")
# Находим все ссылки на странице
links = soup.find_all("a")
for link in links:
    href = link.get("href")
    # Проверяем активность ссылки
    if href.startswith('http'):
        response = requests.head(href)
        if response.status_code == 200:
            print(f"Ссылка {href} активна")
        else:
            print(f"Ссылка {href} неактивна (код {response.status_code})")
# Закрываем драйвер Selenium
driver.quit()
```

Рисунок 1 – Реализация примера 1

Пример 2. Код открывает веб-страницу с помощью веб-драйвера Chrome, вводит текст "Текст для ввода" в определенное поле, получает HTML-код страницы, используя BeautifulSoup, и проверяет содержимое поля ввода на предмет наличия только букв русского и английского алфавита с использованием регулярных выражений [2].

```
import re
from selenium import webdriver
from bs4 import BeautifulSoup
# Запуск веб-драйвера
driver = webdriver.Chrome()
driver.get("https://www.example.com")
# Находим элемент поля ввода
input_element = driver.find_element_by_id("input-field-id")
# Вводим текст в поле
input_element.send_keys("Текст для ввода")
# Получаем содержимое страницы
html = driver.page_source
# Используем BeautifulSoup для парсинга HTML
soup = BeautifulSoup(html, 'html.parser')
# Пример проверки наличия только букв в тексте поля
input_text = input_element.get_attribute("value")
if re.match("^[а-яА-Яа-zA-Z]+$", input_text):
    print("В поле были введены только буквы.")
else:
    print("В поле были введены не только буквы.")
# Закрываем веб-драйвер
driver.quit()
```

Рисунок 2 – Реализация примера 2

Отметим, что выбор подходящих инструментов и разработка эффективной стратегии тестирования пользовательского интерфейса позволяют увеличить эффективность процесса тестирования, сократить время обнаружения ошибок и ускорить процесс исправления дефектов.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Терехов, А.Н. Платонова, М.В. Моделирование бизнес-процессов в цифровую эпоху [Электронный ресурс] // Российский журнал менеджмента 2019. URL: https://cyberleninka.ru/article/n/modelirovanie-biznes-protsessov-v-tsifrovuyu-epohu/viewer

2. Веб-парсинг Python - извлечение данных [Электронный ресурс]. URL: https://isolution.pro/ru/t/python-web-scraping/python-web-scraping-data-extraction/veb-parsing-python-izvlecenie-dannyh

3. Куликов С.С. Тестирование программного обеспечения. Базовый курс. – Минск: EPAM Systems, 2015-2022. – 298 с.

4. Разработка стратегии тестирования [Электронный ресурс]. URL: https://testirovshik.com/test-strategy/

УДК 004.054

ВАСИЛЬЕВА Т.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПОПУЛЯРНЫХ ИНСТРУМЕНТОВ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Проводится сравнение возможностей инструментов автоматизации тестирования программного обеспечения и анализ их эффективности.

Автоматизированное тестирование программного обеспечения (ПО) представляет собой методологию, использующую специализированные инструменты для выполнения тестовых сценариев без прямого участия человека [3]. Этот подход обеспечивает эффективное тестирование на больших объемах данных и при необходимости повторения тестов, что особенно актуально в условиях постоянного развития и обновления ПО [1]. Основным преимуществом автоматизированного тестирования является возможность значительного сокращения времени и ресурсов, что позволяет повысить эффективность и снизить затраты на качественное обеспечение ПО.

Автоматизация тестирования веб-приложений через графический пользовательский интерфейс (GUI) является распространенной практикой, поскольку она имитирует действия пользователя, что обеспечивает более высокую степень совместимости и точности тестирования. Этот подход позволяет проводить тестирование в любое время суток, используя заранее подготовленные скрипты, что увеличивает скорость и эффективность процесса.

Выбор инструментов автоматизации тестирования

Важным аспектом автоматизированного тестирования является выбор подходящих инструментов, значительно влияющих на успешность процесса, например, Ixia, SoapUI, JMeter, HP LoadRunner.

Решения для тестирования от компании Ixia представляют собой инструменты, способные эмулировать широкий спектр сетевых условий, включая различные типы трафика (от мультимедийного до атак различной степени сложности). Подобные решения позволяют

организациям проводить тестирование и валидацию сетевых устройств и систем с использованием реального трафика, и атак [3], что является ключевым для обеспечения безопасности и надежности сетевой инфраструктуры.

В свою очередь, инструмент SoapUI позволяет точно моделировать поведение различных сервисов и баз данных, обеспечивая возможность проверки их работоспособности под разными нагрузками и сценариями, близкими к реальным условиям. Таким образом, средство применимо для выполнения функционального [3] и нагрузочного тестирования путем создания необходимых сервисов и отправки HTTP и JDBC запросов.

Для выполнения нагрузочного тестирования эффективно использовать инструмент JMeter. Он позволяет тестировать не только протоколы передачи данных и JDBC-соединения, но и контролировать процесс создания немалого числа запросов через несколько компьютеров с одного центрального узла управления. Таким образом, средство предоставляет возможность эффективного моделирования нагрузки на приложение и оценки его производительности в разных условиях.

С использованием HP LoadRunner можно проводить тестирование приложений и веб-сайтов различной сложности, моделировать одновременную активность большого числа пользователей, выполняющих разнообразные операции, а также контролировать использование системных ресурсов. Этот инструмент обеспечивает возможность реалистичного воспроизведения нагрузки, анализа производительности и выявления узких мест в приложениях с целью улучшения их эффективности и надежности.

Стоит отметить, что определенные веб-приложения имеют уникальные характеристики (архитектурные, функциональные), а значит указанные инструменты не всегда эффективны для автоматизированного тестирования, в силу неспособности учета их специфики.

Подобные ситуации вынуждают разработчиков ПО писать собственные наборы автоматизированных тестов, учитывая особенности конкретного приложения [1].

Анализ специализированных средств для создания автоматизированных тестов

Для решения данной проблемы существует ряд специализированных средств на рынке информационных технологий (ИТ) для создания автоматизированных тестов: Selenium, Katalon Studio, UFT, TestComplete и WATIR.

Selenium – это ведущий фреймворк с открытым исходным кодом, который широко используется инженерами по тестированию, обладающими высоким уровнем навыков программирования и опытом в написании скриптов [2]. Он совместим с различными ОС и поддерживает распространенные браузеры. Инструмент имеет обширный набор API и библиотек, которые могут быть настроены в соответствии с требованиями конкретного проекта. Фреймворк также включает в себя несколько инструментов с индивидуальной функциональностью для автоматизации, таких как Selenium IDE (для тестирования без необходимости изучения конкретного языка скриптов), Selenium WebDriver (платформа для автоматизации современных веб-приложений), Selenium RC и Selenium Server. С их помощью тестировщики могут создавать сложные тестовые сценарии для удовлетворения различных уровней сложности.

Katalon Studio предназначен для автоматизированного тестирования приложений и сервисов. Инструмент включает в себя несколько инструментов с индивидуальной функциональностью для автоматизации, таких как Katalon Studio IDE (среда разработки тестов), Katalon Analytics (для анализа результатов тестирования), Katalon TestOps (платформа для мониторинга тестов). Средство функционирует на распространенных ОС и совместимо с основными браузерами. Инструмент можно интегрировать в CI/CD-процессы и предоставить пользователям полный обзор хода тестирования [2].

UFT служит для автоматизации функционального [3] тестирования на основе записи и воспроизведения действий пользователя в приложениях с GUI. Необходимо отметить функционирование под единственной ОС (Windows). UFT использует Visual Basic Script (VBScript) и поддерживает непрерывную интеграцию (CI) [2].

TestComplete предназначен для автоматизации тестирования разнообразных приложений. Он умеет выполнять код на некоторых языках программирования, работает только под ОС Windows и поддерживает большинство современных браузеров. Средство может распознавать объекты GUI [2].

WATIR служит для автоматизированного тестирования веб-приложений. Инструмент совместим только с библиотекой Ruby [2]. Он поддерживает кроссплатформенное тестирование и кросс-браузерное тестирование во многих существующих браузерах.

Изучив возможности инструментов, была составлена таблица сравнения для выбора более предпочтительного.

Таблица 1 – Сравнительная таблица для инструментов

Инструмент	Selenium	Katalon Studio	UFT	TestComplete	WATIR
Кроссплатформенность	+	+	-	-	+
Языки программирования	Java, C#, Perl, Python, JS, Ruby	Java и Groovy	VBScript	JS, VBScript, Python, C#, C++	Ruby, C#
Уровень программирования пользователя	-	-	-	-	-
Кросс-браузерность	+	+	+	+	+
Сложность обучения	Высокая	Средняя	Средняя	Средняя	Средняя
Вид тестируемого приложения	Web-приложения	Web, API, десктопные	Web, мобильные, API, десктопные	Web, мобильные, десктопные, API	Web-приложения
Скорость создания тестового сценария	Низкая	Высокая	Высокая	Высокая	Средняя
Тип лицензии	Открытый исходный код	Бесплатное ПО	Патент	Патент	Открытый исходный код

По результатам сравнения можно сделать вывод о том, что Selenium более оптимальный инструмент по сравнению с другими, благодаря своей языковой гибкости, поддержке большого количества ОС и открытому исходному коду, предоставляемому бесплатно.

Отметим, что тестирование может быть осуществлено разными методами в соответствии с доступом к структуре реализованного приложения и автоматизационных требований. При использовании автоматизации в процессе тестирования, инструменты подбираются в соответствии с особенностями приложения.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Терехов А.Н., Платонова М.В. Моделирование бизнес-процессов в цифровую эпоху [Электронный ресурс] // Российский журнал

менеджмента 2019. URL: https://cyberleninka.ru/article/n/modelirovanie-biznes-protsessov-v-tsifrovuyu-epohu/viewer

2. Топ 10 инструментов автоматизации тестирования 2023 [Электронный ресурс]. URL: https://habr.com/post/342234/

3. Куликов С.С. Тестирование программного обеспечения. Базовый курс. – Минск: EPAM Systems, 2015-2022. – 298 с.

УДК 004.415.2.052.03

ВЕНЧИКОВ Д.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ОСНОВНЫЕ ПРИНЦИПЫ ПРИ ПРОЕКТИРОВАНИИ АРХИТЕКТУРЫ ПРИЛОЖЕНИЙ

В данной статье рассматриваются основные принципы проектирования архитектуры, которые позволят создать гибкое решение с потенциалом для дальнейшего развития и позволит осуществлять поддержку с минимальными затратами

Рассмотрение обозначенных принципов предлагается рассмотреть начиная с основы – изучения непосредственно тех задач, которые должен решать программный комплекс. Речь пойдет о разработке коммерческого ПО, следовательно, одним из основных критериев является стоимость реализации. Из этого исходит то, что создавать идеальное программное обеспечения, где будут учтены все нюансы, которые могут возникнуть в будущем не только невозможно, но и не имеет коммерческого смысла, ведь цель программного обеспечения для коммерческой компании – увеличении прибыли в ближайшем или не очень будущем, а не создание ПО ради создания ПО. Однако это ни в коем случае не значит, что надо забыть про все best practices и создавать программный комплекс выбрав в качестве критерия главного критерия оптимизации лишь стоимость и время разработки. Все выше сказанное подводит нас к тому, что нужно всегда искать баланс между использованием best practices, когда это требуется и соблюдением принципа KISS (keep it simple stupid). Поиск этого баланса должен происходить вместе с представителями бизнеса, чтобы заранее знать, какие подходы имеет смысл внедрять, а какие абсолютно не имеет смысла, не смотря на кажущуюся необходимость и красоту решения с технической точки зрения. Хорошо иллюстрирует данную мысль график, представленный на Рисунке 1.

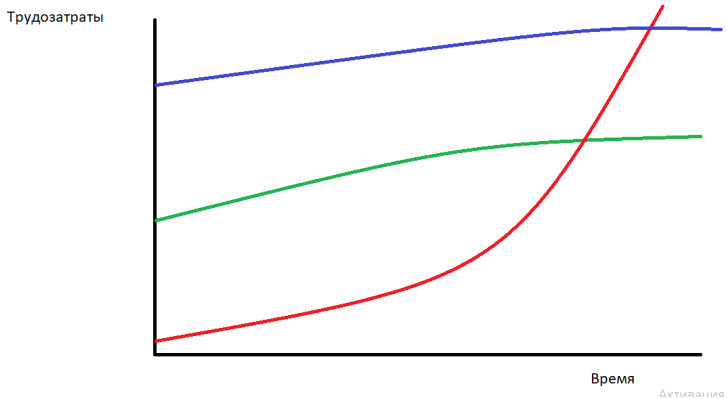


Рисунок 1 – График трудозатрат от времени

Красной линией представлено изменения трудозатрат с течением времени, при несоблюдении архитектурных принципов построение информационной системы. Хорошо видно, что на старте работ, такой подход имеет меньше всего затрат, однако при дальней шей поддержке накапливается так называемый «технический долг», который будет забирать ресурсы команды поддержки на дистанции, что может привести даже к закрытию бизнеса из-за невозможности тратить так много средств на поддержку, даже без внедрения нового функционала. Синей линией представлена ситуация, когда команда разработки не синхронизируется с представителями бизнеса и внедряет не нужные бизнесу подходы. Очевидно, что истина где-то по середине, именно в совокупности с бизнесом необходимо решать, насколько сильно поднимать трудозатраты. Бизнес задает требования, и лишь согласно этим требованиям необходимо применять практики создания архитектуры. В данной статье рассмотрим лишь 2 требования:

- отказоустойчивость (надёжность)
- масштабируемость по количеству человек в проекте/ по количеству запросов в секунду

Итак, перейдем к обзору практик, которые решают данные требования.

Отказоустойчивость. Это основа доступности сервисов, очень важный для бизнеса аспект. Для реализации отказоустойчивости сервисов применяют кластеризацию для разделения БД по разным физическим серверам. В зависимости от того, насколько критична данная система происходит разбиение на нужное количество серверов. Аналогично следует поступить и со BL приложением, реализовав

автоматическую репликацию контекста клиента на резервный блок в случае отказа основного, а также механизм обратной репликации, если восстановление последнего прошло успешно. Однако важно найти баланс между доступностью и затрачиваемыми средствами, которые будут уходить на обслуживание/аренду дополнительных серверов и devops.

Задача бизнеса – четко установить требования сервиса, задача команды разработки сделать финансовую оценку. Если бизнес согласен принять ее команда реализовывает ее согласно требованиям. Существует 4 уровня критичности инфраструктуры:

1. Mission critical;
2. Business critical;
3. Business operational;
4. Office productivity

Выбор требований основан на следующих пунктах:

1. Приоритет обслуживания при массовых инцидентах**.** Безусловно, любую систему нужно восстанавливать после аварии, но если авария задела несколько систем, то сначала нужно восстановить наиболее критичные.

2. Типовые значения SLA (service level agreement).

3. Стандартные инфраструктурные решения. Каждое из перечисленных выше решений обладает определёнными характеристиками надёжности, обеспечивающими скорость восстановления при сбоях, а также определённой стоимостью.

4. Тяжесть последствий остановки системы.

Для каждого класса последствий следует сформулировать критерии тяжести и присвоить им оценки от 0 до 4. Например, таблица может выглядеть так, как представлена на Рисунке 2:

	0	1	2	3	4
Экономические	нет	<0.1% плановой прибыли	0.1%..0.5% плановой прибыли	0.5%..1% плановой прибыли	>1% плановой прибыли
Клиентские	нет	1 клиент	>1% клиентов	>5% клиентов	>10% клиентов
Репутационные	нет	огласка маловероятна	огласка в локальных СМИ	огласка в региональных СМИ	огласка в федеральных СМИ
Юридические	нет	предупреждения регуляторов	штрафы регуляторов	гражданские иски	риск отзыва лицензии

Рисунок 2 – Таблица с описанием критериев уровней критичности

Разумеется, такая оценка условна, она должна быть индивидуальная для каждой компании, однако общая картина должна быть похожа на ту, что представлена на рисунке 3 ниже:

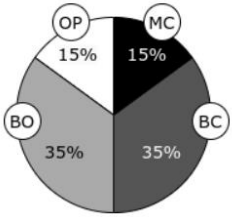


Рисунок 3 – Диаграмма распределения функциональных подсистем по частотам встречаемости различных уровней критичности

Основные правила, которые следует соблюдать при выборе класса критичности:

1. Если система обеспечивает работоспособность другой системы, то её класс критичности не может быть ниже, чем класс зависимой системы

2. Убытки, понесённые в результате простоя системы, не могут быть ниже, чем убытки, нанесённые прерыванием бизнес-процесса, который эта система обеспечивает. В свете этого правила очень интересно бывает оценить корпоративную систему электронной почты, ибо внезапно оказывается, что на неё завязан обмен критичной информацией.

3. Если в компании одну систему используют несколько блоков, и оценки этих блоков для системы отличаются, то следует использовать максимальную оценку.

Масштабируемость. Для реализации этого требования применяются такие подходы как:

1. Применение SOA (Service-oriented architecture)
2. Использование API Composer
3. Использование API Composer (Gateway)
4. Использование JHipster
5. Вертикальное и горизонтальное шардирование данных

Важным нюансом также является то, что при создании проекта рекомендуется сначала разработать и отладить монолитное приложение и лишь потом начинать разделять его на микросервисы. Такой подход поможет заранее выявить некоторые проблемы, а также лучшим образом организовать разделение в дальнейшем ввиду того, что в процессе отладки и разработки неизбежно будет изменяться исходный функционал и логическое разделение лучше организовать

уже после того, как будет понятно, какими будут итоговые функции монолита.

Так как микросервисный подход более затратен с точки зрения реализации, следует ответственно отнестись к выбору такого подхода, так как он тянет за собой следующие аспекты:

1. Сложное развертывание приложения
2. Получаем новый слой вроде сети
3. Более медленная работа из-за дополнительного слоя в виде сети
4. Сложная горизонтальная масштабируемость
5. Нужна работа с Kubernetes / OpenShift
6. Сложность в анализе логов
7. Сложность проверки сервисов на uptime
8. Тестировать микросервисную архитектуру сложнее
9. Сложная конфигурация хранилища для наших сервисов

Любые нововведения в архитектуру проекта должны согласовываться представителями бизнеса, являющимися достаточно компетентными, чтобы выставить четко обозначенные требования для команды разработки. Именно тесное взаимодействие команды разработки с представителями бизнеса определяет успешность проекта.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Классификация критичности информационных систем [Электронный ресурс]: URL: https://habr.com/ru/articles/512556/
2. Разработка архитектуры [Электронный ресурс]: URL: https://habr.com/ru/articles/658151/
3. Технология кластеризации базы данных [Электронный ресурс]: URL: https://cyberleninka.ru/article/n/tehnologiya-klasterizatsii-bazy-dannyh

УДК 00.005

ВЕНЧИКОВ Д.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ОСНОВНЫЕ СПОСОБЫ ОПТИМИЗАЦИИ ДОРОЖНОГО ТРАФИКА И ИХ ОСОБЕННОСТИ

В данной статье рассматриваются основные способы оптимизации трафика и улучшения пропускной способности городской дорожной сети. Проблема пробок в

городах весьма важна, ведь от скорости движения транспорта напрямую зависит экономика города, а значит и страны в целом. В этой статье речь пойдет об основных методах оптимизации дорожного трафика. Целью статьи является приоритизация этих методов по степени влияния на увеличение скорости движения потока транспорта, а также оценка их плюсов и минусов.

Рассмотрим влияние качество дорожного полотна. Очевидно, что качество автодороги сильно влияет на скорость передвижения по ней, однако насколько сильно? Предлагается представить небольшой мысленный эксперимент. Например, имеем некий участок дороги, протяженностью 1 км. Будем считать, что транспорт имеет возможность въехать на него с средней скоростью беспрепятственного движения в городе, примем эту скорость за 60 км/ч. Будем считать, что за участком в один километр есть поворот на 90 градусов – классическая ситуация если брать поездку в городе, следовательно скорость в конце данного участка должна быть не более 30 км/ч, для удачного прохождения поворота. Для моделирования стиля вождения водителя возьмем усредненные показатели – ускорение 10 секунд от нуля до 100 км/ч, что соответствует ускорению 2.77 м/$с^2$ по СИ. Учитывая плавность движения, примем ускорение торможения за 4 м/$с^2$. Попробуем рассчитать среднюю скорость движения на данном участке дороги. Уравнение движения рассчитывается по формуле 1:

$$S = U_H * t_1 + U_H * t_2 - \frac{a_T t_2^2}{2} \quad (1)$$

где $U_н$ – скорость движения авто в начале участка, t_1- время равномерного движения без торможения перед поворотом, $a_т$ – ускорение при торможении, t_2^2 – квадрат времени, в течение которого авто будет тормозить. Итого, время прохождения данного участка – 59,689 секунд.

Далее представим, что на ередине участка дороги есть неровность в виде ямы, которую нельзя проехать на скорости более 5 км/ч. Рассчитаем время прохождения участка снова, но для ситуации с остановкой (формула 2):

$$S = U_H * t_1 + U_H * t_2 - \frac{a_T t_2^2}{2} + 5 * t_3 + \frac{a_y t_3^2}{2} + U_H * (t_4 + t_5) + \frac{a_T t_5^2}{2} \quad (2)$$

Общее время во втором случае – 62,564. Соответственно разница составляет 3 секунды на километр, что составляет около 5 процентов задержки. При этом, это не самый плохой вариант и при более плохом состоянии дорожного полотна время поездки может увеличится

кратно. При самом плохом варианте, средняя скорость может упасть до 10 км/ч, что приведет к 6-и кратному увеличению времени поездки.

Минусом увеличения пропускной способности данным способом является дороговизна проведения ремонтных работ, а также необходимость полного перекрытия линии для ремонта, что может в иных случаях привести к транспортному коллапсу.

Далее рассмотрим, пожалуй, один из самых противоречивых способов увеличения пропускной способности автодороги – ее расширение и добавление новых полос. Идея очевидна – расширив дороги, можно добиться увеличения количества транспорта, проезжающего через определенный промежуток за единицу времени, однако здесь есть много подводных камней, таких как:

1. Дговизна проведения таких работ;

2. Невозможность расширения ввиду банальной нехватки места в условиях города, когда нельзя расширить ширину дорожного полотна не прибегая к разрушению прилегающих зданий;

3. Отрицательное влияние на экологию;

4. Присутствие эффекта «бутылочного горла», когда пропускная способность одного участка дороги, не соответствует пропускной способности другого, примыкающего к ней участка с такой же загруженностью.

Рассмотрим еще один способ улучшить движение – строительство дополнительных дорог. Идея схожа с предыдущей, однако, кажется, что она почти лишена проблемы отсутствия места для расширения – почти всегда можно проложить другой путь к этой же точке, такой, при котором не надо будет коренным способом менять инфраструктуру. Однако в данном способе есть свои, не очевидные проблемы. В 1960-х Дитрих Браес обнаружил теоретическую конфигурацию дорог, в которой строительство новой соединительной дороги может замедлить движение каждого, даже если количество машин остаётся постоянным. И наоборот, закрытие одной дороги в сети Браеса позволит всем добираться домой быстрее. Такое явление настолько странно, что заслуживает собственного определения — «Парадокс Браеса». На представленных ниже схемах (Рисунок 1-2) широкие синие дороги, помеченные как *A* и *B*, никогда не оказываются перегруженными; вне зависимости от пропускаемого ими трафика время движения по ним всегда равно одному часу. На узких красных дорогах *a* и *b* время перемещения равно нулю, когда они пусты, но движение увеличивается с повышением нагрузки; если все машины соберутся на одной красной дороге, время движения по ней тоже становится равным одному часу. Золотой маршрут *X* волшебным

образом обеспечивает мгновенную транспортировку любому количеству автомобилей.

Рисунок 1 – Представление маршрутов в виде графа

Менее абстрактно данную ситуацию можно представить в виде следующей схемы (Рисунок 2).

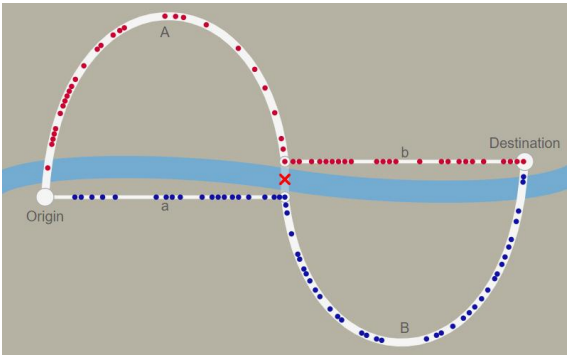


Рисунок 2 – Инфографика маршрута при закрытом пути X

При закрытом пути X автомобили быстрее проходят маршрут в среднем, нежели при открытом X. Граф с открытым маршрутом X представлен на Рисунке 3.

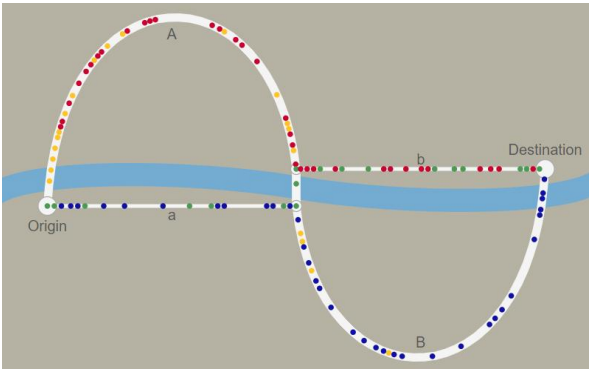


Рисунок 3 – Инфографика маршрутов при открытом пути X

Существование парадокса зависит от наличия дороги X. Без золотого соединения (диаграмма слева) трафик равномерно распределяется между маршрутами Ab и aB, и все автомобили тратят 90 минут на всю поездку. При открытии соединения X (на схеме справа) все водители предпочитают маршрут Ax B и проводят в пути целых два часа. Необходимым предположением для этого парадокса является то, что все стремятся к эгоистичному построению маршрута, выбирая тот путь, который обеспечивает самое быстрое перемещение, и игнорируя все другие факторы, кроме времени движения. Довольно интересно и то, как именно распределяется плотность трафика на определенном участке во времени. В случае закрытого моста, когда общий трафик делится на два приблизительно равных потока, можно предположить, что новые автомобили попеременно выбирают AB или AB, так что система достигает статистически равновесного состояния с одинаковым количеством автомобилей на каждом из двух маршрутов в любой момент времени. Лучший способ проверить это - запустить симуляцию, но график, приведённый на Рисунке 4, также дает нам общее понимание модели.

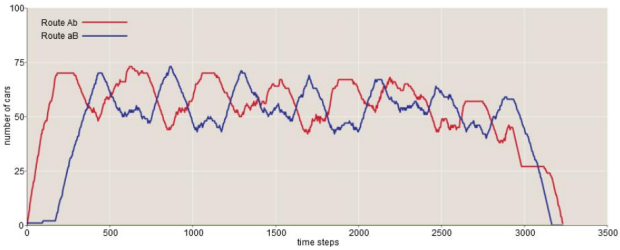


Рисунок 4 – График плотности потока от времени при закрытом пути X

Вместо того, чтобы входить в равновесие, система колеблется с периодом около 500 временных интервалов, что примерно вдвое меньше времени, которое требуется для среднего автомобиля, чтобы перейти из происхождения к месту назначения. Две кривые смещены в фазе почти на 180 градусов. Когда каждый автомобиль входит в дорожную сеть, он выбирает маршрут с кратчайшим ожидаемым временем в пути, основанном на текущем состоянии. Основным решающим фактором ожидаемого времени в пути является количество автомобилей, занимающих разделы A и B, на которых скорость уменьшается при заполнении дорог. Но когда автомобили выбирают менее популярный маршрут, они также увеличивают занятость этого маршрута, что делает его менее желательным для автомобилей, следующих за ними. Кроме того, существует значительная задержка на

маршруте AB до того, как автомобили достигнут сегмента, затронутого переполнением. Задержка и асимметрия сети создают нестабильность - цикл обратной связи, в котором выбросы и чрезмерная коррекция неизбежны. Когда соединительный мост открыт, шаблон становится более сложным, но колебания все еще очень заметны (Рисунок 5).

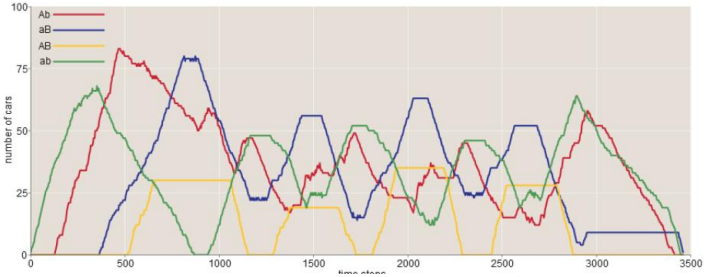


Рисунок 5 - График плотности потока от времени при открытом пути X

Итак, данный способ увеличения скорости трафика требует точного моделирования, как и любое внесение изменений в инфраструктуру. На данный момент, существует много специализированного ПО для этого, например SUMO.

Перейдем к оптимизации дорожного трафика с помощью ИИ и уличных камер. Под данным способом подразумевается обработка изображений с камер посредством ИИ, и регулировка светофоров в соответствии с дорожной ситуацией. По данным исследований в немецком городке Лемго, где применялась такая технология, удалось увеличить скорость дорожного траффика на 10-15%. Сенсоры системы способны отслеживать скопления людей на перекрёстках и соответствующим образом подстраивать фазы. Если же среди пешеходов есть люди с ограниченными возможностями, использующие инвалидные кресла или костыли, то зелёный сигнал для них будет гореть дольше. Исследователи предполагают, что интеллектуальный подход поможет снизить количество переходов в неположенном месте или на красный свет на 25 %. К недостаткам этого способа можно отнести сложность разработки единой системы управления трафиком.

Далее рассмотрим оптимизацию дорожного трафика с использованием синхронизации автопилотируемых авто. Для плавного перехода на данный вид регуляции имеет место так называемая концепция четвёртого сигнала, которая подразумевает дополнение к красному, жёлтому и зелёного цвета ещё и белого, который будет

сообщать водителю «следуй за впереди идущим транспортом»: ехать, если едет он, или стоять, если он стоит. Таким образом беспилотники смогут коммуницировать друг с другом и распределять порядок движения на перекрёстках эффективнее, чем светофоры. В своём моделировании учёные сравнивали движение беспилотных и пилотируемых автомобилей в разных вариациях работы светофора. Выяснилось, что чем больше беспилотников на дороге, тем сильнее сокращается время простоя на перекрёстке, причём с четвёртым белым сигналом результат ещё лучше. Если на беспилотники приходилось 10% участников движения, то задержки снижались всего на 3%, но если их доля составляла 30%, то значение поднималось уже до 10%. Однако исследователи считают, что при 100% доле беспилотных машин задержки могут сократиться более чем на 90%, ввиду того, что полностью будет исключен нюанс с конкурентностью водителей, а функцией оптимизации будет являться общая сумма скоростей водителей. В целом, такой подход также задавать и нужную приоритезацию маршрутов, помогая, например, экстренным службам освобождать дорогу быстрее. Недостатком такого способа является сложность интеграции всего транспорта в единую систему, а также наследуется недостаток, описанный для предыдущего способа.

Подводя итоги, можно сказать, что наиболее приоритетным способом улучшения дорожной ситуации является создание единой системы регулировки трафика, которая будет осуществлять регулировку светофоров, а также корректировать маршруты беспилотных автомобилей, основываясь на данных, предоставляемыми последними, а также спутниковых снимках, видеопотоков с камер наружного наблюдения, а также индукционных петель. Разработка программного обеспечения и внедрение оборудования довольно затратно, однако по сравнению с тратами на строительство дополнительных дорог и расширение уже существующих потребует меньших ресурсов.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Методы оптимизации схемы движения транспортав городе [Электронный ресурс]: URL: https://cyberleninka.ru/article/n/metody-optimizatsii-shemy-dvizheniya-transportav-gorode

2. Оптимизация трафика дорожного движения с использованием компьютерных технологий [Электронный ресурс]: URL: https://www.elibrary.ru/item.asp?id=49994550

3. Оптимизация управления транспортной системой региона [Электронный ресурс]: URL: https://www.mathnet.ru/links/d543ee5da883090cf8ffb4257cfdea92/pfmt431.pdf

4. Модель прогнозирования транспортного потока на основе нейронных сетей для предсказания трафика на дорогах [Электронный ресурс]: URL: https://cyberleninka.ru/article/n/model-prognozirovaniya-transportnogo-potoka-na-osnove-neyronnyh-setey-dlya-predskazaniya-trafika-na-dorogah

УДК 004.032.26

ВЕНЧИКОВА Д.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ВЫБОР ТЕСТОВЫХ ДАННЫХ ДЛЯ РАЗРАБОТКИ СИСТЕМЫ ПЕРСОНАЛИЗАЦИИ БАНКОВСКИХ ПРЕДЛОЖЕНИЙ КРЕДИТОВАНИЯ С ИСПОЛЬЗОВАНИЕМ ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ

В данной статье рассматривается выбор и предварительная обработка тестового датасета для разработки системы персонализации банковских предложений кредитования с использованием искусственной нейронной сети.

Целью работы является разработка системы персонализации банковских предложений кредитования с использованием искусственной нейронной сети.

В настоящее время со стремительно растущими ценами на необходимые для жизни товары, а особенно на недвижимость, кредиты стали еще популярнее. Также из-за финансовой нестабильности многие россияне сталкиваются с неустойками по кредиту и просроченными платежами, что в свою очередь влияет на кредитную историю. Но так как дальнейшее открытие новых кредитов является актуальным для многих людей, то клиент банка, конечно, хочет узнать на какие условия ему могут одобрить кредит.

Существует несколько веб приложений, рассчитывающие скоринговый балл, при этом интегрирующиеся с порталом «Госуслуги». Например, «Сравни.ру». Но результатом проверки является уведомление пользователя о балле и общая характеристика возможно одобренных кредитов – «Вы сможете оформить кредит по более низкой ставке», а точную информацию по желаемому кредиту не предоставляется. То есть после понимания, на что может

рассчитывать, пользователь подает заявку в желаемый банк онлайн или в офисе банка, точно не зная получит ли он в итоге кредит.

Поэтому целью данной работы является разработка системы персонализации банковских предложений, с помощью которой можно узнать скоринговый балл и точные одобренные условия кредитования по желаемому кредиту.

Первой задачей в прогнозировании кредитных предложений для клиента является проведение кредитного скоринга.

Кредитный скоринг — это метод анализа, который банки и другие финансовые организации используют для оценки рисков при выдаче кредитов. Скоринговая оценка основывается на информации о кредитной истории клиента, его финансовом положении и других факторах.

Скоринг позволяет просчитать вероятность того, что заемщик вернет свой кредит в срок, и основывается на статистических моделях и математических алгоритмах, а не на субъективных человеческих оценках.

Расчет баллов кредитного скоринга и так производится во многих банках с помощью нейронных сетей, но параметры, участвующие в расчете, и методы анализа не разглашаются.

Поэтому одной из частей дальнейшей дипломной работы является разработка алгоритма обучения нейронной сети для подсчета баллов кредитного скоринга с использованием рекурентных нейронных сетей (RNN). В качестве обучающего набора взят датасет из kaggle, который содержит информацию о 614 заемщиках. Каждый клиент описывается 12 признаками такими как: семейный статус, пол, количество детей, наличие высшего образования, доход и другие. Целевым признаком является статус платежеспособности клиента.

- Gender – Пол бинарная переменная мужской или женский;
- Married – Семейное положение бинарная переменная в браке или нет;
- Dependents – Количество детей;
- Education – Наличие высшего образования бинарная переменная есть или нет;
- Self_Employed – Вид занятости бинарная переменная самозанятый или нет;
- Applicantlncome – Доход заемщика;
- Coapplicantlncome – Доход супруга клиента;
- Loan_Amount – Сумма займа;
- Loan_Amount_Term – Срок займа;
- Credit_History – Кредитная история бинарная переменная удовлетворяет или нет;

- Property_Area – Область проживания номинативная переменная город, посёлок или деревня
- Loan_Status – Статус заёмщика.

Возможно, при дальнейшем анализе данных датасет будет расширяться или, наоборот, сужаться используемыми для прогнозирования данными. Также потребуется очистить данные от пустых значений и выбросов.

Изначально датасет имеет вид, представленный на Рисунке 1:

Loan_ID	Gender	Married	Dependent	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
LP001002	Male	No	0	Graduate	No	5849	0		360	1	Urban	Y
LP001003	Male	Yes	1	Graduate	No	4583	1508	128	360	1	Rural	N
LP001005	Male	Yes	0	Graduate	Yes	3000	0	66	360	1	Urban	Y
LP001006	Male	Yes	0	Not Graduate	No	2583	2358	120	360	1	Urban	Y
LP001008	Male	No	0	Graduate	No	6000	0	141	360	1	Urban	Y
LP001011	Male	Yes	2	Graduate	Yes	5417	4196	267	360	1	Urban	Y
LP001013	Male	Yes	0	Not Graduate	No	2333	1516	95	360	1	Urban	Y
LP001014	Male	Yes	3	Graduate	No	3036	2504	158	360	0	Semiurban	N
LP001018	Male	Yes	2	Graduate	No	4006	1526	168	360	1	Urban	Y
LP001020	Male	Yes	1	Graduate	No	12841	10968	349	360	1	Semiurban	N
LP001024	Male	Yes	2	Graduate	No	3200	700	70	360	1	Urban	Y

Рисунок 1 – Пример датасета

Для дальнейшей работы с датасетом требуется привести все данные к числовому виду. Таким образом, требуется изменить данные для столбцов Gender, Married, Education, Self_Employed, Property_Area и Loan_Status.Примем, что значения “Yes” принимаются за единицу, а “No” – за ноль, мужской пол примем за значение «0», а женский за «1». Также наличия образования – значение “Graduate” принимаем за единицу, а отсутствие образования – “ Not Graduate” принимаем за ноль. Для места проживания введем следующие обозначения: город – 1, поселок -2, деревня – 3.

Обработанный датасет представлен на Рисунке 2:

Loan_ID	Gender	Married	Dependent	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
LP001002	0	0	0	1	0	5849	0		360	1	1	1
LP001003	0	1	1	1	0	4583	1508	128	360	1	3	0
LP001005	0	1	0	1	1	3000	0	66	360	1	1	1
LP001006	0	1	0	0	0	2583	2358	120	360	1	1	1
LP001008	0	0	0	1	0	6000	0	141	360	1	1	1
LP001011	0	1	2	1	1	5417	4196	267	360	1	1	1
LP001013	0	1	0	0	0	2333	1516	95	360	1	1	1
LP001014	0	1	3	1	0	3036	2504	158	360	0	2	0
LP001018	0	1	2	1	0	4006	1526	168	360	1	1	1
LP001020	0	1	1	1	0	12841	10968	349	360	1	2	0
LP001024	0	1	2	1	0	3200	700	70	360	1	1	1
LP001027	0	1	2	1		2500	1840	109	360	1	1	1
LP001028	0	1	2	1	0	3073	8106	200	360	1	1	1
LP001029	0	0	0	1	0	1853	2840	114	360	1	2	0
LP001030	0	1	2	1	0	1299	1086	17	120	1	1	1
LP001032	0	0	0	1	0	4950	0	125	360	1	1	1
LP001034	0	0	1	0	0	3596	0	100	240		1	1
LP001036	1	0	0	1	0	3510	0	76	360	0	1	0

Рисунок 2 – Пример обновленного датасета

В рамках следующего этапа исследования была выполнена интерпретация базы данных, а именно визуализация и осуществление анализа данных для определения взаимосвязей между признаками и

выявления скрытых признаков с использованием методов машинного обучения, реализованных с помощью языка программирования Python и библиотеки matplotlib. Результат представлен на Рисунке 3.

Рисунок 3 – Матрица корреляции

В результате максимальная взаимосвязь выявлена у пар значений «Кредитная история» - «Статус заемщика» (0,54) и «Доход заемщика» - «Сумма займа» (0,56). Также приемлемая взаимосвязь получена у пары значений «Семейное положение» - «Количество детей» (0,34), но данная пара не особо важна для исследования одобрения кредита. Таким образом, для дальнейшей работы требуется расширить датасет и провести дополнительные исследования для выявления взаимосвязи между данными.

Результатом описанной работы является подготовка начального датасета, его преобразование и начальное исследование данных.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Скоринг в банке: что такое кредитный скоринг простыми словами [Электронный ресурс]: URL: https://www.sberbank.ru/ru/person/blog/chto-takoe-skoring-v-banke

2. Loan Predication [Электронный ресурс]: URL: https://www.kaggle.com/datasets/ninzaami/loan-predication

УДК 004.032.26

ВЕНЧИКОВА Д.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ПРИМЕНЕНИЕ ДЕРЕВА РЕШЕНИЙ ДЛЯ РАЗРАБОТКИ СИСТЕМЫ ПЕРСОНАЛИЗАЦИИ БАНКОВСКИХ ПРЕДЛОЖЕНИЙ КРЕДИТОВАНИЯ С ИСПОЛЬЗОВАНИЕМ ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ

В данной статье рассматривается выбор модели для разработки системы персонализации банковских предложений кредитования с использованием искусственной нейронной сети. В качестве модели был выбран алгоритм построения дерева решений.

Проблема правильной оценки кредитного потенциала заёмщика является актуальной, как для стороны банка, так и для стороны заёмщика. Банк хочет себя обезопасить себя от клиентов, не исполняющих обязательства перед банком, и подстраховать себя в случае потенциально не надёжного заемщика повышенной ставкой и меньшей одобренной суммой. Заёмщик же в свою очередь рассчитывает на одобрение кредита и получение наиболее выгодных условий. Также потенциальные заемщики, зная, что они могут не пройти по каким-то критериям, пытаются подделать результаты. Также есть и обратная сторона, что при расчете всех рисков, банк не должен потерять надежных клиентов. Таким образом, определение кредитоспособности потенциального клиента является важной задачей для банков, и должна сводиться к наименьшей погрешности.

Задача сводится к задаче бинарной классификации – одобрено или не одобрено кредитование. В качестве метода анализа данных было решено использовать дерево решений, так как данный метод предназначен для задач классификации, достаточно прост и позволяет добиться большой точности.

Дерево решений – это способ представления правил принятия решений в иерархической структуре, состоящей из узлов и листьев. Узлы содержат правила принятия решений и осуществляют проверку примеров на соответствие определенному атрибуту обучающего набора данных.

Построение дерева начинается с корневого узла, узлов и листьев. Узлами являются условия для проверки, а листьями – результат.

В общем виде дерево решений представлено на Рисунке 1:

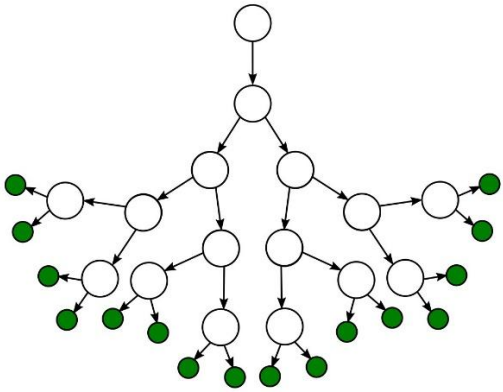


Рисунок 1 – Общий вид дерева решений

В простейшем случае, после проверки узла дерева решений, примеры данных разделяются на два подмножества в зависимости от их удовлетворения определенному правилу. Эти подмножества затем подвергаются новому правилу в каждом следующем узле, что приводит к рекурсивному разбиению, пока не будет достигнуто условие остановки алгоритма. В конечном итоге, в листе не производится дальнейшего разбиения, и он становится решающим "узлом", который определяет соответствующий класс для каждого примера данных в нем. Соответственно, дойти до каждого листа можно только одним путем, поэтому данный алгоритм избавляет от неопределенности.

Для построения дерева решений можно выделить следующие этапы:

1. Выбор атрибута разбиения
2. Выбор критерия остановки обучения.
3. Выбор метода отсечения ветвей.
4. Оценка точности построенного дерева.

Остановимся на каждом этапе поподробнее.

При выборе атрибута для разбиения в узле дерева, необходимо учитывать, чтобы результаты подмножеств содержали примеры с одинаковыми метками класса или были максимально близки к этому, минимизируя количество объектов из других классов (примесей). Для выбора атрибута воспользуемся теоретико-информационным критерием - информационной энтропией.

Энтропия вычисляется по формуле 1:

$$H = -\sum_{i=1}^{n} \frac{N_i}{n} \log(\frac{N_i}{n}) \quad (1)$$

где *n* — число классов в исходном подмножестве, *Ni* — число примеров i-го класса, *N* — общее число примеров в подмножестве.

Таким образом, энтропия может рассматриваться как мера неоднородности подмножества по представленным в нём классам. Если все примеры в узле относятся к одному классу, т.е. *N=Ni*, логарифм от единицы обращает энтропию в ноль. Соответственно, чем больше примесей, тем выше энтропия.

Но на практике обычно применяется не энтропия, а «информация» - величина, обратная энтропии (2):

$$Gain(A) = Info(S) - Info(S_A) \quad (2)$$

где Info(S) — информация, связанная с подмножеством S до разбиения, Info(S_A) — информация, связанная с подмножеством, полученными при разбиении по атрибуту A.

Таким образом, чем выше Gain(A), то есть прирост информации, тем качественнее разбиты данные.

Основным условием для остановки алгоритма является разбиение обучающих данных на подмножества без примесей, то есть в каждом множестве остались примеры одного класса. Но при таком результате может произойти переобучение. Для каждого примера из датасета будет создан отдельный лист. Очевидно, что практической пользы от построения такого дерева нет. Поэтому используется принудительная остановка построения дерева. Существует несколько подходов:

1. Ранняя остановка. Алгоритм будет остановлен, когда будет достигнуто предельное значение какого-то критерия. Например, процент правильно классифицированных примеров в подмножестве. Преимуществом метода является экономия времени, но при этом страдает точность дерева.

2. Ограничение глубины дерева. Алгоритм заканчивается при достижении максимально возможного числа разбиений в ветвях. Также страдает точность дерева.

3. Отсечение ветвей. После построения полного дерева по заранее определенным относительной точности модели (отношение числа правильно распознанных примеров к общему числу примеров) и абсолютной ошибки (число неправильно классифицированных примеров) от дерева отсекаются листья и узлы, которые не приведут к значимому уменьшению точности модели или увеличению ошибки.

Отсечение ветвей происходит снизу вверх, путём последовательного преобразования узлов в листья. Преимуществом данного подхода является возможность поиска оптимального соотношения между точностью и понятностью дерева. К недостаткам относится долгое время обучения.

Иногда даже упрощенное дерево решений может быть слишком запутанным для понимания и восприятия. В таких случаях полезно извлечь правила принятия решений из дерева и организовать их в наборы, описывающие классы.

Для извлечения правил необходимо проследить пути от корневого узла до листьев дерева. Каждый путь дает правило, состоящее из условий, которые проверяются на каждом шаге пути.

Представление сложных деревьев решений в виде наборов решающих правил вместо иерархической структуры из узлов и листьев может быть более удобным для визуализации и понимания.

Таким образом, в данной работе был изучен алгоритм построения дерева решений, выделены его преимущества для решения задачи кредитного скоринга. На практике для построения дерева решения можно воспользоваться средствами библиотеки sklearn на языке высокого уровня Python.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Бабаев А.М., Шемякина М.А. Обзор классических методов машинного обучения в контексте решения задач классификации // Форум молодых ученых 11(27), 2018, с. 137-143

2. Деревья решений: общие принципы [Электронный ресурс]: URL: https://loginom.ru/blog/decision-tree-p1

3. Открытый курс машинного обучения. Тема 3. Классификация, деревья решений и метод ближайших соседей / Хабр [Электронный ресурс]: URL: https://habr.com/ru/companies/ods/articles/322534/

УДК 04.004

ГОСТЕВ С.Ю.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

ИНФОРМАЦИОННАЯ СИСТЕМА СЛУЖБЫ ПОДДЕРЖКИ ПОЛЬЗОВАТЕЛЕЙ

Одной из функций жизнеобеспечения крупных организаций является обеспечение комфортной и бесперебойной работы сотрудников организации, для чего в структуре

предприятий выделяется подразделение службы технической поддержки, реагирующее на заявки от пользователей. Для оптимизации обработки инцидентов – запросов от пользователей на техническую поддержку – применяются программные комплексы – HelpDesk – информационные системы поддержки, построение логической модели которой и является целью настоящей статьи.

До машстабного развития информационных систем, подход к решению проблем сотрудниками крупных предприятий сводился к попыткам (не всегда успешным) самостоятельного решения, либо же обращение к более опытным коллегам, зачастую не имевших квалификации в данном вопросе. Даже выделение отдельной службы технической поддержки на предприятии в полной мере не могло исключить данный фактор, так как, традиционно, в основе взаимодействия пользователей и сотрудников службы поддержки лежали горизонтальные связи. Появление информационных HelpDesk-систем позволило скрыть от пользователя процесс решения создаваемого им инцидента (возникшей проблемы), представляя таким образом службу поддержки, как некоторый «черный ящик», в который передается заявка и через некоторое время проблема решается. При этом, пользователи системы не испытывают психологических барьеров, которые могут возникать при личном обращении к сотрудникам службы, и, в то же время, каждый инцидент решается специалистом, имеющим наибольшую компетентность в конкретном вопросе.

Особенностью HelpDesk-систем является наличие унифицированного интерфейса для регистрации пользовательского инцидента, в связи с чем независимо от возникшей проблемы, пользователь выполняет одни и те же действия для уведомления службы поддержки.

Процесс контроля за затратами времени, человеческих и иных ресурсов, осуществляется руководителем подразделения, осуществляющего техническую поддержку, без уведомления создавшего инцидент пользователя, что позволяет оптимизировать загрузку службы технической поддержки. Уведомление пользователя о ходе работ может производиться только в случае превышения регламентного времени на обработку инцидентов.

Одним из важных механизмов HelpDesk-системы является механизм ранжирования приоритетов инцидентов в зависимости от создавшего их пользователя, типа инцидента или других обстоятельств, связанных со спецификой конкретного предприятия. Данный механизм позволяет обрабатывать инциденты за статистически приемлемые временные промежутки.

Еще один механизм, реализуемый HelpDesk-системой – механизм эскалации – используется для передачи инцидента вверх по уровню службы поддержки в случае недостаточной компетенции текущего уровня.

Хранение разрешенных инцидентов и механизмов их разрешения в базах данных обеспечивает возможность быстрого доступа специалистов службы технической поддержки через HelpDesk-интерфейс к пути решения аналогичных инцидентов и сокращение времени разрешения проблемы. Помимо этого, подобный подход позволяет снизить требуемый уровень компетенции для решения ряда уже разрешенных инцидентов, снижая таким образом часть запросов, проходящих через механизм эскалации и сократить время разрешения, повышая, таким образом, эффективность работы службы технической поддержки.

Помимо процессов, связанных с обработкой и архивированием инцидентов, HelpDesk-системы позволяют формировать разнообразную отчетность по разрешенным проблемам, что позволяет использовать численные статистические метрики качества работы службы поддержки, выявлять на основе потока инцидентов закономерности, соответствующие «узким местам» инфраструктуры предприятия. Ввиду возможности интеграции со средствами материального учета, на HelpDesk-систему может быть возложен функционал контроля за соответствием используемого оборудования решаемым задачам с целью своевременного ремонта, замены или модернизации.

Проектирование информационных систем может производиться с учетом различных парадигм и моделей – как функциональных, так и объектно- или аспектно-ориентированных. В настоящей статье рассматривается объектно-ориентированная модель HelpDesk-системы. Для описания взаимодействия специфических типов данных – классов – системы использована диаграмма классов в UML-нотации, показанная на рисунке 1.

Объектно-ориентированный подход позволяет перейти от разделенного использования данных и методов их обработки к использованию единых структур, совмещающих данные и методы – классов [1]. Данный подход позволяет с легкостью переходить от диаграмм процессов к программам на языках высокого уровня, таких, как C#.

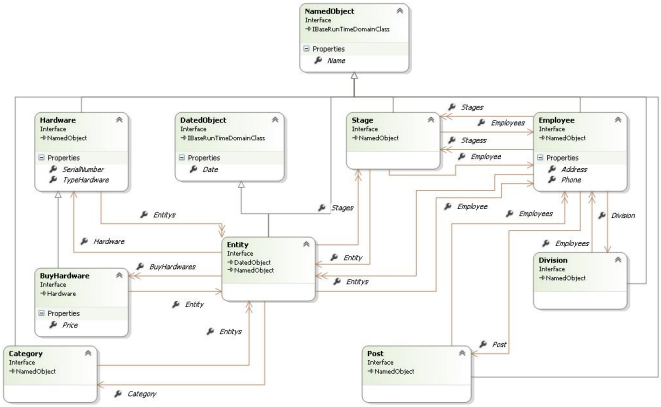


Рисунок 1 – Объектно-ориентированная модель HelpDesk-системы

Построению логической ООП-модели, приведенной на рис. 1, было выполнено выделение сущностей и связей, характеризующих предметную область предприятия и их атрибутов.

Поскольку хранение данных информационной системы производится посредством реляционной базы данных, диаграмма «сущность-связь» которой содержит элементы, несовместимые с объектно-ориентированной парадигмой, необходимо выполнить преобразование атрибутированных и сложных связей в отдельные классы и интерфейсы, а также описать правила поведения системы.

Рассмотрим разработанную объектную модель (рис. 1). Предметная область требует наличия подробного описания, для чего создан базовый абстрактный класс NamedObject со строковым полем Name – от данного класса наследуются все типы данных, требующих описания для анализа сотрудниками службы поддержки. Аналогично, для объектов, требующих описания временных рамок – описано наследование от абстрактного класса DatedObject с единственным полем Date. Таким образом, осуществляется возможность ранжирования по датам и описанию объектов классов предметной области.

Описание оборудования, используемого как пользователями, так и сотрудниками службы поддержки реализован класс Hardware с необходимым набором полей, от которого наследуется класс BuyHardware – список оборудования, закупленного под конкретную задачу (инцидент).

Инцидент (заявка) описывается классом Entity, наследуемым как от NamedObject, так и от DatedObject. Под инцидент может быть закуплено оборудование, соответственно класс BuyHardware

наследуется и от Entity.

Дополнительно в объектной модели формируется структура предприятия, где подразделения описываются классом Division, сотрудники – списком объектов класса Employee, наследуемых от класса должностей Post.

Анализ последовательностей наследования между классами предметной области позволяет выявить взаимосвязи экземпляров, и реализовать серверную часть HelpDesk-системы с помощью, например, среды быстрой разработки информаицонных систем SharpArchitect RAD Studio [2, 3].

В заключении статьи необходимо отметить, что рассматриваемая на рис. 1 объектная модель является базовой, то есть отражает только идеологию системы, и не является законченным прототипом, пригодным к полноценному внедрению HelpDesk-системы на предприятии. Для внедрения, а также адаптации к конкретной деятельности целевого предприятия требуется введение новых производных и базовых классов с соответствующими методами обработки их полей.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений. – Вильямс, 2010.

2. Олейник П.П. Элементы среды разработки программных комплексов на основе организации метамодели объектной системы // Бизнес-информатика. 2013. №4(26). – С. 69-76.

3. Олейник П.П. Программа для ЭВМ "Унифицированная среда быстрой разработки корпоративных информационных систем SharpArchitect RAD Studio", свидетельство о государственной регистрации № 2013618212 от 04 сентября 2013 г.

УДК 004.896

ГОСТЕВ С.Ю.

Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

СФЕРЫ ИСПОЛЬЗОВАНИЯ ЧАТ-БОТОВ

На настоящем этапе развития научно-технического процесса значительно возрос спектр услуг, в которых возрастает доля замены человека автоматизированной системой или роботом. В сфере информационных услуг, в связи с этим, наибольшее распространение приобрели голосовые помощники и текстовые ассистенты – чат-

боты. В настоящей статье рассматриваются существующие и возможные сферы их применения.

На настоящий момент рынок информационных технологий как РФ, так и в общемировом масштабе, испытывает всплеск, причем, в рамках маркетинговых исследований показано, что рост в течение ближайших трех лет будет достигать 30% ежегодно, в связи с чем происходит расширение данного направления разработки. Помимо повышения качества работы чат-ботов, интерес представляет их использование как в уже освоенных, так и в новых областях.

Исследования компании Accenture выделяют пять трендов, оказывающих влияние на рост рынка чат-ботов:

- Развитие нейронных сетей и методов машинного обучения, используемых в системах искусственного интеллекта;
- Рост использования мессенджеров;
- Снижение сложности и стоимости разработки;
- Осуществление взаимодействия между ботами;
- Платежи с помощью бота.

Рассмотрим каждый из аспектов в отдельности. В применении к чат-ботам систем искусственного интеллекта основной задачей является качество чат-бота с точки зрения прохождения теста Тьюринга – неотличимости для человека чат-бота от человека-оператора. На настоящий момент достигнуты определенные успехи в развитии нейросетей генерации текста [1], которые уже формируют связный текст достаточного качества, однако, вызывающий некоторый дискомфорт при чтении, поскольку текст, особенно превышающий по длине одно-два предложения, строится по типовым шаблонам. Данный недостаток связан, во многом, с обучающей выборкой, на которой формировались алгоритмы, например ChatGPT. Поскольку в качестве обучающих текстов использовались материалы Википедии, статьи которой строятся в определенной стилистике, сформированы шаблоны, вызывающие раздражение у пользователей. Выходом из данной ситуации, в применении к чат-ботам, является обучение нейронных сетей на истории переписок квалифицированных операторов коммерческих и производственных чатов с пользователями.

Рост использования мессенджеров, во многом, определяет тренды развития цифровой экономики, ввиду массового распространения мобильных устройств с возможностью интернет-коммуникаций. Как показывает статистика ([2]), на настоящий момент даже в РФ среднее количество мобильных устройств в семьях составляет 2,5, в то время как этот же показатель для персональных компьютеров не превышает 0,3. При этом на настоящий момент звонки

с использованием сотовых сетей совершаются гораздо реже, чем общение, как текстовое, так и голосовое, в мессенджерах. Данный фактор приводит к кардинальным изменениям в маркетинговых стратегиях предприятий, с отходом от рекламы в традиционных СМИ к использованию, в том числе и рекламы в публичных чат-ботах. Соответственно, частично использование чат-ботов в деятельности предприятия может быть профинансировано за счет рекламной деятельности.

Наиболее востребованной в настоящий момент сферой применения виртуальных собеседников является образовательный процесс [3]. Задачей чат-бота в образовании является как сбор ответов учащихся и их систематизация, так и возможность использования в качестве ментора в некритичных областях знаний. Например, возможно применение чат-ботов в изучении иностранных языков, когда имитируется общение пользователя с носителем языка и коррекцией ошибок пользователя в процессе общения. На настоящий момент технологии чат-ботов используются в большинстве высших учебных заведений в РФ. Тем не менее, данное направление ограничено только теми областями знаний, где возможно общение в структурированной форме посредством не слишком длинных сообщений. Те области обучения, в которых как для получения, так и для контроля знаний требуется использование, например, графической информации, на настоящий момент не могут быть автоматизированы в достаточной мере. Тем не менее, и в этих отраслях возможно использование интеллектуальных чатов для решения организационных вопросов и обеспечения более комфортной коммуникации студента и преподавателя. Так, в ряде высших учебных заведений чат-боты помогают студентам спланировать написание и защиту работ промежуточного и финального тестирования, подобрать необходимые методические материалы, осуществлять промежуточный контроль прохождения практик. Не исключено, что в ближайшие годы данная технология позволит в значительной мере снизить бюрократическую нагрузку на преподавателей.

Одной из существующих сфер применения чат-ботов, на настоящий момент, является организация культурно-массовых мероприятий. Так как данная отрасль, как правило, действует в рамках определенных регламентов, то продажа билетов, распределение мест, бронь гостиниц и подобные бизнес-процессы могут быть автоматизированы простейшими чат-ботами, что позволяет разгрузить администраторов мероприятий, снизить долю ошибок, вызываемую человеческим фактором, и снижения накладных расходов на

предприятие.

Еще одним перспективным направлением внедрения автоматизированных чат-ботов является медицина. На настоящий момент, ввиду наличия развернутых протоколов диагностирования и лечения большинства распространенных заболеваний, виртуальный помощник может выполнить, в ряде случаев, роль терапевта районной поликлиники, на основе проведения опроса и результатов базовых и дополнительных анализов направив пациента к профильному специалисту для окончательной постановки диагноза. В полной мере использование чат-бота позволяет реализовать функции «виртуальной регистратуры», работающей круглосуточно, и без выходных и праздников. Применение чат-ботов для напоминания о приеме лекарств и медицинских процедурах, совместно со сбором отзывов о деятельности медицинского учреждения, позволит не только контролировать, но и повышать качество обслуживания.

Наиболее освоенным на настоящий момент направлением применения чат-ботов является финансовая и банковская сфера. История использования автоматизированных помощников в службах технической и пользовательской сферы организаций данного направления насчитывает порядка 10 лет. При этом внедрение искусственного интеллекта требует определенной осторожности. Обучение на открытых данных приводит к ситуациям, когда, например, виртуальные текстовые и голосовые помощники проявляют некорректное поведение по отношению к пользователям. К тому же, необходимо учитывать, что современные технологии обучения нейронных сетей в условиях неполных данных, зачастую генерируют ошибочные ответы на вопросы, что в банковской сфере может приводить к оттоку клиентов. Выходом из ситуации является обучение на закрытом массиве данных и формирование соединения с человеком-оператором в случае отсутствия достоверно корректного ответа на пользовательский вопрос. Тем не менее, на настоящий момент, доля расходов банковской сферы, обусловленная содержанием штата сотрудников службы технической поддержки, сократилась на 60%, в то время, как именно этот штат составлял до 80% сотрудников крупных банковских сетей. Хотя с точки зрения занятости населения, данный фактор является скорее отрицательным, существует надежда на снижение стоимости банковских услуг, что может благотворно сказаться на экономике государства в целом.

Рассмотренные отрасли применения чат-ботов предусматривают модель взаимодействия «бот-пользователь», однако существует направление, позволяющее повышать эффективность технологии за

счет модели взаимодействия «бот-бот». Например, в случае отсутствия ответа на вопрос пользователя в собственной базе знаний, возможно осуществление рассылки вопроса от имени одного бота другим виртуальным помощникам соответствующих сфер деятельности. При этом существующая модель чат-бота может обучаться за счет полученных ответов. Необходимым условием успеха подобных взаимодействий является высокая надежность и добросовестность контрагентов, либо взаимодействие чат-ботов единой сети.

В заключение можем сделать вывод, что сфера применения чат-ботов, в свете развития систем искусственного интеллекта, расширяется до практически всех областей народного хозяйства. Ввиду невысокой удельной стоимости запроса к боту, в сравнении с человеком-оператором, повсеместное внедрение чат-ботов позволит снизить цены на большинство услуг, связанных с взаимодействием с пользователем.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Солдатенкова Ю.А. YandexGPT и ChatGPT: характеристика, сравнение и основные отличия нейросетей / Ю.А. Солдатенкова, А.В. Свищев // Моя профессиональная карьера. – 2023. – Т. 3, № 55. – С. 277-284.

2. Аристова А.С. Использование чат-ботов в образовательном процессе / А.С. Аристова, Ю.С. Безносюк, П.К. Ведикер, Н.Е. Воронович // The 2th International Conference on Digitalization of (DSEME-2019). – Екатеринбург, 2019. – С.95-99.

3. Олешкевич К.И. Технологии чат-ботов как современный инструмент продвижения услуг в сфере культуры / К.И. Олешкевич, В.А. Шурмакова // Scientific collection «Interconf». – 2021. – № 41.

УДК 004.65

ГРУШКИН С.Г.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ПОСТАНОВКА ЗАДАЧИ О НАЗНАЧЕНИЯХ И АЛГОРИТМЫ ЕЁ РЕШЕНИЯ

Рассматривается задача о назначениях. Описываются венгерский алгоритм и алгоритм полного перебора для решения задачи о назначениях. Приводится сравнительный анализ указанных алгоритмов.

Практическая постановка задачи о назначениях заключается в том, чтобы оптимально распределить ресурсы (например, задачи, оборудование, персонал) по определённому множеству объектов (например, проекты, работы, места) с целью максимизации эффективности и минимизации затрат.

Задача о назначениях широко применяется в различных областях, где необходимо распределить ресурсы или задания оптимальны образом. Некоторые из областей задачи о назначениях включают:

1. *Логистика и транспорт.* Задача о назначениях может быть использована для оптимизации распределения грузов и транспортных средств, чтобы минимизировать время доставки и затраты на транспортировку.

2. *Производство.* В производственных предприятиях задача о назначениях может быть применена для оптимального распределения заданий между работниками или оборудованием, чтобы повысить производительность и эффективность процесса производства.

3. *Медицина.* В медицинской сфере задача о назначениях может применяться для оптимизации распределения пациентов по медицинским учреждениями или врачам, чтобы обеспечить быстрое и эффективное обслуживание.

4. *Образование.* В образовательной сфере задача о назначениях может использоваться для распределения учебных заданий среди учителей или студентов, чтобы оптимизировать процесс обучения и повысить успеваемость.

Математическая постановка задачи о назначениях

Пусть имеется n работников, которые должны выполнить n заданий, по одному заданию каждый. Стоимость выполнения i-ым работником j-го задания известна и равна c_{ij} для всех пар $i, j = 1,...,n$. Задача заключается в следующем: надо распределить задания между работниками таким образом, чтобы они были выполнены с наименьшей (наибольшей) стоимостью (прибылью).

Задача о назначениях определяется матрицей стоимости (прибыли) С. Пример такой матрицы представлен в виде таблицы 1.

Таблица 1 – пример матрицы стоимости

	Задание 1	Задание 2	Задание 3	Задание 4
Работник 1	9	2	7	8
Работник 2	6	4	3	7
Работник 3	5	8	1	8
Работник 4	7	6	9	4

Например, если назначить работника 1 на задание 1, работника 2 на задание 2, работника 3 на задание 3 и работника 4 на задание 4, то суммарная стоимость будет: $9+4+1+8=22$. Данное решение не является оптимальны.

Для решения задачи о максимальном назначении необходимо преобразовать матрицу стоимости в матрицу прибыли. Для этого необходимо найти максимальный элемент в матрице и вычесть его из каждого элемента матрицы. Для избавления от отрицательных значений необходимо перемножить каждый элемент на -1. Данное преобразование позволяет использовать венгерский алгоритм для решения задачи о назначениях.

Решить задачу о назначениях можно с помощью алгоритма полного перебора. Данный метод позволяет всегда находить оптимальное решение, но требует большого количества вычислительных ресурсов.

Венгерский метод

В основе венгерского алгоритма лежат эквивалентные преобразования матрицы затрат с неотрицательными коэффициентами с целью получения так называемой системы независимых нулей. Под системой независимых нулей понимаются нулевые элементы матрицы С, каждая пара из которых не содержится в одной строке и в одном столбце.

Для решения задачи о назначениях венгерским алгоритмом необходимы следующие структуры данных:

- двумерный массив целых неотрицательных чисел для хранения матрицы прибыли;
- два двумерных массива индексов для хранения индексов отмеченных нулей;
- два одномерных массива для хранения выделенных строк и столбцов;
- целочисленные переменные.

Венгерский алгоритм включает 2 этапа:

Подготовительный этап. В каждом j-ом столбце находится минимальный элемент γ, который вычитается из каждого элемента j-го столбца.

$$c_{ij}' = c_{ij} - \gamma;\ j = 1,\ldots,n$$

В каждой i-ой строке находится минимальный элемент β, который вычитается из каждого элемента i-ой строки.

$$c_{ij}'' = c_{ij}' - \beta;\ i = 1,\ldots,n$$

После преобразований матрицы в каждой строке и столбце будет находится хотя бы по 1 нулевому элементу. На основе матрицы c'' строится система независимых нулей. Каждый ноль в этой системе отмечается как 0*.

Итерационный этап. На данном этапе алгоритм находит новые нули, которые можно включить в систему независимых нулей, тем самым расширяя её до размеров соответствующих размерности матрицы прибыли.

Данный этап можно описать как последовательность выполнения следующих блоков:

Блок 1. Подсчитываем число независимых нулей k.

Блок 2. Если число независимых нулей 0* равно размерности матрицы прибыли $k = n$, то конец алгоритма, вывод всех 0*. Иначе переход к блоку 3.

Блок 3. Столбцы, которые содержат 0* выделяются знаком “+”.

Блок 4. Обходим невыделенную часть матрицы прибыли и ищем непомеченный 0. Если 0 найден, то переход к блоку 6. Иначе переход к блоку 5.

Блок 5. Находим минимальный элемент s в невыделенной части матрицы. Прибавляем s к каждом элементу из выделенных столбцов и вычитаем s из каждого элемента невыделенных строк. Таким образом получаем непомеченный 0. Переход к блоку 6.

Блок 6. Если в одной строке с найденным непомеченным 0 есть 0*, то переход к блоку 8. Иначе переход к блоку 7.

Блок 7. Отмечаем найденный 0 штрихом 0’. Снимаем выделение “+” со столбца с 0*, находящимся в одной строке с найденным 0. Выделяем строку, содержащую найденный 0 знаком “+”. Переход к блоку 4.

Блок 8. Отмечаем невыделенный ноль штрихом 0’. Переход к блоку 9.

Блок 9. Строим цепочку из выделенных 0. Начинаем с последнего отмеченного штрихом нуля 0’, он становится текущим нулем. После чего ищем 0* в столбце, в котором находится текущий ноль. Если 0* не найден, то заканчиваем построение цепочки. Иначе делаем текущим нулем найденный 0*. Ищем в одной строке с текущим нулем 0’. Такой элемент обязательно найдется, делаем его текущим. Повторяем процедуру пока не достроим цепочку.

В результате получим цепочку, в которой количество 0’ на 1 больше, чем количество 0*.

$$0' \to 0^* \to 0' \to 0^* \to 0'$$

С элементов входящих в полученную цепочку снимаем пометки “*” и заменяем пометки у 0’ на “*”. В результате получена система независимых нулей, которая включает в себя на 1 ноль больше предыдущей.

В результате может получиться, что цепочка включает всего 1 элемент 0’. Такая цепочка называется вырожденной. Она не влияет на порядок выполнения алгоритма и не вызывает исключительных ситуаций.

После получения новой системы независимых нулей снимаем выделением со столбцов и строк матрицы, убираем все пометки у 0’. Переход к блоку 1.

В результате выполнения всех итераций будет получена система независимых нулей, которая включает соответствующее размерности матрицы количество нулей. После чего индексы данных элементов используют для получения варианта назначения согласно начальной матрице стоимости(прибыли). Данный вариант назначения всегда будет оптимальным.

Достоинствами алгоритма являются:

- высокая эффективность (оценка скорости работы $O(n^3)$);
- приемлемые затраты памяти.

Недостатками алгоритма являются:

- необходимость преобразования матрицы, если в ней есть отрицательные значения;
- сложность реализации на ЭВМ.

Алгоритм полного перебора

Идея данного алгоритма заключается в генерации всех перестановок из n элементов. Каждая перестановка рассматривается как вариант назначения. Для каждого варианта считается целевая функция. В качестве решения выбирается перестановка с наибольшим значением целевой функции. В качестве алгоритма для генерации перестановок выбран алгоритм Джонсона-Троттера. В качестве альтернативы можно выбрать любой рекурсивный алгоритм, но стоит помнить о затратах памяти при хранении полного набора перестановок.

Для решения задачи о назначениях методом полного перебора необходимы следующие структуры данных:

- два одномерных массива для хранения перестановок;
- одномерный массив для хранения направлений для алгоритма Джонсона-Троттера;
- целочисленные переменные.

Данный алгоритм можно описать как последовательность выполнения следующих блоков:

Блок 1. Создаём начальную перестановку из n уникальных элементов от 1 до n. Создаём массив направлений размерностью n. Каждый элемент данного массива имеет направление влево. Приравниваем текущую перестановку начальной. Приравниваем оптимальную перестановку начальной.

Блок 2. Используем алгоритм Джонсона-Троттера для получения очередной перестановки из текущей перестановки и массива направлений. В результате выполнения алгоритма получим новую перестановку, которую делаем текущей, и обновлённый массив направлений.

Блок 3. Если текущая перестановка пуста, то конец алгоритма, вывод оптимальной перестановки. Иначе переход к блоку 4.

Блок 4. Рассматривая текущую перестановку как вариант назначения, рассчитываем значение целевой функции.

Блок 5. Если значение целевой функции текущей перестановки выше (ниже) значения целевой функции оптимальной перестановки, то переход к блоку 6. Иначе переход к блоку 2.

Блок 6. Текущее значение целевой функции становится оптимальным, а текущая перестановка становится оптимальной. Переход к блоку 2.

В результате выполнения данного алгоритма будет получена перестановка, которая всегда будет отображать оптимальное назначение.

Достоинствами данного алгоритма являются:

- простота понимания и реализации;
- высокая эффективность при низких значениях размерности матрицы прибыли.

Недостатками данного алгоритма являются:

- скорость работы алгоритма всегда составляет $O(n!)$;
- большие затраты по памяти при использовании рекурсивных алгоритмов генерации перестановок;
- уменьшенная эффективность при использовании итерационных алгоритмов генерации перестановок.

Сравнительный анализ

Сравнивая данные алгоритм по скорости работы можно заметить, что венгерский алгоритм работает гораздо быстрее, т.к. имеет кубическую сложность, когда алгоритм полного перебора факториальную. По затратам памяти алгоритм полного перебора лучше венгерского только в случае низкой размерности матрицы

прибыли или использовании итерационного алгоритма генерации перестановок, такого как алгоритм Джонсона-Троттера.

Для практического анализа данные алгоритмы были реализованы на языке программирования Python. В качестве экспериментальных данных выступали случайно сгенерированные матрицы. В таблице 2 представлены результаты выполнения реализованных алгоритмов на интервале размерностей от 4 до 9. Значения представлены в миллисекундах. В таблице 3 представлены результаты выполнения венгерского алгоритма на значения размерности матрицы от 20 до 100.

Таблица 2 – результаты тестирования алгоритмов

	4	5	6	7	8	9
Венгерский	0,913	0,804	0,978	0,867	1,561	2,153
Полного перебора	0,186	1,006	6,060	45,706	385,656	3769,398

Таблица 3 – результаты тестирования венгерского алгоритма

	20	40	60	80	100
Венгерский	13,738	35,258	107,032	243,025	508,947

Графически результаты тестирования представлены в виде графиков. Таблица 1 представлена на рисунке 1, а таблица 2 на рисунке 2.

По данным экспериментального тестирования можно сделать выводом о том, что алгоритм полного перебора хорошо справляется с матрицами низкой размерности. С ростом размерности видно, что венгерский алгоритм справляется гораздо быстрее алгоритма полного перебора, а при $n = 9$ венгерский справился в 1750 раз быстрее алгоритма полного перебора.

Реализация метода полного перебора с помощью рекурсивных алгоритмов генерации перестановок показала уменьшенные затраты по времени, но значительно увеличенные затраты по памяти, связанные с необходимостью хранения всех перестановок.

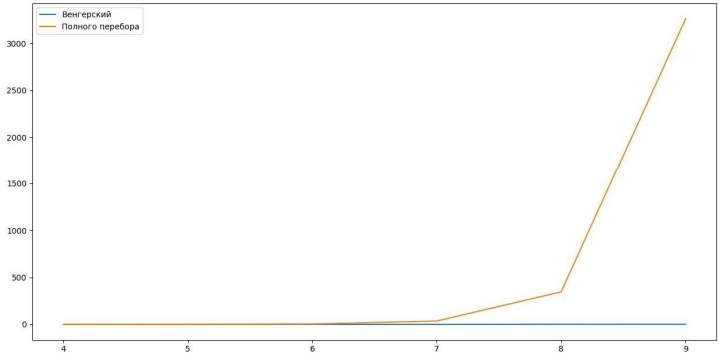


Рисунок 1 – зависимость времени выполнения от размерности

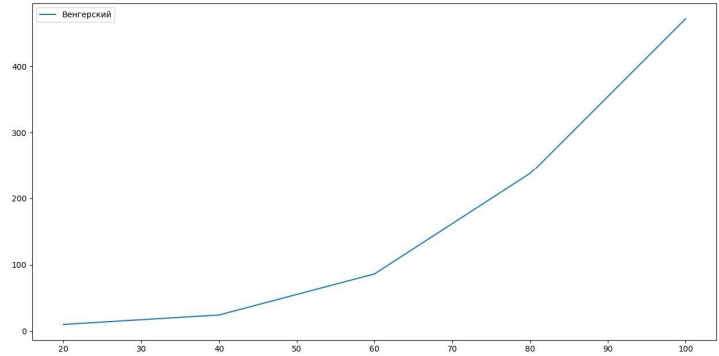


Рисунок 2 – результаты тестирования венгерского алгоритма

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Бакулева М.А., Скворцов С.В., Хрюкин В.И. Методы оптимизации. – Рязань: РФ МЭСИ, 2015. 160 с.

2. Деньдобренко Б.Н., Малика А.С. Автоматизация конструирования РЭА: учебник для вузов. – М.: Высшая школа, 1980. – 384 с.

3. Левитин А. Алгоритмы: введение в разработку и анализ. – М.: Издательский дом “Вильямс”, 2006. – 576 с.

УДК 004.65

ГРЫЗЛОВ В.И.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

GPU И СПОСОБЫ ЕГО ПРИМЕНЕНИЯ

Рассматриваются основные направления использования GPU, примеры внедрения в работу, а также общая информация про GPU.

Введение

За последние годы графические процессоры произвели революцию в мире вычислительной техники. Изначально разработанные для ускорения рендеринга графики в компьютерных играх, графические процессоры превратились в мощные параллельные вычислительные устройства, способные выполнять широкий спектр сложных задач.

Графические процессоры – от машинного обучения и искусственного интеллекта до научного моделирования и майнинга криптовалют - стали незаменимым инструментом для многих отраслей промышленности и приложений.

В этой статье будет рассмотрена история, применяемость и возможности графических процессоров. Также приведены личные исследования для показа скачка быстроты вычислений после открытия GPU.

Что такое GPU

Графический процессор (GPU) — это электронная схема, которая может выполнять математические вычисления с высокой скоростью. Вычислительные задачи, такие как рендеринг графики, машинное обучение и редактирование видео, требуют применения сходных математических операций к большому набору данных. Конструкция графического процессора позволяет параллельно выполнять одну и ту же операцию с несколькими значениями данных. Это повышает эффективность обработки многих задач, требующих больших вычислительных ресурсов.

Современные графические процессоры очень эффективно обрабатывают и отображают компьютерную графику, благодаря специализированной конвейерной архитектуре они намного эффективнее в обработке графической информации, чем типичный центральный процессор.

Отличительными особенностями по сравнению с CPU являются:

- архитектура, максимально нацеленная на увеличение скорости расчёта текстур и сложных графических объектов;
- ограниченный набор команд.

Высокая вычислительная мощность GPU объясняется особенностями архитектуры. Современные CPU содержат небольшое количество ядер (по сравнению с графическими процессорами), тогда как графический процессор изначально создавался как многопоточная структура с множеством ядер. Разница в архитектуре обусловливает и разницу в принципах работы. Если архитектура CPU предполагает последовательную обработку информации, то GPU исторически предназначался для обработки компьютерной графики, поэтому рассчитан на массивно параллельные вычисления

Графический процессор в современных видеоадаптерах применяется в качестве ускорителя трёхмерной графики.

Может применяться как в составе дискретной видеокарты, так и в интегрированных решениях (встроенных в северный мост либо в гибридный процессор).

Историческая справка по GPU

До появления графических процессоров (GPU) для отображения информации использовались матричные экраны, представленные в 1940-х и 1950-х годах. Позже появились векторные и растровые дисплеи, за которыми последовали первые игровые приставки и персональные компьютеры (PC). В то время управление отображением на экране осуществлялось с помощью непрограммируемого устройства, называемого графическим контроллером. Традиционным процессором для этих устройств был центральный процессор, а в некоторых из них использовался встроенный процессор.

Примерно в то же время был реализован проект 3D-визуализации, направленный на создание одного пикселя на экране с использованием одного процессора. Целью было создать изображение, объединяющее несколько пикселей за короткий промежуток времени. В этом проекте использовался графический процессор в том виде, в каком мы его знаем сегодня.

Первые графические процессоры появились только в конце 1990-х годов и предназначались для продажи на рынках игр и систем автоматизированного проектирования (САПР). Графический процессор объединил ранее работавший на программном обеспечении графический движок, механизм преобразования и освещения с графическим контроллером, и все это на программируемом чипе.

В 1999 году Nvidia выпустила первый однокристальный графический процессор GeForce 256, ознаменовав начало эры роста.

Графические процессоры были дополнены такими функциями, как трассировка лучей, затенение ячеек и аппаратная тесселяция, что привело к улучшению генерации изображений и производительности графики.

Только в 2007 году Nvidia выпустила архитектуру CUDA, программный уровень, который сделал доступной параллельную обработку данных на GPU. Примерно в то же время стало ясно, что графические процессоры обладают высокой эффективностью при выполнении конкретных задач, особенно тех, которые требуют значительных вычислительных мощностей для достижения определенного результата.

Выпуск Nvidia CUDA открыл доступ к программированию на графических процессорах для более широкой аудитории. Разработчики смогли использовать технологию графических процессоров для различных практических приложений, требующих значительных вычислительных ресурсов. Вычисления на графических процессорах становились все более распространенными.

Таким образом, до появления графического процессора управление дисплеем осуществлялось с помощью графического контроллера, а традиционными процессорами были центральные процессоры. Первые графические процессоры появились в конце 1990-х годов, а выпуск CUDA в 2007 году открыл возможности программирования на графических процессорах для более широкой аудитории, что привело к использованию графических процессоров для различных практических приложений, требующих значительных вычислительных ресурсов.

Инструменты для работы в коде с GPU

Основные – CUDA и OpenCL на C++, так как CUDA была сделана в первую очередь для этого языка. Для Java сделали Jcuda – обертку в Java язык. Для питона – CuPy и PyTorch, как некоторые варианты. Рассмотрим примеры выше более подробно.

CUDA — это платформа параллельных вычислений и модель интерфейса прикладного программирования (API), созданная компанией Nvidia. Она позволяет разработчикам программного обеспечения использовать графический процессор (GPU) с поддержкой CUDA для обработки данных общего назначения. CUDA предоставляет Си-подобный язык программирования для выражения параллельных вычислений, а также включает в себя библиотеки и инструменты для отладки и оптимизации графических приложений.

OpenCL — это открытый стандарт для параллельного программирования на центральных и графических компьютерах и

других процессорах. Он поддерживается некоммерческим технологическим консорциумом Khronos Group. OpenCL предоставляет C-подобный язык программирования для выражения параллельных вычислений и поддерживается широким спектром производителей оборудования, включая Nvidia, AMD, Intel и ARM.

Jcuda — это библиотека Java для программирования на CUDA. Она предоставляет Java API для CUDA, который позволяет разработчикам Java использовать графические процессоры с поддержкой CUDA для обработки общего назначения. jcuda включает в себя уровень JNI (Java Native Interface) для взаимодействия с CUDA и поддерживает широкий спектр функций CUDA, включая управление памятью графического процессора, выполнение ядра и синхронизацию.

PyTorch — это библиотека машинного обучения с открытым исходным кодом для Python. Она основана на Torch, научной вычислительной платформе для Lua. PyTorch предоставляет динамический вычислительный граф для построения и обучения нейронных сетей и включает в себя серверную часть CUDA для ускорения графического процессора. PyTorch также предоставляет ряд инструментов машинного обучения, включая загрузчики данных, оптимизаторы и функции потери данных.

CuPy — это библиотека массивов для Python с графическим ускорением. Она похожа на NumPy, популярную библиотеку массивов для Python, но использует графический процессор для ускорения. CuPy предоставляет API, совместимый с NumPy, и поддерживает широкий спектр функций графического процессора, включая управление памятью графического процессора, выполнение ядра и синхронизацию. CuPy также поддерживает ряд функций, ускоряемых графическим процессором, включая линейную алгебру, преобразования Фурье и генерацию случайных чисел.

Для разных языков есть инструменты по работе с GPU, как показано ранее, однако большая часть обычно использует связку C++ и CUDA. Если графический процессор не от Nvidea, то тогда используется OpenCL.

Современные области применения GPU. Суперкомьпютеры

Графические процессоры (GPU) стали незаменимыми компонентами во многих приложениях, требующих высокой вычислительной мощности, включая крупномасштабные финансовые, оборонные и исследовательские приложения. Вот некоторые из наиболее распространенных применений графических процессоров на сегодняшний день:

Игры: Первые графические процессоры, не связанные с визуализацией, были использованы в крупных компаниях и государственных учреждениях для персональных игр. Они использовались в игровых приставках в 1980-х годах и до сих пор используются в ПК и современных игровых консолях. Графические процессоры необходимы для сложного графического рендеринга.

Профессиональная визуализация: графические процессоры используются в профессиональных приложениях, таких как системы автоматизированного проектирования (CAD), видеомонтаж, руководства по продуктам и интерактивные приложения, медицинская визуализация и сейсмическая визуализация. Они также используются в других сложных приложениях для редактирования изображений и видео и визуализации. Веб-браузеры могут даже использовать графический процессор с такими библиотеками, как WebGL.

Машинное обучение: Обучение модели машинного обучения требует значительных вычислительных ресурсов. Теперь они могут работать на графических процессорах для получения более быстрых результатов. Хотя обучение модели на самостоятельно приобретенном оборудовании может занять много времени, вы можете быстро достичь результатов, используя графический процессор на базе облачных вычислений.

Блокчейн-технологии: Криптовалюты основаны на блокчейнах. Особый тип блокчейна, Proof-of-Work, обычно сильно зависит от работы графических процессоров. Специализированные микрочипы (ASIC), похожие, но отличающиеся друг от друга чипы, в настоящее время являются распространенной заменой графических процессоров для этого процесса. Блокчейн-алгоритмы Proof-of-Stake устраняют необходимость в огромных вычислительных мощностях, но Proof-of-Work по-прежнему широко используются.

Имитационное моделирование: Передовые приложения для моделирования, такие как приложения, используемые в молекулярной динамике, прогнозировании погоды и астрофизике, могут быть реализованы с использованием графических процессоров. Графические процессоры также являются основой многих приложений для проектирования автомобилей и крупногабаритных транспортных средств, включая гидродинамику.

Многие современные суперкомпьютеры используют графические процессоры. Самый мощный суперкомпьютер по-прежнему находится в США - Frontier, который использует комбинацию процессоров AMD EPYC 64C, оптимизированных для искусственного интеллекта и высокопроизводительных

вычислительных задач, и графических процессоров AMD Instinct MI250X.

Среди отечественных суперкомпьютеров тройка лидеров - "Червоненки", "Галушкин" и "Ляпунов" - использует 8-кратную NVIDIA A100 в сочетании с 2-кратной AMD EPYC разных версий. Все три принадлежат Яндексу.

Суперкомпьютер Национального центра управления обороной России, имеющий уровень производительности в 16 петафлопс и претендующий на звание самого мощного военного суперкомпьютера в мире, не участвует в рейтинге Top-500(Рис. 1). Тем не менее, по состоянию на ноябрь 2021 года это третий по мощности суперкомпьютер в России.

Распределение суперкомпьютеров из списка Top500 по странам мира (ноябрь 2023 года)[9]

Страна	Количество суперкомпьютеров
США	161
Китай	104
Германия	36
Япония	32
Франция	23
Великобритания	15
Италия	12
Южная Корея	12
Нидерланды	10
Канада	10
Бразилия	9
Саудовская Аравия	7
Россия	7
Австралия	6
Швеция	6
Норвегия	5
Тайвань	5
Польша	4
Ирландия	4
Индия	4
Сингапур	3
Швейцария	3
Испания	3

Рисунок 1 – Топ 500 суперкомпьютеров в мире

Исследования возможностей GPU

Для оценки разницы, которую внесли графические процессоры своим появлением, была разработана небольшая программа, одна реализация которой была на центральном процессоре последовательно, вторая – на GPU с помощью CUDA. Цель программы – корреляционный анализ «в лоб».

Программа написана на C++, исполнялась на ноутбуке, Windows 10, CPU – Intel i7-12650H, GPU – NVIDIA RTX 4060 Laptop.

Результаты экспериментов с различными размерами изображений – ниже в таблице 1.

Таблица 1 – Результаты экспериментов

CPU, секунды	GPU, секунды	Ускорение	Размеры ТИ	Размер ЭИ
25,42	0,21	121,05	260x260	58x53
27,98	0,22	127,18	342x342	60x42
39,11	0,23	170,04	400x400	60x50
40,23	0,23	174,91	440x440	60x62
69,45	0,26	267,12	1060x1060	100x100

1. Для большого количества расчетов GPU работает намного быстрее, чем CPU;

2. Для улучшения точности анализа изображения переводятся в серый спектр;

3. Чем больше изображение (больше пикселей) – тем больше ускорение CUDA

Вывод – чем большие размеры у изображения и чем качественнее, тем лучше всего показывает себя расчет через GPU, нежели через CPU.

Заключение

В заключение отметим, что графические процессоры используются в широком спектре приложений, помимо игр и профессиональной визуализации. Они используются в крупных финансовых, оборонных и исследовательских приложениях, а также в машинном обучении, блокчейн-технологиях и имитационном моделировании. Возможность выполнять параллельную обработку данных общего назначения сделала графические процессоры популярными для широкого спектра приложений, требующих больших вычислительных затрат.

Более того, графические процессоры могут использоваться как автономные чипы или интегрироваться с другим вычислительным оборудованием. Их можно арендовать и запускать в облаке, что

упрощает использование их вычислительной мощности для редактирования видео, рендеринга графики, искусственного интеллекта и других возможностей параллельной обработки. Благодаря возможности виртуализации графических процессоров их также можно использовать для выполнения рабочих нагрузок, не беспокоясь о техническом обслуживании базового оборудования.

Многие аспекты вычислений переносятся на видеокарты, так как они намного быстрее высчитывают множество формул одновременно. Это показывает исследование выше. Поэтому GPU часто используются там, где нужны вычисления. К примеру, так называемый «кризис видеокарт» случился потому, что с помощью «ферм» из GPU, которые беспрерывно вычисляли различные значения, зарабатывалась криптовалюта.

Таким образом, графические процессоры стали незаменимыми для широкого спектра приложений, помимо игр и профессиональной визуализации. В настоящее время они используются, в частности, в машинном обучении, блокчейн-технологиях и имитационном моделировании. Вычислительная мощность суперкомпьютеров, в том числе используемых в оборонных и исследовательских целях, также зависит от графических процессоров.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Елесина С.И. Математическое и алгоритмическое обеспечение методов глобальной оптимизации при совмещении изображений: учебное пособие / С.И. Елесина, А.А. Логинов, М.Б. Никифоров. – Рязань: РГРТУ, 2014. – 80 с.

2. Сергеев А.М. О совмещении изображений и способах их реализации // ИВД. – 2022. – №8 (92).

УДК 004.021

ДОРИН А.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

МЕТОД РОЯ ЧАСТИЦ

Рассматривается метод роя частиц и возможность его применения для нахождения оптимального уровня сигнала маршрутизаторов в компьютерной сети.

Алгоритм роя частиц (PSO) [1] был предложен в 1995 году Джеймсом Кеннеди и Расселом Еберхартом. Идея алгоритма была

частично заимствована из исследований поведения скоплений животных (косяков рыб, стай птиц и т.п.), модель была несколько упрощена, а также были добавлены элементы поведения толпы людей, поэтому, в отличие, например, от алгоритма пчел агенты алгоритма (возможные решения) и были названы нейтрально – частицы.

Метод основан на утверждении, что знание заключается в социальном обмене информацией не только между поколениями, но и между элементами одного и того же поколения. Хотя PSO имеет индивидуальные характеристики, этот алгоритм также обладает некоторым сходством с характеристиками, обнаруженными в других вычислительных моделях, основанных на популяции, таких как генетические алгоритмы и другие эволюционные вычислительные методы. Преимущества метода PSO заключается в том, что он относительно прост, и его алгоритм сравнительно легко описать и реализовать.

Его простота и очевидная компетентность в нахождении оптимальных решений в сложных пространствах поиска привели к тому, что алгоритм PSO стал хорошо известен в научном сообществе, что способствовало его изучению и совершенствованию. Таким образом, было предложено множество подходов и протестированы различные приложения, особенно ха последнее десятилетие.

Для того, чтобы понять алгоритм роя частиц, нужно представить себе многомерное пространство (область поиска), в котором располагаются частицы (агенты алгоритма). Начальный шаг алгоритма предполагает расположение частиц случайным образом по всей области поиска. При этом каждая частица в начале имеет нулевой вектор скорости. В каждой точке пространства, где оказывается частица, происходит расчёт значения целевой функции. При этом каждая частица запоминает, какое лучшее значение целевой функции она лично нашла и координаты соответствующей точки в пространстве. Также каждая частица знает где расположена точка, являющаяся лучшей среди всех точек, в которых находились частицы. На каждой последующей итерации алгоритма частицы корректируют свою скорость (модуль и направление), чтобы, с одной стороны, оказаться ближе к лучшей точке, которую частица нашла сама (авторы алгоритма называют этот аспект поведения "ностальгией"), и, в то же время, приблизиться к точке, которая в данный момент является глобально лучшей. Спустя некоторое количество итераций частицы концентрируются в некоторой области около глобального оптимума, однако существует вероятность, что часть частиц может оказаться где-то в относительно неплохом локальном экстремуме, но главное, чтобы

хотя бы одна частица оказалась вблизи глобального экстремума.

Рисунок 1 – Блок-схема метода роя частиц

Самое интересное в алгоритме – это коррекция скорости частицы, именно от этого шага зависит сходимость алгоритма. В первоначальном виде алгоритма коррекция скорости определяется следующим образом:

$$v_{i,j}^{t} = v_{i,j}^{t-1} + \varphi_1 r_{p,j}^{t}(p_{best_i,j}^{t-1} - x_{i,j}^{t-1}) + \varphi_2 r_{g,j}^{t}(g_{best_i,j}^{t-1} - x_{i,j}^{t-1}),$$

где $v_{i,j}^t$ – j-я компонента скорости i-ой частицы на t-ой итерации, φ_1 и φ_2 – весовые коэффициенты, определяемые программистом, r_p^t и r_g^t – независимые случайные векторы, значения компонент которых находятся в диапазоне [0,1].

Перемещение частицы в новую точку области поиска происходит по следующей формуле:

$$x_{i,j}^t = x_{i,j}^{t-1} + v_{i,j}^t$$

По завершению процедуры коррекции рассчитывается значение целевой функции в каждой новой точке, каждая частица проверяет, не стала ли новая координата лучшей среди всех точек, где она побывала. Затем среди всех новых точек проверяется, не нашлось ли новой глобально лучшей точки, и, если это произошло, запоминаются ее координаты и значение целевой функции в ней.

Существуют также различные модификации данного алгоритма [2]. По большей части эти модификации были предназначены для того, чтобы сходимость алгоритма не так сильно зависела от выбора весовых коэффициентов φ_1 и φ_2.

В одной из модификаций алгоритма роя частиц предполагается введение еще одного весового коэффициента, называемого коэффициентом инерции. Таким образом, формула коррекции скорости частицы принимает следующий вид:

$$v_{i,j}^t = wv_{i,j}^{t-1} + \varphi_1 r_{p,j}^t (p_{best_i,j}^{t-1} - x_{i,j}^{t-1}) + \varphi_2 r_{g,j}^t (g_{best_i,j}^{t-1} - x_{i,j}^{t-1}),$$

Введение нового коэффициента способствует более плавному изменению скорости частицы. Коэффициент инерции может иметь константное значение или изменяться от шага к шагу.

Еще одной распространенной модификацией данного алгоритма является канонический алгоритм. В нем вводится нормировка коэффициентов φ_1 и φ_2. Это сделано для того, чтобы уменьшить зависимость сходимости алгоритма от выбора коэффициентов.

Формула коррекции скорости частиц приобретает следующий вид:

$$v_{i,j}^t = \chi[v_{i,j}^{t-1} + \varphi_1 r_{p,j}^t (p_{best_i,j}^{t-1} - x_{i,j}^{t-1}) + \varphi_2 r_{g,j}^t (g_{best_i,j}^{t-1} - x_{i,j}^{t-1})],$$

$$\chi = \frac{2k}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|},$$

$$\varphi = \varphi_1 + \varphi_2, \varphi > 4$$

Коэффициент k определяется значением из интервала (0,1).

Существуют также и такие модификации алгоритма PSO, учитывающие лучшей не точку, в которой каждая частица оказывалась, а точку, которая является лучшей для самой частицы и ближайших ее соседей.

Если говорить о недостатках алгоритма роя частиц, то нельзя не упомянуть, что, как и в других стохастических алгоритмах оптимизации, не гарантируется сходимость алгоритма при конечном количестве частиц. Также по причине ориентации алгоритма на одну лучшую точку увеличивается вероятность остановки алгоритма в относительно неплохом, но не глобальном оптимуме. К плюсам алгоритма можно отнести его простоту и возможность достаточно быстро модифицировать алгоритм на применение иной формулы расчета скорости частиц, если этого требует решаемая задача.

Помимо всего вышеперечисленного, несомненным достоинством данного алгоритма является то, что для его использования не требуется заранее знать точного градиента оптимизируемой функции. Данная особенность позволяет использовать алгоритм роя частиц для определения оптимальных значений уровня сигналов маршрутизаторов в компьютерной сети.

Сформулируем решаемую задачу.

Пусть имеется некоторое множество маршрутизаторов $M = \{m_1,...,m_i\}$, где $i = \overline{1..n}$. Каждый маршрутизатор характеризуется уровнем сигнала p_i, значение которого лежит в диапазоне от 0 до 10. От уровня сигнала зависит площадь зоны покрытия сети. Зона покрытия представляет собой круг с центром в точке нахождения маршрутизатора. Зона перекрытия – область пересечения двух или более зон покрытия. Таким образом, задача сводится к нахождению оптимальных значений уровней сигналов каждого маршрутизатора таких, чтобы суммарная площадь зон покрытия за вычетом штрафов зон перекрытия была максимальной.

Целесообразно будет ввести по отношению к данной задаче систему бонусов и штрафов. Благодаря этому задача оптимизации становится однокритериальной, что и дает возможность использования метода роя частиц. Целевая функция будет иметь вид:

$$C = \sum S_{покрi} - \sum S_{перекрj}$$

Рассмотрим подробнее части этого выражения.

Зона покрытия сетью – зона, в которой уровень сигнала не ниже некоторого граничного значения l. Уровень сигнала в некоторой точке обратно пропорционален квадрату расстояния от этой точки до

источника сигнала. Таким образом, радиус зоны покрытия определяется по формуле $r = \sqrt{\frac{x}{l}}$. Тогда площадь зоны покрытия будет определяться равенством: $S_{покрi} = \pi * r_i^2 = \pi * \frac{x}{l}$.

Перекрытие – зона, в которой действуют несколько сигналов, уровень которых не ниже граничного значения. В этой зоне также есть доступ к сети, однако уровень сигнала ниже из-за помех, возникающих из-за того, что присутствует несколько сигналов на одном канале. Штраф за перекрытие будет определяться как произведение площади перекрытия на величину, обратную значению соотношения сигнал/шум (ОСШ, S/R, SNR).

$$SNR = \frac{P_{сигнал}}{P_{шум}}$$

При этом сигналом считаем сильнейший сигнал, действующий в области перекрытия, а шумом – остальные сигналы в области.

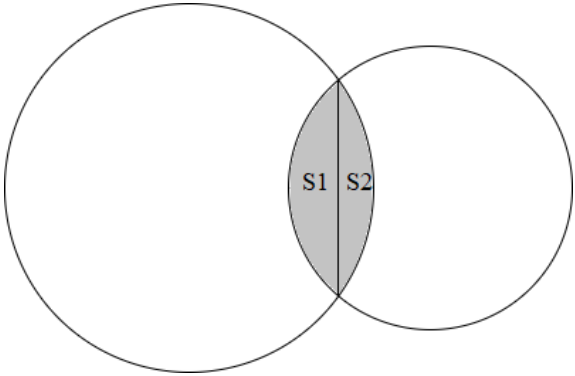


Рисунок 2 – Пример перекрытия

Площадь перекрытия вычисляется следующим образом:

$$S_{перекр} = S_1 + S_2$$

$$S_1 = \frac{R_1^2 * (F_1 - \sin(F_1))}{2}$$

$$S_2 = \frac{R_2^2 * (F_2 - \sin(F_2))}{2},$$

где

$$F_1 = 2 * ar\cos\left(\frac{R_1^2 - R_2^2 + D^2}{2 * R_1 * D}\right)$$

$$F_2 = 2 * ar\cos\left(\frac{R_2^2 - R_1^2 + D^2}{2 * R_2 * D}\right)$$

где R_1 – радиус первой окружности, R_2 – радиус второй окружности, D – расстояние между центрами окружностей.

Штраф за пересечение 3 зон и более определяется как сумма штрафов за перекрытие попарно взятых зон.

Пересечения находятся следующим образом. Рассматриваются все возможные пары зон, проверяется условие пересечения. Если условие выполняется, то находятся точки пересечения зон и рассчитывается площадь зоны перекрытия.

Тогда целевая функция определяется следующим образом:

$$C = \sum_{i=1}^{n} \pi * \frac{x}{l} - \sum\left(S_{перекрk} * \frac{1}{SNR_j}\right)$$

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Kennedy J., Eberhart R. Particle swarm optimization // Proceedings of IEEE International Conference on Neural Networks, vol. 4, pp. 1942-1948, 1995.

2. Маккаффри Д. Искусственный интеллект: Метод роя частиц // Журнал MSDN Magazine Август 2011.

УДК 004.65

ЕНИВАТОВ А.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ОСОБЕННОСТИ ОБУЧЕНИЯ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ИСПОЛЬЗОВАНИЯ В СФЕРЕ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА

В этой статье будет рассматриваться особенности обучения нейронных сетей для использования в сфере образовательного процесса.

За последние несколько лет технологии искусственного интеллекта и машинного обучения претерпели существенное развитие,

что позволило их использовать в различных сферах жизни, включая образование. Одним из вариантов нейронных сетей, применимых для решения задач в сфере образования, является мультимодальная нейронная сеть.

В основе мультимодальности лежит взаимодействие нескольких нейросетей, способных решать различные задачи, и входящих в одну модель, называемую ансамблем. Одной из таких моделей является GigaChat.

GigaChat — это мультимодальная модель, способная создавать текст и графические изображения. За текст отвечают модели ruGPT-3 и FRED-TP, за генерацию изображений — Kandinsky 2.1, а за оценку семантической близости изображений и текста — ruCLIP. Этот ансамбль получил название ***NeONKA*** (NEural Omnimodal Network with Knowledge-Awareness) [3].

По сравнению с другими популярными и известными нейронными сетями, такими как ChatGPT и YandexGPT, GigaChat имеет ряд преимуществ, и выделяется своими уникальными возможностями.

Одно из ключевых преимуществ является то, что GigaChat имеет способность к созданию изображений на основе текстовых запросов. Этот исключительный функционал реализован благодаря использованию модели Kandinsky 2.1. Второе важное преимущество мультимодальной модели GigaChat это то, что она обладает высокой степенью приспосабливаемости и масштабируемости, что делает ее идеальным выбором для решения различных задач в сфере обработки естественного языка.

Следующей важной особенность GigaChat является способность к обучению на небольших объемах данных, что позволяет примененять данную модель в условиях ограниченных ресурсов. На данный момент, модель GigaChat представляет собой мощный инструмент для генерации и инновации в области искусственного интеллекта.

Необходимо осознавать то, что при использовании ИИ в сфере образования следует учитывать особенности:

1. Качество данных: для успешного применения ИИ необходимо наличие качественных данных. Если информация будет неправильно собрана или обработана, то результаты могут оказаться неверными.

2. Этика: использование ИИ в области образования поднимает этические вопросы, которые связанны с конфиденциальностью личной информации обучающихся и студентов, а также с возможностью манипуляции результатами обучения.

3. Обучение преподавателей: для успешного внедрения ИИ в современное образование необходимо проводить обучение учителей и преподавателей работе с новыми технологиями.

4. Оценка эффективности: после внедрения ИИ необходимо провести мониторинг его эффективности и внести необходимые корректировки в дальнейшую программу обучения.

Для обучения GigaChat'a используется метод supervised fine-tuning, reinforcement learning with human feedback (SFT and RLHF) с помощью оценки и анализа ответов. Данный метод обучения включает в себя несколько этапов:

1. Предобучение. Под предобучением нейронной сети понимается её подготовка к работе на больших объёмах информации, до её дообучения для решения конкретных задач. Это позволит нейронной сети оперировать куда большими объемами закономерностями на этапе дообучения.

2. Дообучение прошлой языковой модели на первом типе размеченных данных — с готовыми ответами (Supervised fine-tuning). Что позволяет настроить нейронную сеть на решение конвертной задачи.

3. Обучение модели вознаграждения на втором типе данных — ранжировании людьми разных ответов бота (Reward model). Это позволит нейронной сети корректно обрабатывать запросы различных категорий потребителей.

4. Использование reward model для дообучения языковой модели средствами обучения с подкреплением (Reinforcement learning). Данный этап позволит сохранить основные параметры, которые необходимы для разрешения конкретной задачи при дальнейшем использовании нейронной сети.

Схема обучения модели GigaChat представлена на рисунке 1.

В рамках применения ИИ в сфере обучения на начальном этапе предполагается использовать её как вспомогательную систему, предназначенную для облегчения поиска необходимого учебного материала.

Применительно к сфере образования, на этапе предобучения следует использовать учебные материалы, используемые в учебной программе. Это позволить сформировать информационную базу для дальнейшей работы.

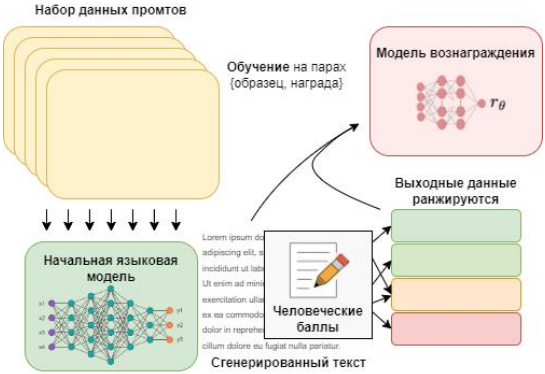


Рисунок 1 – Схема обучения модели GigaChat

На этапе дообучения нейросети предполагается использовать текстовые запросы с ожидаемым ответом. В качестве таких заданий будут выступать различные темы, рассматриваемые в учебном процессе, а в качестве ожидаемого ответа – содержание этих тем. Это позволить настроить нейросеть на корректную выборку из имеющихся у неё данных.

На этапе обучения с вознаграждением предполагается «бета-тестирование» нейросети на студентах с целью её настройки на «реальные» запросы.

Данная система позволит повысить уровень усвояемости материала студентами, а также позволит более эффективно его использовать при решении лабораторных и практических задач.

Обучение нейронной сети для решения задач в сфере образования сопряжено со множеством особенностей, которые вполне реально учесть, используя мультимодальную модель GigaChat, перспективы развития которой делают её привлекательной для интеграции в образовательный процесс.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Аймен Эль Амри. GPT-4. Руководство по использованию API Open AI / пер. с англ. В. С. Яценкова. – М.: ДМК Пресс, 2024. – 274 с.

2. Кублик С., Сабу Ш. GPT-3: Руководство по использованию API OpenAI / пер. с англ. В. С. Яценкова. – М.: ДМК Пресс, 2023. – 172 с.

3. Русскоязычная мультимодальная нейросеть от Сбера/ [Электронный ресурс] // Код Дурова: сайт. URL: https://kod.ru/gigachat-sber-neuro.

УДК 004.032.26

ЗАВАЛИШИН Г.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ПРИМЕНЕНИЕ МОДЕЛИ SAM ДЛЯ СЕГМЕНТИРОВАНИЯ ОБЪЕКТОВ

Segment Anything Model (SAM). Эта модель представляет собой значительный прорыв в области обнаружения контрастных объектов на изображениях и имеет широкий потенциал применения в различных сферах.

В современном мире компьютерное зрение играет ключевую роль в различных областях, от медицинской диагностики до автоматизации производственных процессов. Однако, одной из основных проблем, с которой сталкиваются алгоритмы компьютерного зрения, является обнаружение объектов на изображениях, особенно в условиях низкого контраста. Segment Anything Model представляет собой инновационный подход к решению этой проблемы.

Segment Anything Model основана на глубоком обучении и использует архитектуру свёрточной нейронной сети (CNN). Однако, в отличие от других моделей, SAM не требует явных аннотаций или масок для обучения. Она обучается с использованием обучения с подкреплением, где её целью является максимизация площади сегментированных объектов. Это позволяет модели изучать широкий спектр контрастных особенностей объектов на изображениях и делает её более универсальной и адаптивной.

Преимуществом SAM является её способность обнаруживать контрастные объекты даже в условиях низкого контраста или сложных фонов. Это делает её идеальным инструментом для различных приложений, таких как медицинская диагностика, промышленный контроль качества и автоматическое видеонаблюдение. SAM может использоваться для обнаружения дефектов на поверхности материалов, идентификации патологий на медицинских изображениях или даже для обнаружения движущихся объектов на видеозаписях.

Пример работы модели представлен на рис. 1

Рисунок 1 – Результат работы модели SAM

Ключевые особенности модели Segment Anything Model (SAM)

- Задача сегментации по подсказке: SAM был разработан с учетом задачи сегментации по подсказке, что позволяет ему генерировать правильные маски сегментации по любой подсказке, например, по пространственным или текстовым подсказкам, идентифицирующим объект.
- Продвинутая архитектура: В модели Segment Anything Model используется мощный кодер изображений, кодер подсказок и легкий декодер масок. Эта уникальная архитектура обеспечивает гибкие подсказки, вычисление масок в реальном времени и осознание неоднозначности в задачах сегментации.
- Набор данных SA-1B: Представленный проектом Segment Anything, набор данных SA-1B содержит более 1 миллиарда масок на 11 миллионах изображений. Являясь самым большим набором данных для сегментации на сегодняшний день, он обеспечивает SAM разнообразным и масштабным источником обучающих данных.
- Zero-Shot Performance: SAM демонстрирует выдающиеся показатели zero-shot в различных задачах сегментации, что делает его готовым к использованию инструментом для различных приложений с минимальной потребностью в оперативном проектировании.

Процесс обучения SAM начинается с изучения признаков на большом наборе изображений. Затем модель приступает к процессу сегментации, где она пытается разделить изображение на различные объекты и фоны. Обратная связь используется для корректировки сегментации в соответствии с целевой метрикой, такой как площадь сегментированных объектов. Это позволяет модели улучшать свою

точность по мере обучения и адаптироваться к различным условиям изображений.

У SAM существует три модели в зависимости от размера. Перечислю их в порядке убывания: Vit-H, Vit-L, Vit-B.

Также приведу сравнение работы самой маленькой модели SAM популярной моделью YOLO v8 в таблице 1.

Таблица 1 – Сравнение моделей

Модель	Размер	Параметры	Скорость
Meta's SAM-b	358 МБ	94.7 M	51096 ms/im
YOLO v8	6,7 МБ	3,4 M	59 ms/im

Это сравнение показывает разницу на порядок в размерах и скорости моделей между собой. В то время как SAM представляет уникальные возможности для автоматического сегментирования, он не является прямым конкурентом сегментным моделям YOLOv8 , которые меньше, быстрее и эффективнее.

Но для решения проблемы скорости может использоваться модель FastSAM. В ней используется только 2% параметров от оригинальной модели и скорость её работы увеличивается в 50 раз, что позволяется работать если не в режиме реального времени, то близкому к нему.

На рис. 2 приведены результаты сравнение режим работы двух данных моделей.

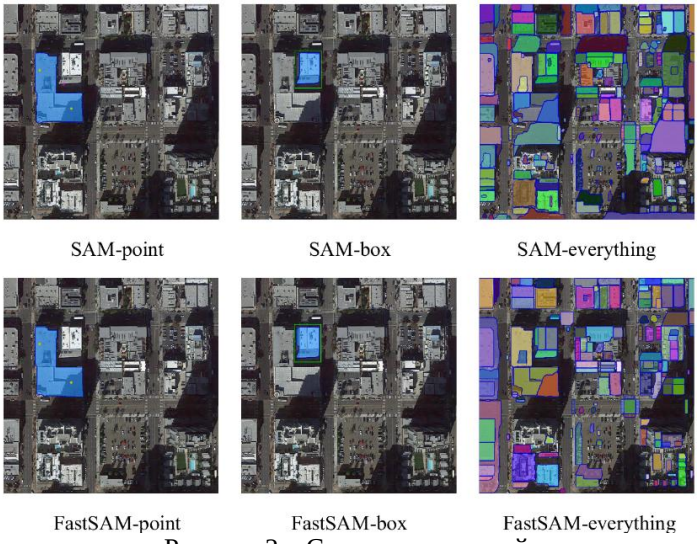


Рисунок 2 – Сравнение моделей

- Point. Выделение области происходит точкой, с помощью курсора мыши.
- Box. Выделяется область на картинке, в которой модели будет необходимо выделять сегменты.
- Everything. Модель выделяет сегменты на всём изображении целиком.

Segment Anything Model представляет собой значимый шаг в области обнаружения контрастных объектов на изображениях. Её способность работать без явных аннотаций делает её высоко перспективной для различных областей применения. Дальнейшие исследования и развитие этой модели могут привести к новым достижениям в области компьютерного зрения и обработки изображений.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Ultralytics YOLOv8 - новейшая версия модели обнаружения объектов и сегментации изображений в реальном времени [Электронный ресурс] – URL: https://docs.ultralytics.com/ru.

УДК 004.032.26

ЗАВАЛИШИН Г.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ПРИМЕНЕНИЕ НЕЙРОННЫХ СЕТЕЙ В СИСТЕМАХ ОБНАРУЖЕНИЯ И РАСПОЗНОВАНИЯ ОБЪЕКТОВ

Направление, связанное с использованием искусственных нейронных сетей (ИНС), в настоящее время является одним из ведущих направлений в развитии технологий компьютерного зрения.

ИНС все более широко применяются для решения задач обнаружения, сопровождения и распознавания объектов на отдельных изображениях и видеопоследовательностях.

Перечисленные задачи имеют место в системах обработки изображений специального назначения, разрабатываемых и производимых на производстве. Следовательно, вопрос о применении технологий ИНС с целью повышения эффективности таких систем на данный момент становится все более актуальным.

Рассмотрим текущее состояние основных аспектов при разработке ИНС.

1. Архитектуры нейронных сетей: Сверточные нейронные сети (CNN) стали краеугольным камнем систем обнаружения и распознавания объектов. Такие архитектуры, как Faster R-CNN, YOLO и SSD, достигли значительных успехов в точном обнаружении и распознавании объектов на изображениях и видео.

2. Доступность данных: Наличие крупномасштабных аннотированных наборов данных, таких как COCO (Common Objects in Context), ImageNet и Open Images и д.р, облегчило обучение глубоких нейронных сетей для задач обнаружения и распознавания объектов. Такое обилие данных способствовало повышению надежности и обобщающих возможностей этих систем.

3. Трансферное обучение: Методы трансферного обучения позволяют точно настраивать предварительно обученные модели нейронных сетей для конкретных задач обнаружения и распознавания объектов с ограниченными аннотированными данными. Этот подход стал важным в решении проблем нехватки данных и сокращении вычислительных ресурсов, необходимых для обучения новых моделей.

4. Производительность в реальном времени: Последние достижения в области аппаратного ускорения, в частности графических процессоров (GPU) и специализированных чипов, таких как TPU и FPGA, позволили использовать приложения для обнаружения и распознавания объектов в реальном времени, что делает их практичными для различных сценариев реального мира.

5. Применение в различных областях: Системы обнаружения и распознавания объектов на основе нейронных сетей применяются в различных областях, включая автономное вождение, безопасность и видеонаблюдение, медицинскую визуализацию, промышленную автоматизацию и дополненную реальность, демонстрируя их универсальность и эффективность.

Из этих аспектов следует следующие перспективы развития:

1. Улучшение точности: Дальнейший прогресс в разработке архитектур нейронных сетей и методов обучения направлен на повышение точности систем обнаружения и распознавания объектов, особенно в условиях ограниченной освещенности, размытости и заслонение объектов.

2. Снижение потребления ресурсов: Одним из важных направлений развития является уменьшение потребления ресурсов. Это включает в себя оптимизацию алгоритмов, аппаратное ускорение и разработку эффективных методов работы с данными.

3. Улучшение интерпретируемости: Для повышения доверия к нейросетевым алгоритмам, необходимо развивать методы, способные

объяснять принимаемые решения и их основания. Это позволит улучшить интерпретируемость и объяснимость работы нейросетевых моделей.

4. Применение в реальном мире: Развитие нейросетевых технологий в системах обнаружения и распознавания объектов будет продолжаться в направлении их внедрения в реальные условия эксплуатации, что потребует решения новых технических, организационных и правовых задач.

Но также присутствует некоторые проблемы при реализации систем обнаружения и распознавания объектов. Самые основные это точность распознавания, устойчивость к враждебным атакам, адаптация к предметно области, обработка больших объемов данных, доступность данных. Для решения данных проблем можн воспользоваться следующими методами:

1. Точность распознавания можно увеличить за счет адаптации модели к изменению освещенности, масштаба, ориентации и заслонения объекта. Для этого применяются методы расширения обучающих данных за счет трансформации, на этапе обучения модели. Лучшим вариантом адаптации модели является сбор и дообучение модели на тех данных, на которых модель показывает наименьшую точность.

2. Для устойчивости модели к враждебным атакам, необходимо использовать адаптивные метрики конечных прогнозов и применять методы обобщения по типам классов объектов.

3. Адаптация к предметной области улучшается за счет увеличения объема обучающих данных. Данные должны быть получены с разных источников, собранные в разное время года, в разное время суток, при различных атмосферных явлениях. Чем разнообразнее данные, тем выше степень адаптации модели к предметной области.

4. Для обработки больших объемов данных необходимо использовать современную специализированную технику и применять различные методы автоматизации сбора и обработки данных для обучения моделей нейронных сетей.

5. Для получения более широкого набора данных для различных предметных областей, необходимо использовать все доступные средства и методы сбора информации.

Было проведено несколько экспериментов обнаружения и распознавания.

Эксперимент 1. Оценка возможности обнаружения и распознавания наземных объектов в сложных условиях наблюдения с борта летательного аппарата с помощью дообученной модели ИНС.

Модель обнаружения и распознавания – YOLOv8. Объекты классификации – БТР («ARMORED») и грузовой автомобиль («KAMAZ»). Исходные данные – видеопоследовательность с обзорной системы (10000 кадров, размеченных вручную). Обучение – 7410 кадров, контроль – 1587 кадров.

Результат работы ИНС представлен на рисунке 1.

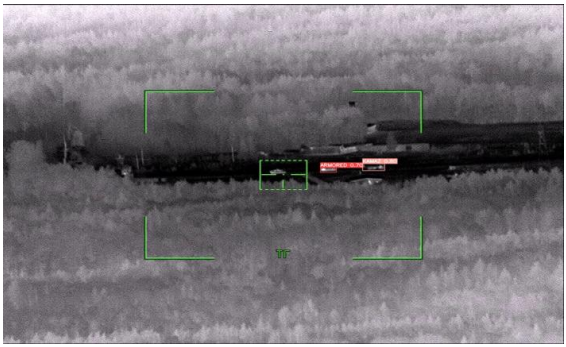


Рисунок 1 – Результат эксперимента 1

Эксперимент 2. Оценка возможности обнаружения и распознавания надводных объектов.

Модель обнаружения и распознавания - YOLOv8. Объекты классификации – надводные корабли («BOAT»). Исходные данные – видеопоследовательность с обзорной системы. Обучение – 35500 кадров, контроль – 7500 кадров. Результаты работы ИНС представлен на рисунках 2-3.

Рисунок 2 – Результат эксперимента 2

Рисунок 3 – Результат эксперимента 2

По результатам эксперимента можно сделать следующие выводы:

- Современные модели ИНС способны обнаруживать и распознавать объекты интереса в условиях, характерных для работы изделий специального назначения, разрабатываемых и производимых на производстве.
- Автоматическое обнаружение и распознавание объектов в сложной целевой обстановке на основе ИНС способно повысить эффективность и надежность систем обработки изображений, снизить нагрузку на операторов этих систем.
- Для обеспечения высоких показателей качества обнаружения и распознавания объектов необходимо в процессе обучения ИНС использовать достаточно большие обучающие выборки, отражающие многообразие возможных объектов, фонов и условий наблюдения. Обучение ИНС по большим выборкам изображений представляет собой итерационный процесс и требует большого объема вычислений.
- Обученные модели ИНС могут работать в реальном или близком к реальному масштабах времени на базе ограниченных вычислительных ресурсов: при этом могут использоваться уже существующие аппаратные платформы, ориентированные на параллельные вычисления.

Заключение

Современные исследования и разработки показывают, что технологии ИНС представляют собой значительный задел для повышения показателей качества обнаружения, отслеживания и распознавания объектов на видеоизображениях. В отличие от классических подходов, основанных, как правило, на подборе сложного комплекса эвристик, которые не всегда применимы в конкретных условиях, нейросетевые технологии базируются на использовании больших обучающих выборок и современных методов машинного обучения.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Ultralytics YOLOv8 - новейшая версия модели обнаружения объектов и сегментации изображений в реальном времени [Электронный ресурс] – URL: https://docs.ultralytics.com/ru.

УДК 004.4

ЗАМЕШАЕВ Д.В., СКВОРЦОВ С.В.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ГЕНЕРАЦИЯ ТЕСТОВЫХ ВАРИАНТОВ ДЛЯ СТРУКТУРНОГО ТЕСТИРОВАНИЯ ПРОГРАММНЫХ МОДУЛЕЙ

Предлагается подход к автоматизированному построению тестовых вариантов для проведения тестирования программных модулей методом белого ящика.

При разработке программных продуктов требуется решать задачи, связанные с их тестированием. Одним из видов тестирования является структурное тестирование или тестирование методом «белого ящика» [1]. Этот метод предполагает знание внутренней структуры тестируемой программы и гарантирует выполнение каждого оператора программы хотя бы один раз. Данный вид тестирования можно разбить на несколько основных этапов.

Этап 1. Формирование управляющего графа программы, определяющего внутреннюю структуру программного модуля. Управляющим графом называется ориентированный граф, показывающий ход потока управления внутри программы. Данный граф включает одну начальную вершину (начало программы), одну конечную вершину (конец программы) и множество промежуточных вершин, которые в свою очередь подразделяются на операторные вершины (соответствуют линейным участкам программы) и предикатные (соответствуют условным операторам). Дугами в управляющем графе обозначается передача управления. Алгоритм построения управляющего графа программы изложен в работах [1, 2].

Этап 2. Вычисление цикломатической сложности. Цикломатическая сложность – это метрика программного обеспечения, которая обеспечивает количественную оценку логической сложности программы [1] и определяет верхнюю оценку числа тестов, которые гарантируют выполнение всех операторов программы хотя бы по одному разу.

Этап 3. Формирование множества независимых путей. Под независимым понимается путь из начальной вершины управляющего графа в конечную, имеющий хотя бы одну дугу, которая отсутствует в других путях, уже входящих в это множество. Алгоритм формирования множества независимых путей можно найти в работах [3, 4].

Этап 4. Генерация тестовых вариантов на основе сформированного множества независимых путей. Под тестовым вариантом понимается пара вида <ИД, ОР>, где ИД – исходные данные, представляющие собой начальные значения значимых переменных, а ОР – ожидаемый результат, представляющий собой значения переменных по окончанию работы программы.

Генерация тестовых вариантов является достаточно непростой задачей с точки зрения автоматизации. Если начальные значения можно выбрать исходя из анализа строк кода, соответствующих предикатным вершинам, входящим в состав анализируемого независимого пути, то с ожидаемыми результатами ситуация обстоит несколько сложней. Чтобы получить ожидаемый результат требуется выполнить интерпретацию всех строк кода, входящих в состав независимого пути. Под интерпретацией здесь понимается моделирование исполнения операторов программы для выбранного варианта исходных данных. Очевидно, что поток управления будет соответствовать некоторому варианту независимого пути, определяемого исходными данными. При этом следует учитывать, что один тестовый вариант может «покрывать» сразу несколько независимых путей. Также необходимо отметить, что некоторые независимые пути не могут тестироваться самостоятельно, и должны тестироваться как части других путей [1].

В качестве примера рассмотрим формирование тестовых вариантов для программного модуля, текст которого показан на рис. 1, управляющий граф и таблица соответствия вершин операторам программы приведены на рис. 2. На основе полученного графа вычислена цикломатическая сложность, а также построены независимые пути для проведения структурного тестирования (рис. 3).

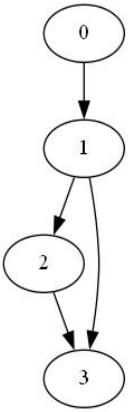
Учитывая, что тестовый вариант состоит из двух частей (исходных данных и ожидаемого результата), процесс его формирования также предлагается разделить на две части – формирование исходных данных и прогнозирование результата.

Для формирования исходных данных требуется проанализировать семантику предикатных вершин, которые входят в состав независимого пути, а также их номера, и номера вершин,

которые следуют за предикатными вершинами в независимом пути. Если номер предикатной вершины отличается от номера вершины, следующей за ней на один, то это означает, что в данном независимом пути условие предикатной вершины является истинным, в противном случае – ложным. Исключением является цикл с постусловием, где ситуация является обратной. Однако данное исключение не относится к языку Pascal ввиду того, что в нём тело цикла с постусловием выполняется пока заданное условие является ложным.

```
1  Program test_prog;
2  var
3    a: integer;
4  Begin {test_prog}
5    read(a);
6    if a < 10 then
7      a := a * 10;
8    print(a);
9  End.
```

Рисунок 1 – код тестируемой программы




```
{0}     |Program test_prog;
----------------------------
{0}     |var
----------------------------
{0}     |  a: integer;
----------------------------
{0}     |Begin {test_prog}
----------------------------
{0}     |  read(a);
----------------------------
{1}     |  if a < 10 then
----------------------------
{2}     |    a := a * 10;
----------------------------
{3}     |  print(a);
----------------------------
{3}     |End.
----------------------------
```

Рисунок 2 – управляющий граф исследуемой программы и соответствие вершин графа строкам программы

```
Независимые пути:
['0', '1', '3']
['0', '1', '2', '3']
Цикломатическая сложность : 2
```

Рисунок 3 –цикломатическая сложность и независимые пути

Если одна и та же предикатная вершина в пути встречается несколько раз, то в основном исследуется ситуация её первого появления в пути. В рассмотренном примере первый независимый путь содержит предикатную вершину с номером 1. Семантика данной вершины означает, что это простое условие if. Поскольку вершина, следующая за данной предикатной вершиной, имеет номер 3 (т.е. отличающийся более чем на 1), то делаем вывод, что условие if ложно, т.е. $a >= 10$.

Анализ второго независимого пути показывает, что предикатная вершина с номером 1 определяет простое условие, и номер следующей за ней вершины отличается на 1. Исходя из этого, делаем вывод, что условие истинно, т.е. $a < 10$.

На основании анализа независимых путей, можно сформировать следующие исходные данные для тестовых вариантов:

тестовый вариант 1, исходные данные: $a >= 10$;

тестовый вариант 2, исходные данные: $a < 10$.

Для получения ожидаемого результата требуется интерпретация программы с заданными исходными данными. Например, для первого тестового варианта можно задать $a = 11$, тогда, ожидаемый результат будет иметь вид: $a = 11$. Для второго тестового варианта можно задать $a = 9$, тогда ожидаемый результат будет иметь вид: $a = 90$ (условие истинно и переменная a будет умножена на 10, т.е. $a = 9 * 10 = 90$).

При выборе исходных данных дополнительно следует учитывать возможность их модификации. К примеру, если бы в приведённом примере перед условием if находился бы оператор «a := a – 2», то для первого тестового варианта следовало бы брать a равным не 11, а 13.

Таким образом, предложенный подход к генерации тестовых вариантов предполагает наличие интерпретатора, обеспечивающего моделирование исполнения программного модуля для заданных исходных данных. Его простейший вариант можно разработать самостоятельно, либо попробовать использовать уже существующий, например Pascal Script для языка Pascal.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Замешаев Д.В., Скворцов С.В. Автоматизация построения независимых путей и входных данных для решения задач структурного тестирования программ // Новые информационные технологии в научных исследованиях. – Рязань: ИП Коняхин А.В. (Boot Jet), 2023 – С. 7-8.

2. Замешаев Д.В., Скворцов С.В. Алгоритм построения управляющего графа программы для задач тестирования // Информационные технологии в прикладных исследованиях. – Рязань: ИП Коняхин А.В. (Boot Jet), 2023. – С. 87-90.

3. Орлов С.А., Цилькер Б.Я. Технологии разработки программного обеспечения. СПб.: Питер, 2021. 608 с

4. Рудаков В.Е., Скворцов С.В. Построение базового множества независимых путей потокового графа для тестирования программных модулей // Системы управления и информационные технологии. 2012. Т. 50. № 4. С. 67-70

УДК 004.89

КИРЬЯНОВ М.И.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ОПТИМИЗАЦИЯ ВЫБОРА АЛГОРИТМОВ КЛАСТЕРИЗАЦИИ ДЛЯ АНАЛИЗА ЧИСЛОВЫХ ДАННЫХ: СРАВНИТЕЛЬНОЕ ИССЛЕДОВАНИЕ

В данной статье проводится сравнительный анализ четырех популярных алгоритмов кластеризации: K-means, иерархической кластеризации, DBSCAN и спектральной кластеризации. Результаты исследования демонстрируют, что не существует универсального решения, и выбор алгоритма должен основываться на конкретных требованиях к задаче, размере и структуре данных.

В эпоху цифровизации и накопления огромных объемов данных, задача их структурирования и анализа становится все более актуальной. Одним из ключевых инструментов в арсенале аналитика данных является кластеризация — это фундаментальный метод в науке о данных, используемый для группировки схожих объектов на основе их характеристик. Этот метод обучения без учителя играет решающую роль в выявлении закономерностей, аномалий и структур в данных, что делает его незаменимым для исследовательского анализа данных, сегментации клиентов и обнаружения аномалий, а также для других задач. Эффективность кластеризации во многом зависит от используемого алгоритма, так как каждый из них имеет свои сильные и слабые стороны в зависимости от характера набора данных и рассматриваемой проблемы. От выбора алгоритма во многом зависит не только качество результата, но и скорость обработки данных, а также возможность интерпретации полученных кластеров. Таким образом, выбор правильного алгоритма кластеризации имеет первостепенное значение для получения значимых аналитических сведений.

Целью данной статьи является сравнение популярных алгоритмов кластеризации, теоретическая оценка их производительности по различным критериям для определения наилучшего варианта для кластеризации числовых данных.

Рассмотрим несколько основных методов кластеризации:

1. K-means – это один из самых популярных и легко понимаемых алгоритмов кластеризации. Он делит набор данных на k предварительно заданных непересекающихся подмножеств (кластеров), минимизируя сумму квадратов расстояний от каждой точки до центра ее кластера. Начальные центроиды выбираются

случайным образом, затем объекты распределяются по ближайшим кластерам. В дальнейшем центры кластеров пересчитываются на основе среднего значения объектов в каждом кластере, и процесс итеративно повторяется до сходимости.

2. Иерархическая кластеризация строит иерархическую структуру кластеров, называемую дендрограммой. Он начинается с каждого объекта, рассматриваемого как отдельный кластер, и затем объединяет ближайшие кластеры, пока все объекты не объединятся в один кластер. Методы иерархической кластеризации могут быть агломеративными (начинают с отдельных объектов и объединяют их) или дивизивными (начинают с одного кластера и разделяют его на более мелкие).

3. DBSCAN определяет кластеры на основе плотности распределения объектов в пространстве. Он идентифицирует плотные области, в которых объекты находятся близко друг к другу, и отделяет их от менее плотных областей. DBSCAN также может обнаруживать выбросы или шум в данных. Основными параметрами являются: минимальное число точек в окрестности для формирования кластера и радиус этой окрестности.

4. Спектральная кластеризация основана на теории графов. В отличие от традиционных алгоритмов, таких как k-means, которые работают напрямую с объектами в исходном признаковом пространстве, спектральная кластеризация сначала строит граф, где вершины соответствуют объектам, а ребра отражают их сходство. Затем, используя собственные значения (спектр) матрицы смежности или лапласиана графа, алгоритм выделяет кластеры, которые минимизируют разрезы в графе, тем самым группируя "похожие" объекты. Это позволяет ей обнаруживать кластеры сложной формы, которые могут быть недоступны для других алгоритмов.

Поиск наилучшего алгоритма кластеризации для числовых данных требует систематической оценки на основе набора критериев, имеющих решающее значение для объективного сравнения. Эти критерии включают в себя:

- Масштабируемость: способность алгоритма эффективно работать с большими объемами данных.
- Скорость: время, за которое алгоритм приходит к решению.
- Точность: эффективность алгоритма в правильной группировке похожих точек данных.
- Простота интерпретации: простота понимания и интерпретации полученных кластеров.

Масштабируемость

- *K-Means* известен своей масштабируемостью и эффективностью, особенно при обработке больших наборов данных. Его вычислительная сложность относительно невелика, что делает его оптимальным для решения масштабных задач кластеризации. Однако его производительность может ухудшиться при чрезвычайно большом количестве измерений из-за проклятия размерности.
- *Иерархическая кластеризация* сталкивается с трудностями масштабируемости; ее требования к вычислительной мощности и памяти значительно возрастают с увеличением размера набора данных, что делает ее менее подходящей для больших наборов данных. Необходимость вычисления и хранения матрицы расстояний между всеми парами точек становится узким местом.
- *DBSCAN* обладает умеренной масштабируемостью. Он хорошо работает с наборами данных среднего размера, но может столкнуться с проблемами с очень большими наборами данных, в первую очередь из-за вычислений на основе плотности. Методы оптимизации и индексации могут в некоторой степени устранить эти проблемы.
- *Спектральная кластеризация*, хотя и эффективна для идентификации сложных структур, страдает от масштабируемости. Разложение матрицы подобия на собственные значения требует больших вычислительных затрат, что ограничивает ее практичность для больших наборов данных.

Скорость

- На практике *K-Means* может похвастаться высокой скоростью сходимости, что объясняется его простотой. Однако его производительность чувствительна к первоначальному размещению центроидов, что может привести к изменению времени сходимости.
- *Иерархическая кластеризация*, особенно агломеративный подход, по своей природе медленна из-за последовательного процесса слияния и вычисления матрицы расстояний, что делает ее менее подходящей для приложений, чувствительных ко времени.
- Скорость работы *DBSCAN* сильно зависит от плотности распределения данных в наборе. Его производительность можно оптимизировать, используя методы пространственной индексации, которые помогают ускорить поиск соседних точек.
- Скорость *спектральной кластеризации* снижается из-за этапа декомпозиции по собственным значениям, что делает ее более медленной по сравнению с алгоритмами, подобными K-Means. Тем не менее, для небольших наборов данных или когда выявление сложных

кластерных структур имеет первостепенное значение, она остается жизнеспособным вариантом.

Точность

- *K-Means* обычно обеспечивают высокую точность для наборов данных с хорошо разделенными сферическими кластерами, но может плохо работать с невыпуклыми кластерами или кластерами разного размера и плотности.
- *Иерархическая кластеризация* отличается высокой точностью, когда набор данных имеет четкую иерархическую структуру. Её гибкость в создании кластеров с разным разрешением может стать существенным преимуществом.
- *DBSCAN* отличается высокой точностью кластеризации данных со сложной структурой, шумом и различной плотностью. Его способность выявлять выбросы повышает точность в реальных сценариях обработки данных.
- *Спектральная кластеризация* обеспечивает высокую точность обнаружения нелинейных границ кластеров, что делает ее превосходной для наборов данных, в которых кластеры переплетены между собой или их трудно разделить линейными границами.

Простота интерпретации

- Результаты *K-Means* относительно легко интерпретировать, что позволяет четко различать кластеры. Однако ограниченность алгоритма при обработке несферических кластеров иногда может приводить к ошибочным интерпретациям.
- *Иерархическая кластеризация* обеспечивает интуитивно понятные результаты, особенно при использовании дендрограмм, которые визуально отображают иерархию кластеров, помогая в интерпретации взаимосвязей данных.
- Интерпретация *DBSCAN* может быть неоднозначной из-за его подхода, основанного на плотности, что может привести к различному количеству кластеров. Тем не менее, его способность выявлять выбросы повышает глубину анализа данных.
- *Спектральная кластеризация* позволяет получить детальное представление о сложных структурах данных, но интерпретация ее результатов может быть сложной из-за абстрактного характера спектрального преобразования и кластеризации в ограниченном пространстве.

Чтобы проиллюстрировать практическое применение, рассмотрим пример с использованием числового набора данных. Этот набор представлен на рисунке 1 и включает две характеристики - координаты x и y.

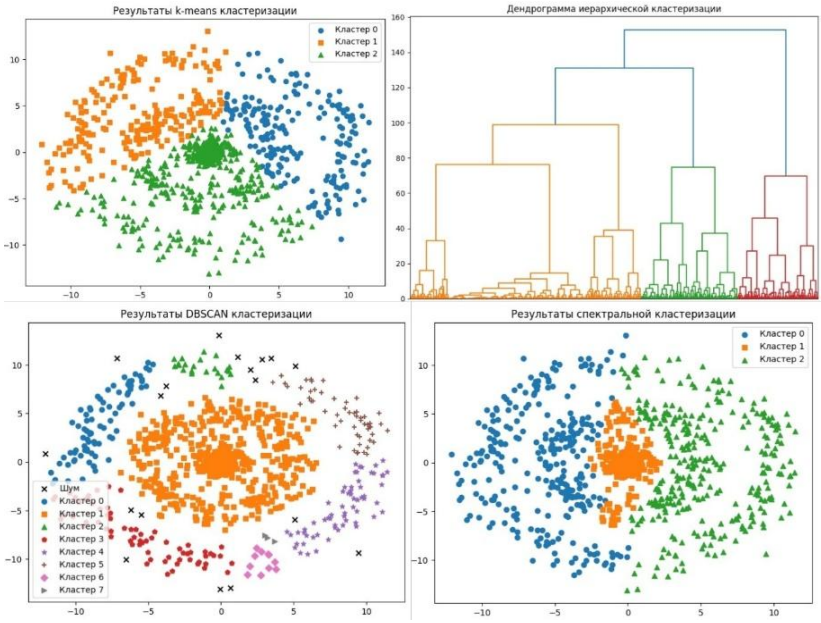


Рисунок 1 – Результаты кластеризации числовых данных

Результаты кластеризации различными методами демонстрируют следующее:

K-Means кластеризация основана на минимизации дисперсии внутри каждого кластера, что приводит к относительно компактным и сферическим кластерам, но могут не отражать истинную структуру более сложных распределений данных

Результаты кластеризации *DBSCAN* подчеркивают способность алгоритма обнаруживать кластеры на основе плотности данных, эффективно идентифицируя отдельные группы и классифицируя разреженные точки как шум. Этот метод не ограничивает формы кластеров сферическими, как это делает K-средние, что позволяет более естественно фиксировать структуру данных.

Результаты *иерархической кластеризации* показывают четкое разделение данных на три кластера. Подобно методу K-средних, этот метод позволяет идентифицировать компактные группы, но иерархический характер алгоритма позволяет получить более детальное представление о том, как эти кластеры связаны в разных масштабах. Иерархическая кластеризация хороша для данных, структура кластеров которых неизвестна заранее, позволяя аналитику определить оптимальное число кластеров.

Спектральная кластеризация показала хорошие результаты на данных, где кластеры не обязательно хорошо разделяются линейными границами. Рассматривая взаимосвязи между точками данных, спектральная кластеризация эффективно группирует данные в когерентные кластеры, которые соответствуют базовой структуре набора данных

На основе проведенного сравнения мы можем сделать вывод о том, что не существует универсального алгоритма кластеризации, оптимально подходящего для всех типов задач и данных. K-Means выделяется своей скоростью и масштабируемостью, что делает его подходящим для больших, хорошо разделенных наборов данных. Иерархическая кластеризация обеспечивает глубокое понимание структуры данных, что идеально подходит для наборов данных, в которых интересна иерархическая связь между точками. DBSCAN превосходно справляется с обработкой шума и идентификацией сложных шаблонов, подходит для наборов данных с неправильной формой и плотностью. Спектральная кластеризация, несмотря на свои вычислительные требования, не имеет себе равных в обнаружении сложных кластерных структур.

Выбор наилучшего алгоритма кластеризации для числовых данных зависит от нескольких факторов, включая размер набора данных, требуемую скорость обработки, форму и плотность кластеров. Для больших наборов данных с хорошо разделенными кластерами простой формы K-Means может быть наилучшим выбором. Если структура данных сложнее и предполагается наличие шума, стоит рассмотреть DBSCAN или спектральную кластеризацию.

В заключение, каждый алгоритм кластеризации имеет свои сильные и слабые стороны, и выбор наилучшего должен базироваться на конкретных требованиях к задаче, размере и структуре данных, а также на ресурсах, доступных для вычислений. Важно экспериментировать с разными алгоритмами и параметрами, чтобы найти оптимальное решение для вашей задачи. Кластеризация — это мощный инструмент в арсенале аналитика данных, и правильный выбор алгоритма может значительно повысить качество и ценность получаемых результатов.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Scikit-Learn: Clustering. [Электронный ресурс]. – URL: https://scikit-learn.org/stable/modules/clustering.html

2. Machine Learning: Clustering. [Электронный ресурс]. – URL: https://developers.google.com/machine-learning/clustering

УДК 004.032.26

КОМАРОВА М.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ЗАДАЧИ ДООПРЕДЕЛЕНИЯ ИДЕНТИФИКАТОРА ТРАНСПОРТНОГО СРЕДСТВА ПО НЕЧЕТКИМ ИСХОДНЫМ ДАННЫМ

В данной статье рассматривается математическая модель задачидоопределения идентификатора транспортного средства по нечетким исходным данным.

Задача доопределения – задача обработки исходных данных с неполной или нечеткой информацией о реальной предметной области, целью решения которой является выбор из множества альтернативных текущих состояний предметной области того, которое наиболее точно соответствует исходным данным.

Для реализации распознавания изображения необходимо использование метода Виолы-Джонса, основным преимуществом которого является скорость обработки изображения, и сверточной нейронной сети, использование которой обусловлено повышением устойчивости распознавания к различным искажениям символов в сравнении с классическими нейронными сетями и другими методами классификации изображений, а также снижением сложности нейронной сети и сложности ее обучения.

Рассмотрим математическую модель метода Виолы-Джонса, благодаря которому решается задача поиска объекта на изображении. Основу этого метода составляют три ключевые идеи: использование интегрального представления изображения по признакам Хаара, создание классификатора на основе адаптивного бустинга и метод комбинирования классификаторов в каскадную структуру.

Интегральное представление изображения — это матрица, размер которой соответствует размерам исходного изображения. В каждой ячейке матрицы хранится значение, равное сумме интенсивностей всех пикселей, находящихся выше и слева от данной ячейки, до границ прямоугольника, определяемого координатами текущей ячейки и левого верхнего угла (0,0). Для вычисления элементов матрицы L применяется формула (1):

$$L(x, y) = \sum_{i=0,\, j=0}^{i \le x,\, j \le y} I(i, j), \quad (1)$$

где I(i,j) – яркость пикселя исходного изображения.

При расчете значений элементов матрицы используется количество пикселей в исходном изображении, поэтому время, затрачиваемое на вычисление интегрального представления, пропорционально количеству пикселей. Благодаря этому интегральное представление изображения может быть рассчитано за один проход.

Для расчета элементов матрицы применяется формула (2):

$$L(x, y) = I(x, y) + L(x-1, y-1) + L(x, y-1) + L(x-1, y). \quad (2)$$

Используя интегральное представление изображения, можно быстро вычислить суммарную яркость произвольной прямоугольной области на изображении.

В методе Виолы-Джонса при обнаружении объекта применяется окно фиксированного размера, которое движется по изображению. Для каждой области, на которой находится окно, определяется признак Хаара, который используется для поиска объекта.

Для вычисления значения F признака Хаара используется формула (3):

$$F = X - Y, \quad (3)$$

где X – сумма значений яркостей точек, закрываемых светлой частью примитива, Y –сумма значений яркостей точек, закрываемых темной частью.

Однако, признаки Хаара недостаточно точны для классификации объектов. Для более точного определения объекта, признаки Хаара подвергаются обработке каскадным классификатором. Классификатор быстро отбрасывает окна, в которых не найден объект, и выдает значение «истина» или «ложь», указывающее на наличие или отсутствие объекта.

Классификатор строится на основе алгоритма бустинга для выбора наиболее подходящих признаков для искомого объекта на данной части изображения. Бустинг является последовательной процедурой построения композиции алгоритмов машинного обучения, каждый из которых компенсирует недостатки предыдущих, улучшая точность классификатора.

В результате работы алгоритма бустинга на каждой итерации формируется простой классификатор вида:

$$h_{j(z)} = \begin{cases} 1, \text{ если } p_j f_j(z) < p_j \theta_j, \\ 0, \text{иначе} \end{cases} \quad (4)$$

где p_i – направление знака неравенства, θ_j— значение порога, $f_{i(z)}$ – вычисленное значение признака, z – окно изображения.

После обнаружения объектов на изображении необходимо их распознать. Для этого подойдет использование сверточной нейронной сети. Рассмотрим ее математическую модель.

Сверточная нейронная сеть состоит из трех слоев: сверточный слой, субдискретизирующий слой и выходной слой. При рассмотрении математической модели каждого из слоев будем использовать следующие обозначения:

$l \in [1; L]$ - рассматриваемый в данный момент слой нейронной сети, где $L = 2a + 2, a \in Z^+$-количество слоев в сети;

N^l - количество карт признаков на слое *l*;

$f_l(.)$ - функция активации рассматриваемого слоя *l*;

y_n^l - *n*-я карта признаков на слое *l*.

1. Математическая модель сверточного слоя.

l– нечетное число, т.е. *l=1, 3, …, 2a+1.* Для карты признаков *n*:

- $w_{m,n}^l = \left\{ w_{m,n}^l (i, j) \right\}$ - свертка, которая применяется к карте признаков *m* слоя *(l-1)*, на слое *l* с картой признаков *n*;

- b_n^l - пороговые значения, которые присоединяются к карте признаков *n* на слое *l*;

- V_n^l - список всех уровней слоя *(l-1)*, соединяемые с картой признаков *n* слоя *l*.

Тогда карта признаков *n*сверточного слоя *l* будет вычисляться по формуле (5):

$$y_n^l = f_l(\sum_{m \in v_n^l} y_m^{l-1} \otimes w_{m,n}^l + b_n^l), \qquad (5)$$

где $\otimes$ математическая операция двумерной свертки.

Размер входных карт рассчитывается по формуле (6).

$$y_m^{l-1} = H^{l-1} \times W^{l-1} \qquad (6)$$

Размер применяемой к входным картам свертки рассчитывается по формуле (7).

$$w_{m,n}^l = r^l \times c^l \qquad (7)$$

Из данных формул следует, что размер выходной карты признаков рассчитывается по формуле (8).

$$y_n^l = (H^{l-1} - r^l + 1) \times (W^{l-1} - c^l + 1) \qquad (8)$$

2. Математическая модель субдискретизирующего слоя.

l– четное число, т.е. *l=2, 4, …, 2a.*Для карты признаков *n*:

- $w_{m,n}^{l}$ - фильтр, применяемый к *n* на слое *l*;

- b_n^l - добавочное пороговое значение.

Разделив карту признаков *n(l-1)*-го слоя на непересекающиеся блоки размером 2 на 2 пикселя и просуммировав значения 4-х пикселей в каждом блоке, получим матрицу $z_n^{l-1} = \left\{ z_n^{l-1}(i, j) \right\}$. Для расчета элемента матрицы применяется формула (9).

$$z_n^{l-1} = y_n^{l-1}(2i-1, 2j-1) + y_n^{l-1}(2i-1, 2j) + y_n^{l-1}(2i, 2j-1) + y_n^{l-1}(2i, 2j) \quad (9)$$

Для расчета карты признаков *n*субдискретизирующего слоя *l* применяется формула (10).

$$y_n^l = f_l(z_n^{l-1} \times w_{m,n}^l + b_n^l) \quad (10)$$

Размер карты признаков y_n^l субдискретизирующего слоя *l* рассчитывается по формуле (11).

$$H^l \times W^l = \frac{H^{l-1}}{2} \times \frac{W^{l-1}}{2} \quad (11)$$

3. Математическая модель выходного слоя.

L – выходной слой, который состоит из единичных нейронов. N^L - количество нейронов на данном слое. $w_{m,n}^L$ - фильтр, который применяется к карте признаков *m*последнего сверточного слоя, чтобы осуществить переход к нейрону *n* выходного слоя. b_n^L - пороговое значение, которое добавляется к нейрону *n*.

Выходной нейрон *n* рассчитывается по формуле (12).

$$y_n^L = f_L(\sum_{m=1}^{N^{L-1}} y_m^{L-1} m_{m,n}^L + b_n^L) \quad (12)$$

Выходом нейронной сети является вектор следующего вида: $y = [y_1^L, y_2^L, ..., y_{N^L}^L]$.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Гермиханова, Х. Р. Сверточные нейронные сети (некоторые аспекты) / Х. Р. Гермиханова // Наука молодых - будущее России: сборник научных статей 5-й Международной научной конференции перспективных разработок молодых ученых: в 4 т., Курск, 10–11 декабря 2020 года. – Курск: Юго-Западный государственный университет, 2020. – С. 21-23.

УДК 004.032.26

КОМАРОВА М.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ПРИНЦИП РАБОТЫ СВЕРТОЧНОЙ НЕЙРОННОЙ СЕТИ

В данной статье рассматривается сверточная нейронная сеть и ее отличия от классических нейронных сетей.

Классические нейронные сети - этотип модели искусственного интеллекта, состоящий из большого количества связанных нейронов, которые обрабатывают информацию и обучаются на основе входных данных. Классические нейронные сети состоят из нескольких слоев нейронов, каждый из которых передает сигналы следующему слою.

Помимо классических нейронных сетей естьсверточные нейронные сети (далее - СНС). Оба этих вида являются основными типами нейронных сетей, применяемых в области машинного обучения. Они имеют ряд ключевых различий, которые определяют их применение и эффективность в различных задачах:

1. Архитектура: Классические нейронные сети являются полносвязными, то есть каждый нейрон в одном слое связан со всеми нейронами в следующем слое. СНС имеют специальные слои, которые помогают извлекать и агрегировать признаки изображения.

2. Обработка изображений: СНС, благодаря своей архитектуре и специальным слоям, хорошо подходят для обработки изображений. Они могут автоматически извлекать признаки из входного изображения и обучаться на основе этих признаков.

3. Параметры: СНС обычно содержат меньшее количество параметров, чем классические нейронные сети, что делает их более эффективными для обучения на больших объемах данных.

4. Применение: Классические нейронные сети чаще применяются в задачах, где важно учитывать взаимосвязи между всеми признаками, например, в задачах обработки естественного языка. СНС лучше подходят для задач компьютерного зрения и распознавания образов.

СНС специализируются на обработке изображений и видео, и лучше всего работают с локальным контекстом, когда информация находится в близком пространственном расположении.

Например, такие нейросети хорошо работают с пикселями, которые являются частями изображения и содержат визуальные данные, такие как яркость и цвет. Если СНС видит какой-либо предмет

в одном пикселе на фотографии или рисунке, то с большой вероятностью она сможет распознать этот предмет и в соседних пикселях.

СНС чаще всего используются для решения двух задач: распознавания и классификации.

Структура СНС напоминает воронку, начиная с общей картины и смещаясь в детали. Они состоят из нескольких слоев, и чем больше слоев, тем мощнее архитектура и лучше происходит обучение нейросети.

В состав СНС входит три основных вида слоев: сверточный, субдискретизирующий и выходной (полносвязный). Каждый слой СНС следует друг за другом (Рисунок 1).

К слоям двумерной размерности относятся верточный и субдискретизирующий слои, а выходной слой представляет собой вектор из пространства R^1. У двумерных слоев в СНС имеется несколько уровней, каждый из которых представляет собой двумерный массив. Выход каждого уровня называется картой признаков.

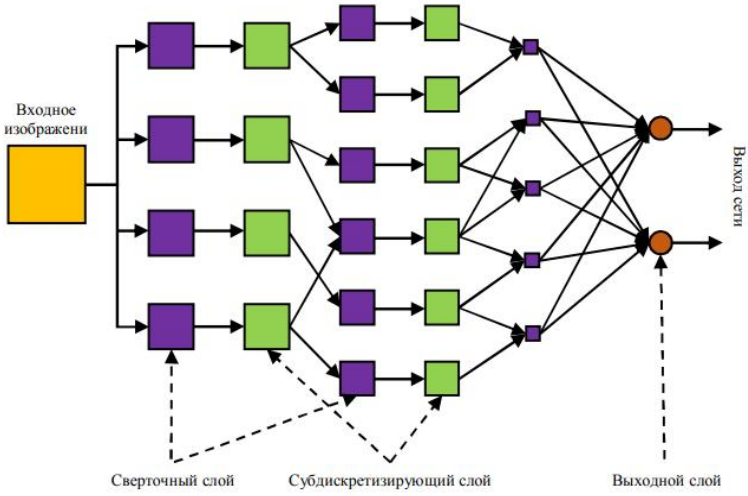


Рисунок 1 – Архитектура сверточной нейронной сети

Для того, чтобы применить математическую операцию свертки (фильтра) к изображению используется сверточный слой. Под сверткой понимается двумерная матрица коэффициентов. Фрагмент двумерного изображения является входом фильтра, а некоторое число – выходом (Рисунок 2).

Главным преимуществом применения данных фильтров заключается в том, что число на выходе тем больше, чем больше элемент изображения похож на применяемый к нему фильтр. Используя операцию свертки, возможно получить на выходе изображение, каждый пиксель которого будет соответствовать степени подобия кусочка изображения на фильтр. Таким образом, получается карта признаков.

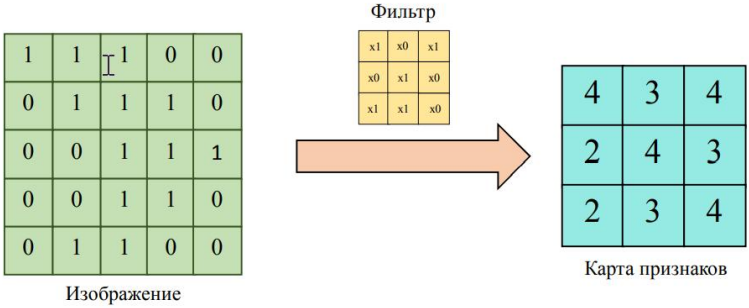


Рисунок 2 – Пример использования двумерной операции свертки для формирования

Для уменьшения размера изображения и увеличения степени инвариантности применяемых к нему фильтров используется субдискретизация. Так как наличие какого-либо признака на изображении гораздо важнее, чем точное знание его координат, то смыслом субдискретизации является выбор максимального нейрона из нескольких соседних.

Далее карта признаков уменьшается, и этот нейрон принимается за ее последующий элемент. С помощью этой операции увеличивается инвариантность к масштабу входного изображения.

Благодаря тому, что СНС строится посредством чередования слоев свертки и субдискретизации, появляется возможность строить карты признаков из карт признаков и распознавать иерархии признаков.

Концом работы СНС является преобразование карты признаков в вектор или скаляр, для чего и добавляется полносвязный слой на выходе.

Каждый нейрон в полносвязном слое является персептроном с нелинейной функцией активации. Для функции активации используется либо логистическая функция вида:

$$f(x) = \frac{1}{1+e^{-x}},$$

либо гиперболический тангенс:

$$f(x) = A \cdot \tanh(Bx), \text{ где A,B=const.}$$

Каждый нейрон, соединяясь с выходами нейронов предыдущего слоя, суммирует свои входы, умножает на веса, добавляет порог и результат подает на вход функции активации. Выходом нейрона является данное значение.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

2. Как работает сверточная нейронная сеть (CNN). [Электронный ресурс] – URL: https://neurohive.io/ru/osnovy-data-science/glubokaya-svertochnaja-nejronnaja-set/.

3. Наглядно о том, как работает свёрточная нейронная сеть [Электронный ресурс] – URL: https://habr.com/ru/companies/skillfactory/articles/565232/.

УДК 004.4

КОМЛЕВА Е.Р.

Рязанский государственный радиотехнический университет имени В. Ф. Уткина

АНАЛИЗ CASE-СРЕДСТВ ДЛЯ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

Рассматриваются различные CASE-средства проектирования баз данных, их анализ и сравнение.

CASE–технологии (Computer Aided Software Engineering) – это методология проектирования информационных систем (ИС) и набор инструментов, при помощи которых можно в наглядно смоделировать предметную область, а также проанализировать модель на разных этапах разработки и проектирования, а также разработать приложение с учетом потребностей пользователей.

Использование в учебном процессе CASE-технологий или CASE-средств для проектирования баз данных (БД) при изучении ряда курсов ИТ-направлений подготовки требует от образовательных организаций значительных вложений на приобретение специализированного программного обеспечения (ПО). Также известные IT-компании, продукты которых были широко

распространены в нашей стране, закрыли доступ к их использованию, и поиск альтернатив необходим в сложившейся ситуации. Одним из возможных решений проблемы может стать применение бесплатного (freeware), бесплатно распространяемого ПО или бюджетного программного продукта.

На сегодняшний день на рынке ПО представлено большое число коммерческих CASE-средств проектирования БД, различающихся по своим характеристикам. Сдерживающим фактором широкого использования в российских вузах CASE-технологий проектирования БД является высокая стоимость данного программного обеспечения.

Анализ CASE-средств

С учетом обозначенной цели настоящего исследования, в приведенный ниже обзоре включены бесплатные ПО для эффективной профессиональной работы по проектированию, разработке и управлению БД.

1. MySQL Workbench Community Edition (MWCE) – интегрированная среда для проектировщиков, разработчиков и администраторов БД, реализующая функции визуального проектирования, разработки и эксплуатации БД MySQL.

2. dbForge Studio for MySQL – профессиональный инструмент для разработки, администрирования и управления БД MySQL и Maria DB от компании Devart. С его помощью автоматизируются задачи проектирования, разработки и администрирования БД MySQL.

Отличительная особенность продукта – простота управления, наглядный, интуитивно понятный русскоязычный интерфейс, характеризующийся использованием визуальных конструкторов, мастеров, отсутствием англоязычной терминологии.

3. EMS SQL Management Lite Studio for MySQL – программный инструмент для разработки и администрирования БД MySQL.

Преимуществами данного программного продукта являются понятный русскоязычный интерфейс, а также наличие встроенной справочной системы на русском языке.

4. HeidiSQL – бесплатное ПО с открытым исходным кодом, для управления БД MySQL, Microsoft SQL Server, PostgreSQL.

Удобной для пользователей особенностью программы является наличие переносимой версии (portable application), не требующей для своего запуска процедуры установки.

5. SQLyog Community – инструмент для управления БД MySQL.

Следует отметить, что в версии Community отключен ряд базовых функций, в том числе: визуальный конструктор БД, мастер синхронизации схемы с БД на сервере, возможность импорта внешних

данных, построитель запросов, мастер создания расписания резервного копирования БД и ряд других инструментов.

6. Valentina Studio – инструмент управления БД от компании Paradigma Software, поддерживающий БД MySQL, Microsoft SQL, PostgreSQL, SQLite и собственную базу Valentina DB.

Преимуществом программы является возможность одновременного подключения к нескольким серверам различных СУБД.

7. ERwin – средство логического моделирования БД, использующее методологию IDEF1X. ERwin реализует проектирование схемы БД, генерацию её описания на языке целевой СУБД и реинжениринг существующей БД. ERwin выпускается в нескольких различных конфигурациях, ориентированных на наиболее распространенные средства разработки приложений 4GL. Версия ERwin /Open полностью совместима со средствами разработки приложений PowerBuilder и SQLWindows и позволяет экспортировать описание спроектированной БД непосредственно в репозитории данных средств.

Сравнительный анализ CASE-средств проектирования БД

С целью систематизации описания рассмотренных выше CASE-средств построим таблицу, отражающую возможность выполнения ими тех или иных функций. Оценка функциональных характеристик инструментальных средств производилась при непосредственной работе с программными продуктами (таблица 1).

Анализ функциональных возможностей CASE-средств проектирования и разработки БД позволил выделить минимальный базовый набор функций и инструментальных средств, которыми располагают практически все изученные продукты.

Программные среды, включающие только перечисленные функции, могут успешно использоваться для разработки простых БД, управления данными в таблицах БД, а также как средство администрирования сервера БД.

В ходе сравнения функциональных возможностей и инструментов для проектирования БД тройку лидеров заняли следующие программые продукты:

1. MySQLWorkbench.
2. dbForge Studio for MySQL
3. Valentina Studio

Проанализировав все имеющиеся программные продукты, можно прийти к следующему выводу:

У каждого средства есть свои преимущества и недостатки и

каждый из них является оптимальным решением в определенной области. Поэтому нельзя сказать о преимуществе определенного программного продукта.

Подводя итоги, CASE-средства являются важной составляющей современного цикла разработки программного обеспечение.

Современные CASE-инструменты охватывают обширную область поддержки многочисленных технологий проектирования ИС: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл ПО.

На сегодняшний день на рынке программных средств представлено большое число коммерческих CASE-систем проектирования БД, различающихся по характеристикам и стоимости. Замещение должно опираться на четкое представление о функциональных характеристиках и особенностях работы этих программ.

Таблица 1 – Функциональные характеристики CASE–средств проектирования БД

Продукт / Функция	MySQL Workbench	Valentina Studio	dbForge Studio for MySQL	SQLyog	EMS SQL Management Lite Studio for MySQL	HeidiSQL
Построение графической модели БД	+	+	+	–	–	–
Отображение модели в нотациях	+	–	–	–	–	–
Сохранение схемы БД	+	–	+	–	–	–
Создание БД на сервере из модели	+	+	+	–	–	–
Восстановление схемы БД из существующей на сервере БД	+	+	+	–	–	–
Синхронизация БД с моделью	+	+	+	–	–	–
Сохранение модели БД в виде DDL-скрипта	+	–	+	–	–	–
Импорт модели из DDL-скрипта	+	–	–	–	–	–
Возможность подключения к нескольким серверам БД	+	+	+	+	+	+
Визуальные инструменты создания и редактирования	+	+	+	+	+	+

объектов БД						
Визуальные инструменты Управления пользователями и привилегиями	+	–	+	+	+	+
Редактор SQL-кода	+	+	+	+	+	+
Средства экспорта/импорта данных, которые создают и загружают дампы БД	+	+	+	+	+	+
Средства экспорта/импорта внешних данных	+	–	+	–	–	–
Возможность сравнения БД	+	–	+	–	–	–
Возможность копирования БД с одного сервера на другой	–	–	–	+	–	–
Русскоязычный интерфейс	–	–	–	–	+	–
Русскоязычная встроенная справочная система	–	–	–	–	+	–

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Вендров, А.М. Case-технологии: Соврем. методы и средства проектирования информ. систем / А. М. Вендров. - Москва: Финансы и статистика, 1998. – 175,[1] с

2. Назарова, О.Б. Разработка реляционных баз данных с использованием CASE-средства All Fusion Data Modeler: учеб.- метод, пособие / О.Б. Назарова, О.Е. Масленникова. – 3-е изд., стер. – Москва: ФЛИНТА, 2019. – 73 с.

3. Хомоненко, А.Д. Базы данных: учебник для высших учебных заведений / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев; под ред. А. Д. Хомоненко. – 5-е изд., доп. - Москва: Бином-Пресс; Санкт-Петербург: Корона принт, 2006. – 736 с.

УДК 004.7

КОМЛЕВА Е.Р.
Рязанский государственный радиотехнический университет
имени В. Ф. Уткина

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА ДЛЯ МАРШРУТИЗАЦИИ В КОМПЬЮТЕРНОЙ СЕТИ

Рассматривается решение задачи маршрутизации с помощью генетического алгоритма.

В настоящее время практически в каждой компании присутствует компьютерная сеть. Это важное инфраструктурное средство, которое обеспечивает связь между компьютерами, серверами, принтерами и другими устройствами. Компьютерные сети позволяют эффективно организовывать рабочие процессы, обмениваться информацией, доступ к общим ресурсам и использовать различные программы и приложения. Наличие надежной и оптимизированной компьютерной сети является ключевым элементом успешной работы любой компании в настоящее время.

Для повышения эффективности и экономии ресурсов компании необходимо обратить внимание на оптимизацию маршрутов сети. Это позволит снизить нагрузку на сеть, улучшить скорость передачи данных и обеспечить более надежное подключение к серверам и приложениям. Оптимизация маршрутов сети также поможет улучшить общее качество обслуживания сотрудников и клиентов компании. Поэтому рекомендуется провести анализ текущих маршрутов сети и внедрить необходимые изменения для оптимизации их работы.

Для оптимизации процесса маршрутизации в компьютерной сети применяют генетический алгоритм. Этот метод основан на эвристическом алгоритме поиска, используемый для решения задач оптимизации и моделирования путём естественного отбора и естественной генетики.

Применение генетического алгоритма позволяет подобрать оптимальное распределение данных между узлами сети, учитывая такие факторы, как пропускная способность каналов, задержки, надежность связи и другие параметры. Генетический алгоритм также способен адаптироваться к изменяющимся условиям и требованиям, что обеспечивает более гибкую и эффективную маршрутизацию в сети.

Пример применения генетического алгоритма для нахождения оптимального маршрута в компьютерной сети.

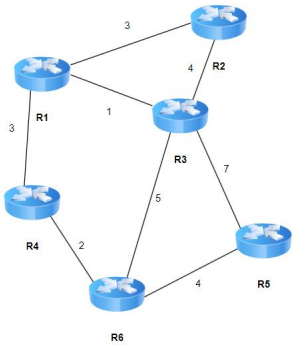


Рисунок 1 – Исходная модель

Часто для описания графа пользуются матрицей смежности, которая на пересечении строк и столбцов содержит значения весов дуг между соответствующими вершинами:

	1	2	3	4	5	6
1	0	3	1	3		
2	3	0	4			
3	1	4	0		7	5
4	3			0		2
5			7		0	4
6			5	2	4	0

Рисунок 2 – Представление маршрутов в виде графа

Те клетки, что не содержат значений, означают отсутствие связей. По идее, здесь можно прописать бесконечные величины, означающие бесконечно длинные маршруты.

Первым делом нам нужно формализовать задачу и определиться со способом хранения информации в хромосомах. Здесь могут быть самые разные варианты. Одна хромосома описывает все возможные маршруты от исходной точки (узла 1) до всех остальных вершин графа. Формируем начальную популяцию (случайные маршруты от вершины 1). Данный список представляет 6 списков со случайными неповторяющимися значениями.

[1, 4, 2, 6, 3, 5], [2, 1, 5, 4, 6, 3], [3, 5, 2, 6, 4, 1],
[1, 4, 6, 5, 3, 2], [4, 6, 1, 5, 2, 3], [4, 6, 1, 5, 2, 3]

Каждый маршрут представлен отдельным списком. Так как длина пути изначально неизвестна, то он заканчивается тогда, когда

встречается вершина с номером назначения. Например, первый список определяет маршрут из 1-й вершины в 1-ю и здесь первым стоит 0. Это как раз и есть кратчайший маршрут. Второй список – это маршрут из 1-й вершины во 2-ю. И здесь сразу стоит 1. Значит, кратчайший путь из 1 в 2 – это напрямую между этими вершинами. То же самое получается для 3-й и 4-й вершин. А вот до 5-й нужно идти через вершину 3. То же самое для 6-й, здесь нужно проходить через вершину 4. Это и есть кратчайшие маршруты и вот так следует интерпретировать значения генов в хромосоме.

Вычисляем приспособленность отдельных особей. Для этого с помощью алгоритма Дейкстры перебираем каждый список в хромосоме индивидуума, выделяя те узлы, которые относятся к текущему маршруту.

Способ скрещивания двух особей. Перебираем пары маршрутов первой и второй особи, далее осуществляется алгоритм упорядоченного скрещивания (неизменные индексы у списков).

Порядок мутации генов в хромосоме. Перебираем списки маршрутов и для каждого из них производим перемешивание индексов и вычисляем приспособленность особей.

Исходя из этого, можно выделить кратчайшие пути:

[1, 2, 3, 6, 5, 4] = 16,
[1, 2, 3, 5, 4, 6] = 14,
[1, 3, 5, 2, 4, 6] = 8,
[1, 3, 6, 5, 2, 4] = 10,
[1, 4, 6, 3, 5, 2] = 17,
[1, 4, 6, 5, 2, 3] = 9.

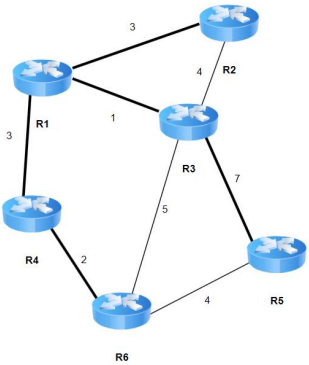


Рисунок 3 – Результат генетического алгоритма

На сегодняшний день генетические алгоритмы доказали свою конкурентоспособность при решении трудных задач, в том числе маршрутизации в компьютерных сетях. Генетические алгоритмы являются универсальным методом оптимизации многопараметрических функций, что позволяет решать широкий спектр задач. Генетические алгоритмы имеют множество модификаций и сильно зависят от параметров. Зачастую небольшое изменение одного из них может привести к неожиданному улучшению результата.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Батищев Д.И. Генетические алгоритмы решения экстремальных задач / Д.И. Батищев; Нижегородский госуниверситет. – Нижний Новгород: 1995.с. – 62с.

2. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы [Текст] / Под ред. В.М. Курейчика. – 2-е изд., испр. и доп. – М.: ФИЗМАТЛИТ, 2006. – 320 с.

3. Дарвин Ч. О происхождении видов путём естественного отбора или сохранении благоприятствуемых пород в борьбе за жизнь [Текст] / Ч. Дарвин. – М.: АН СССР, 1939. – Т.3.

УДК 004.41

КОНДРАШКИНА Е.С.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

ВИДЫ НЕЙРОННЫХ СЕТЕЙ И КОРРЕКТИРОВОЧНАЯ ФУНКЦИЯ

В статье рассматриваются нейронные сети, их виды, а также такой параметр как корректировочная функция и ее влияние на сети.

Введение

Нейронные сети – это компьютерные системы, работающие по принципу человеческого мозга, используя различные нейронные связи и их взаимодействие. Они используются в машинном обучении для обработки информации, извлечения закономерностей из данных и принятия решений на основе обучающих примеров [1].

Нейронные сети играют важную роль в машинном обучении. Она заключается в том, что они позволяют создавать модели, способные обучаться на больших объемах данных и делать прогнозы с

высокой точностью. Нейросети могут использоваться для решения разнообразных задач, таких как распознавание изображений, обработка естественного языка, анализ временных рядов, автоматическое управление и другие.

Благодаря своей способности к адаптации и обучению на данных, нейронные сети стали одним из основных инструментов в области искусственного интеллекта и машинного обучения, позволяя создавать сложные модели, которые могут эффективно решать различные задачи в различных областях.

Один из важных параметров, который влияет на обучение нейронных сетей, это корректировочная функция. Корректировочная функция, или функция потерь, является одним из ключевых параметров в обучении нейронных сетей и используется для оценки того, насколько хорошо модель предсказывает истинные значения на обучающем наборе данных. Целью корректировочной функции является минимизация ошибки между предсказанными и истинными значениями.

В зависимости от задачи, для которой обучается нейронная сеть, выбирается соответствующая корректировочная функция. Выбор правильной корректировочной функции важен, так как она влияет на способность модели к обучению и качество ее предсказаний. Подбор оптимальной функции потерь может помочь улучшить производительность нейронной сети и достичь лучших результатов на тестовых данных.

Виды нейронных сетей

Рассмотрим несколько видов нейронных сетей:

1. Самыми простыми являются однослойные нейронные сети или персептроны. Они состоят из одного слоя нейронов, где каждый нейрон связан с входом, а выход каждого нейрона вычисляется как взвешенная сумма входов, которая затем подается на функцию активации [2].

Входной слой данной системы принимает входные данные, скрытый слой состоит из одного слоя нейронов, где каждый нейрон связан со всеми входами, а выходной слой выдает результат работы сети.

Примером применения однослойной сети является бинарная классификация. В таком случае нейронная сеть может применяться для принятия решения является ли письмо спамом или нет. Другой областью применения является линейная регрессия. Например, для предсказания цены дома на основе его характеристик. А также

персептроны применяются в простых логических операциях таких как and, or, xor и другие.

Хотя однослойные нейронные сети имеют ограниченные возможности по сравнению с более сложными многослойными сетями, они все еще могут быть эффективны в решении определенных типов задач и могут служить хорошим стартовым пунктом для изучения глубокого обучения.

2. Многослойные нейронные сети (МНС) представляют собой более сложную архитектуру нейронных сетей, состоящую из нескольких слоев нейронов, включая скрытые слои между входным и выходным слоями [2]. Это позволяет МНС моделировать более сложные нелинейные зависимости в данных и улучшать способность сети к обобщению.

В отличии от однослойных нейронных сетей МПС могут содержать несколько скрытых слоев, где нейроны каждого слоя принимают входы от предыдущего слоя и передают выходы следующему.

Многослойные нейронные сети имеют следующие преимущества перед однослойными:

1. Способность к моделированию сложных нелинейных зависимостей в данных;

2. Улучшенная способность обобщения и обучения на разнообразных данных;

3. Повышенная эффективность и точность предсказаний благодаря более глубокой архитектуре.

Многослойные нейронные сети применяются для обработки изображений в сверточных сетях и для обработки естественного языка в рекуррентных сетях. МНС применяются также для построения персонализированных рекомендаций в интернет-магазинах, социальных сетях и стриминговых сервисах.

3. Еще одним видом нейронных сетей являются рекуррентные. RNN представляют собой класс нейронных сетей, способных работать с последовательными данными, такими как тексты, временные ряды или звуковые сигналы.

Основное отличие рекуррентных нейронных сетей от других типов сетей заключается в их способности запоминать информацию о предыдущих состояниях и использовать эту информацию при обработке последующих входных данных [2]. В скрытых слоях данной сети содержатся нейроны, которые имеют обратные связи и могут передавать информацию из предыдущих состояний.

Рекуррентные сети применяются в машинных переводах и анализе тональности текста, например для определения эмоциональной окраски текстовых данных. А также для генерации текстов, например создание текстов, стихов, статей, и распознания речи и интерпретации речевых сигналов.

Корректировочная функция

Параметр корректировочной функции, также известный как learning rate (скорость обучения), представляет собой числовое значение, которое определяет величину шага, с которым веса нейронной сети обновляются во время обучения. Роль параметра скорости обучения заключается в том, чтобы определить, насколько быстро или медленно веса сети должны изменяться в процессе обучения.

Корректировочная функция имеет важные аспекты параметра скорости обучения:

1. Скорость сходимости. Выбор оптимального значения скорости обучения позволяет сети сходиться к минимуму функции потерь быстро и без затухания или расхождения.

2. Избегание переобучения. Слишком большое значение скорости обучения может привести к переобучению модели, когда она неспособна обобщать на новые данные. Слишком маленькое значение может замедлить процесс обучения.

3. Регуляризация. Некоторые методы регуляризации, такие как L1 и L2 регуляризация, могут также влиять на выбор оптимального значения скорости обучения.

Выбор оптимального значения параметра скорости обучения является не простой задачей. Существует несколько подходов к ее решению. Приведем два основных:

1. Эмпирический подход. В данном подходе проводится несколько экспериментов с различными значениями скорости обучения и выбирается значение, при котором модель показывает лучшие результаты на валидационном наборе данных.

2. Адаптивная скорость обучения. Данные методы, например Adam или RMSprop, автоматически регулируют скорость обучения на основе градиентов и истории обновлений весов.

В случае, когда влияние параметра скорости обучения на процесс обучения имеет большое значение, это может привести к колебаниям и расходимости процесса обучения. В случае, когда влияние не велико, это приводит к медленной сходимости модели и увеличению времени обучения.

Виды коррекционных функций

Параметр скорости обучения является важным элементом при обучении нейронных сетей, поскольку он влияет на эффективность и стабильность процесса обучения [3]. Правильный выбор значения этого параметра помогает модели быстрее сходиться к оптимальным результатам и избегать проблем, связанных с переобучением или недообучением. Рассмотрим некоторые виды корректировочных функций:

1. Сигмоидная функция.

Это гладкая нелинейная функция, которая преобразует входной сигнал в диапазоне от 0 до 1. Она часто использовалась в скрытых слоях нейронных сетей, но из-за проблемы затухания градиента она часто заменяется на другие функции.

2. ReLU функция.

Это простая нелинейная функция, которая возвращает 0 для всех отрицательных значений и само значение для всех положительных значений. ReLU широко используется в сверточных нейронных сетях из-за своей простоты и хорошей производительности.

3. Гиперболический тангенс.

Это функция, которая преобразует входной сигнал в диапазоне от -1 до 1. Гиперболический тангенс часто используется в скрытых слоях нейронных сетей.

4. Softmax функция.

Это функция, которая преобразует вектор значений в вероятностное распределение. Softmax часто используется в выходном слое многоклассовой классификации, где каждый выход представляет вероятность принадлежности к определенному классу.

Каждая из этих корректировочных функций имеет свои особенности и применения в различных типах нейронных сетей. Выбор подходящей корректировочной функции зависит от конкретной задачи и требований модели.

Различные типы нейронных сетей, такие как сверточные, рекуррентные или глубокие нейронные сети, предназначены для различных типов задач и имеют свои особенности. Выбор правильного типа нейронной сети может значительно повлиять на качество прогнозов и скорость обучения, точно так же, как и правильно выбранная корректировочная функция.

В будущем нейронные сети будут продолжать развиваться и находить все более широкое применение в различных областях. С появлением новых технологий, увеличением вычислительной мощности и развитием методов обучения, нейронные сети станут еще

более мощным инструментом для решения сложных задач в машинном обучении, распознавании образов, обработке естественного языка, медицине, финансах и других областях.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Корячко В.П. Интеллектуальные системы и нечеткая логика [Текст]: учебник для магистров высших учебных заведений, обучающихся по направлению подготовки 2.09.04.01 "Информатика и вычислительная техника" (квалификация "магистр") / В.П. Корячко, М.А. Бакулева, В.И. Орешков. – Москва: КУРС, 2017. – 346 с.

2. Круглов В.В. Искусственные нейронные сети: Теория и практика / В.В. Круглов, В.В. Борисов. – 2. изд. – Москва: Горячая линия – Телеком, 2002. – 381с.

3. Николенко С. Глубокое обучение. Погружение в мир нейронных сетей / С. Николенко, А. Кадурин, Е. Архангельская. – Санкт-Петербург [и др.]: Питер, 2021. – 476 с.

УДК 004.41

КОНДРАШКИНА Е.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

РЕАЛИЗАЦИЯ НЕЙРОННОЙ СЕТИ НА PYTHON

В статье рассматриваются нейронные сети, программная реализация однослойной сети для логической операции XOR и роль нейронной сети в данной программе.

Введение

Нейронные сети – это мощный инструмент искусственного интеллекта, вдохновленный работой человеческого мозга. Они способны обучаться на основе большого объема данных, выявлять сложные закономерности и делать прогнозы. Нейронные сети используются в различных областях, таких как компьютерное зрение, обработка естественного языка, рекомендательные системы, медицинская диагностика и многое другое [1].

В современных технологиях нейронные сети играют ключевую роль в развитии различных приложений и сервисов. Они позволяют автоматизировать процессы, оптимизировать принятие решений, улучшать качество предсказаний и повышать эффективность работы систем. Благодаря своей способности к адаптации и обучению,

нейронные сети становятся неотъемлемой частью современного цифрового мира, привнося в него новые возможности и перспективы.

Описание программы на Python для решения задачи XOR с использованием нейронных сетей

Задача XOR (исключающее ИЛИ) является классическим примером задачи, которую невозможно решить линейными методами, но которую можно успешно решить с помощью нейронных сетей. В задаче XOR есть два бинарных входа (0 или 1) и один бинарный выход [2]. Результат вычисляется по следующему правилу: результат равен 1, если только один из входов равен 1, в противном случае результат равен 0.

Проблема заключается в том, что XOR не является линейно разделимой задачей, то есть нельзя провести прямую линию, которая разделит точки с результатами 0 и 1. Однако нейронные сети способны моделировать сложные нелинейные зависимости и идеально подходят для решения подобных задач.

Для решения задачи XOR с помощью нейронных сетей можно использовать простую архитектуру, состоящую из двух входных нейронов, одного скрытого слоя с нелинейной функцией активации (например, сигмоидальной) и одного выходного нейрона с пороговой функцией активации.

Ниже приведем этапы программы с некоторыми отрывками кода и пояснением:

1. Импорт библиотек
2. Определение функции активации - сигмоиды и ее производной:

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 – x)
```

3. Инициализация входных данных (X) и выходных данных (y) для задачи XOR:

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
Y = np.array([[0], [1], [1], [0]])
```

4. Инициализация весов нейронной сети:

```
np.random.seed(1)
input_layer_neurons = 2
hidden_layer_neurons = 2
output_layer_neurons = 1
```

```
weights_input_hidden =
np.random.uniform(size=(input_layer_neurons,
hidden_layer_neurons))
weights_hidden_output =
np.random.uniform(size=(hidden_layer_neurons,
output_layer_neurons))
```

5. Обучение нейронной сети с помощью обратного распространения ошибки на протяжении заданного количества эпох:

```
epochs = 10000

learning_rate = 0.1

for _ in range(epochs):
     #Прямое распространение
    hidden_layer_input = np.dot(X, weights_input_hidden)
    hidden_layer_output = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output,
weights_hidden_output)
    output = sigmoid(output_layer_input)
       #Обратное распространение
    error = y - output
    output_delta = error * sigmoid_derivative(output)
    hidden_error =
output_delta.dot(weights_hidden_output.T)
    hidden_delta = hidden_error *
sigmoid_derivative(hidden_layer_output)
       # Обновление весов
    weights_hidden_output +=
hidden_layer_output.T.dot(output_delta) * learning_rate
    weights_input_hidden += X.T.dot(hidden_delta) *
learning_rate
```

6. Тестирование нейронной сети на входных данных для проверки результатов.

На рисунке 1 приведен результат вышеописанной программы.

```
Результаты тестирования:
Входные данные: [0 0] - Прогноз: [0.25837323]
Входные данные: [0 1] - Прогноз: [0.69141587]
Входные данные: [1 0] - Прогноз: [0.69132956]
Входные данные: [1 1] - Прогноз: [0.39870372]
```

Рисунок 1 – Результат работы программы

Роль нейронной сети в программе

В приведенной программе, реализующей XOR с использованием нейронной сети, они играют ключевую роль в обучении компьютера, решая сложную задачу классификации данных. Сеть в данном случае состоит из нескольких слоев нейронов, где каждый нейрон принимает входные сигналы, обрабатывает их и передает выходной сигнал следующему слою.

Процесс обучения нейронной сети для данной задачи выглядит следующим образом. На вход подаются наборы обучающих данных (например, [0, 0], [0, 1], [1, 0], [1, 1]) и соответствующие им правильные ответы (0, 1, 1, 0). Нейронная сеть корректирует свои веса и параметры в процессе обучения таким образом, чтобы минимизировать ошибку между предсказанным и правильным ответом.

В результате обучения нейронная сеть находит оптимальные веса и параметры, которые позволяют ей правильно классифицировать входные данные. Это происходит благодаря ее способности извлекать сложные закономерности из данных и строить сложные модели для решения задач.

Таким образом, нейронные сети используются для обучения компьютеров решать сложные задачи путем адаптации своих параметров на основе обучающих данных. Их способность к извлечению признаков из данных и построению сложных моделей позволяет им эффективно решать разнообразные задачи, включая классификацию, регрессию, кластеризацию и многое другое.

Нейронные сети в логических операциях

Рассмотрим, как нейронные сети помогают в решении задачи XOR и других логических операций:

1. Обучение на данных.

Нейронные сети обучаются на обучающих данных, которые представляют собой входные значения и соответствующие им правильные ответы. Например, для задачи XOR обучающие данные будут состоять из входных значений [0, 0], [0, 1], [1, 0], [1, 1] и соответствующих им правильных ответов 0, 1, 1, 0.

2. Извлечение признаков.

Нейронные сети способны извлекать сложные признаки из данных, что позволяет им строить модели для классификации и предсказания. В случае задачи XOR нейронная сеть должна извлечь признаки, которые позволят ей разделить входные данные на два класса (0 и 1).

3. Построение сложных моделей.

Нейронные сети могут строить сложные модели, которые учитывают нелинейности и взаимодействия между признаками. Для задачи XOR нейронная сеть должна построить модель, которая способна правильно классифицировать входные данные, учитывая их нелинейную зависимость.

4. Коррекция весов.

В процессе обучения нейронная сеть корректирует свои веса и параметры таким образом, чтобы минимизировать ошибку между предсказанным и правильным ответом. Это позволяет ей находить оптимальные параметры для решения задачи XOR и других логических операций.

Обобщая все вышесказанное, заметим, что нейронные сети путем извлечения признаков из данных, построения сложных моделей и коррекции параметров в процессе обучения помогают в решении задачи XOR и других логических операций.

Блок-схема описанной выше программы

На рисунке 2 приведена блок-схема данной программы, а затем описание каждого блока.

Рисунок 2 – Блок-схема алгоритма

Описание блоков:

В первом блоке происходит инициализация параметров нейронной сети, таких как количество слоев, количество нейронов в каждом слое, функции активации и т. д.

Во втором блоке происходит обучение нейронной сети на обучающих данных для задачи XOR. Нейронная сеть корректирует свои веса и параметры на основе ошибки между предсказанным и правильным ответом.

В следующем блоке нейронная сеть использует обученные параметры для предсказания результатов на тестовых данных или новых входных значениях.

В последнем блоке выводятся результаты предсказания нейронной сети, например, правильные и предсказанные значения для задачи XOR.

Такая структура программы позволяет реализовать задачу XOR на нейронных сетях, пройдя через этапы инициализации, обучения, предсказания и вывода результатов.

Заключение

Нейронные сети являются мощным инструментом в области искусственного интеллекта и машинного обучения. Они нашли применение во многих различных областях, таких как компьютерное зрение, обработка естественного языка, медицина, финансы, робототехника и многое другое. Появление глубокого обучения и сверточных нейронных сетей привело к значительному улучшению результатов во многих задачах, превосходя человеческие возможности в некоторых случаях.

Значимость нейронных сетей заключается в их способности обучаться на больших объемах данных, выявлять сложные закономерности и делать точные предсказания. Они могут адаптироваться к новым условиям, улучшать свои результаты с каждым новым примером, что делает их очень эффективными инструментами для решения различных задач.

Таким образом, нейронные сети играют ключевую роль в современном мире и будут продолжать развиваться и применяться во многих областях, способствуя решению сложных задач и улучшению качества жизни людей.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Корячко В.П. Интеллектуальные системы и нечеткая логика: учебник для магистров высших учебных заведений, обучающихся по направлению подготовки 09.04.01 "Информатика и вычислительная

техника" (квалификация "магистр") / В.П. Корячко, М.А. Бакулева, В.И. Орешков. – Москва: КУРС, 2017. – 346 с.

2. Мюллер А. Введение в машинное обучение с помощью Python [Текст]: руководство для специалистов по работе с данными / Андреас Мюллер, Сара Гвидо; [перевод с английского и редакция А. В. Груздева]. – Москва [и др.]: Диалектика, 2017. – 472 с.

УДК 004.422.81

КОРОТКИХ И.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ИСПОЛЬЗОВАНИЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ ПРИ СОЗДАНИИ ИНФОРМАЦИОННЫХ СИСТЕМ ПРИЕМА И ОБРАБОТКИ ПЛАТЕЖЕЙ ЗА УСЛУГИ ЖКХ

Рассматривается основные вопросы, связанные с использованием различных архитектур при создании информационных систем по приему и обработки платежей за услуги ЖКХ, рассматриваются их приимущества и недостатки в этом контексте.

Стабильная работа систем приема и обработки платежей за услуги ЖКХ имеет ряд важных причин:

1. Удобство для пользователей: Жилищно-коммунальные услуги является жизненно и социально значимым. Их оплачивают каждый день десятки и сотни тысяч людей по всей стране. Они это делают регулярно. Повышение удобства оплаты, в связи с этим имеет большое значение, так как это экономит людям время.

2. Финансовая дисциплина: Регулярная и своевременная оплата услуг ЖКХ важна для поддержания финансовой устойчивости и обеспечения надлежащего уровня обслуживания жилых помещений и коммунальной инфраструктуры.

3. Предотвращение проблем с долгами: Задолженность по коммунальным услугам является серьезной проблемой для многих ресурсоснабжающих и управляющих компаний. Она приводит к сложностям в своевременном обслуживании инфраструктуры и выплате заработной платы людям, которые работают в этой сфере, снижают уровень качества поставляемых услуг.

Системы приема и обработки платежей за услуги ЖКХ являются весьма комплексными. Они выполняют большую работу по начислению платежей, составлению единого платежного документа, взаимодействию с государственной информационной системой ЖКХ и

так далее. Исходя из этого попытаемся понять, какая архитектура лучше подходит подобным системам.

Как уже было сказано, информационные системы по приему и обработке платежей за услуги ЖКХ весьма сложны. Вот несколько причин, почему микросервисы подходят для сложных систем [1]:

1. Масштабируемость: при использовании микросервисов есть возможность масштабировать сервисы как горизонтально, добавляя новые экземпляры сервиса, так и вертикально, увеличивая вычислительную мощность оборудования, на котором развернуто приложение.

2. Гибкость и независимость: Обновление приложение заключается в обновлении какого-то одного сервиса из множества, что позволяет обновлять приложение, при этом работоспособность не нарушается полностью.

3. Устойчивость к отказам: при использовании микросервисов отказ одного сервиса не приводит к полной остановке всей системы. Благодаря изоляции компонентов другие сервисы могут продолжать работу, минимизируя влияние отказа.

4. Распределенная разработка: Разработку приложения, которое пишется в микросервисном стиле, легко распараллелить, назначив на каждый сервис набольшую команду. Однако надо четко определить, каким образом микросервисы будут общаться между собой – «контракт» взаимодействия.

5. Технологическое разнообразие: Каждый микросервис может быть написан на разных технологиях, что позволяет выбирать наиболее подходящий инструмент для каждой конкретной задачи.

Приведем пример использования микросервисов в такой системе и попытаемся понять, почему использование монолитной архитектуры затруднительно в данной ситуации.

Существуют разные виды платежных систем. По способу выставления счета выделяют платежные системы с прямым выставлением счета и консолидирующие платежные системы.

Платежные системы с прямым выставлением счета (Direct Billing System или DBS) в контексте системы жилищно-коммунального хозяйства непосредственно рассчитывают сумму, которую надо выставить абоненту. Такие системы есть у каждой организации, которые выставляют счет. Они могут представлять собой систему на базе 1С либо иные, более специализированные системы, например, расчетный комплекс «Абонент+» [2].

С целью создания единого платежного документа необходимо консолидировать информацию о платежах из различных источников.

Этим занимаются консолидирующие платежные системы (ConsolidatingBilling Systems или CBS). Обычно для этого используются специфические решения под конкретную предметную область. В предметной области, связанной с обработкой платежей за услуги жилищно-коммунального хозяйства, можно использовать, например, платежную систему «Абонент+» [2].

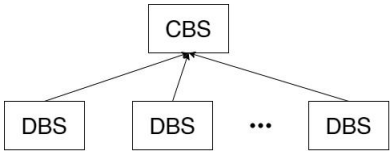


Рисунок 1 – Схема взаимодействия платежных систем с прямым выставлением счета и консолидирующих платежных систем

Расчетно-кассовый центр (РКЦ) — это специализированная организация, которое осуществляет обработку транзакций и проведение расчетов между клиентами РКЦ, банков и ресурсоснабжающими организациями. РКЦ выполняет ряд функций, связанных с обработкой платежей, переводов, инкассации и других операций. РКЦ принимают платежи у населения с помощью клиента платежной системы. Поскольку в РКЦ часто формируются очереди из людей, необходимо быстро загружать информацию об абонентах, чтобы быстрее их обслуживать. Достигается это использованием буферной БД, в которой данные хранятся в менее нормализованном виде. Рассмотрим взаимодействие систем на рисунке 2.

Как мы видим, наличие большого количества источников данных, необходимость в нескольких базах данных, необходимость в очередях приводит к тому, что необходимо разделить систему на несколько сервисов. Кроме того, отказ в одном из сервисов не приводит к отказу всей системы. Например, если происходит ошибка в сервисе, который обрабатывает пакеты приходящие из расчетных системы в платежную, не происходит к прекращению приема платежей у населения. Важность этого была подчеркнута в начале этой статьи. С помощью монолитной архитектуры добиться такого эффекта весьма затруднительно. Эффект от использования микросервисов в подобных системах оправдывает издержки на более сложный процесс разработки, поддержки и развертывания. Кроме того, системы такого масштаба разрабатывает большая команда разработчиков. Новых разработчиков проще интегрировать в рабочий процесс в случае, если используется микросервисная архитектура, так как работнику не нужно хорошо разбираться во всей системе, необходимо знать детали работы только сервиса, за который разработчик ответственен.

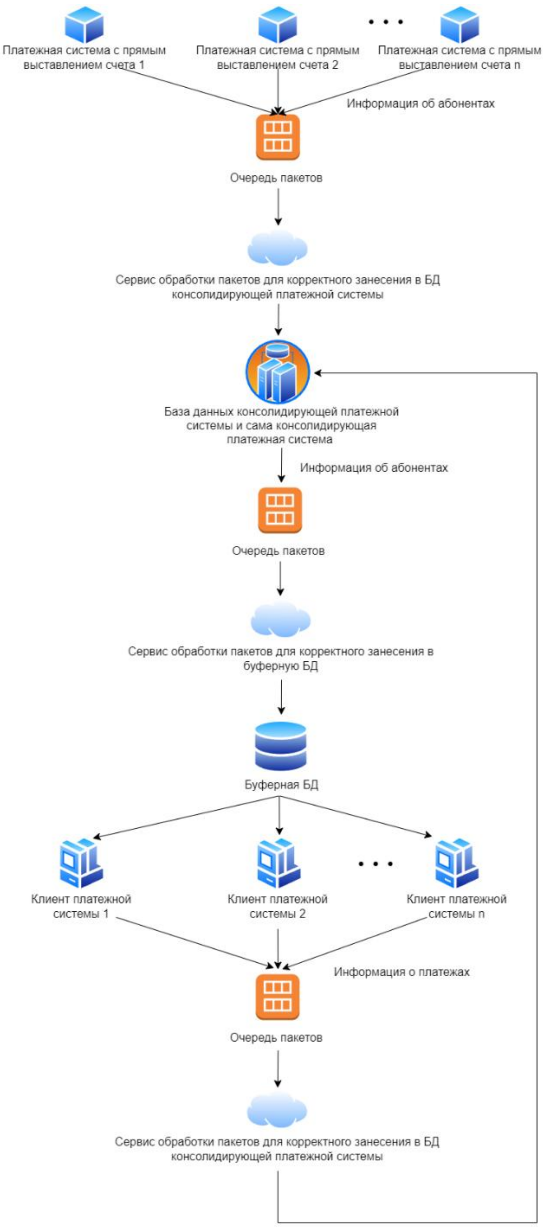


Рисунок 2 – Взаимодействие систем

Таким образом, использование микросервисной архитектуры при создании информационных систем по приему и обработки платежей за услуги ЖКХ является весьма оправданным.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. «Монолитная и микросервисная архитектура. Сравнение.» [Электронный ресурс]: Хабр. [Электронный ресурс]. – URL: https://habr.com/ru/companies/haulmont/articles/758780/.

2. ООО «Абонент+» [Электронный ресурс]: Абонент+ Расчетно-платежный комплекс. [Электронный ресурс]. – URL: https://abonent.plus/.

УДК 004.422.81

КОРОТКИХ И.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

СРАВНИТЕЛЬНЫЙ АНАЛИЗ МОНОЛИТНОЙ И МИКРОСЕРИВИСНОЙ АРХИТЕКТУР

Рассматриваются вопросы, связанные с особенностями одних из основных современных архитектур построения информационных систем. Приводится их сравнительный анализ.

Микросервисная архитектура – одна из ключевых архитектур при создании современных информационных систем. Она подразумевает разделение системы на ряд сервисов, обладающих высокой степенью независимости. Такая система является децентрализованной и отказ в одной ее части не приводит к полному прекращению работы всей информационной системы [1].

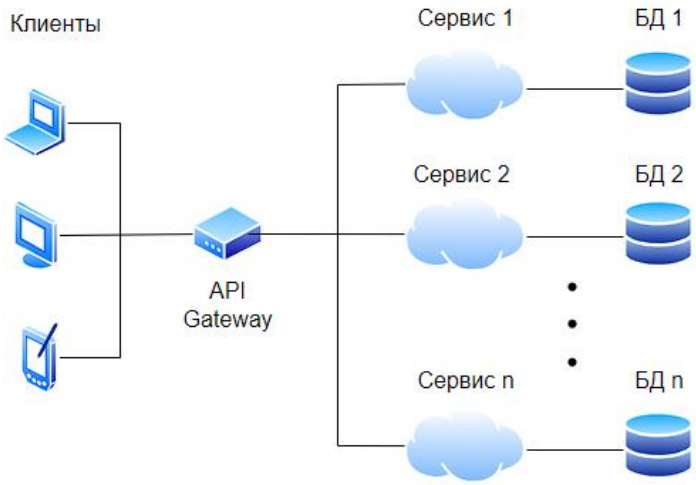


Рисунок 1 – Схема микросервисной архитектуры

Использование микросервисной архитектуры может принести следующие преимущества [1]:

1. Гибкость и масштабируемость: так как сервисы независимы друг от друга, их можно независимо тестировать и разворачивать, таким образом быстрее реагируя на изменения.

2. Отказоустойчивость: Сбой в одном сервисе не приводит к сбою всей системы; частично сохраняется работоспособность.

3. Легкость внедрения новых технологий: Каждый микросервис может быть реализован на своей собственной технологической базе.

4. Масштабируемость разработки: Разработку можно организовать параллельно, привлекая разных разработчиков для создания разных частей системы.

5. Поддерживаемость: Изменения в одном микросервисе не влияют на другие сервисы, что упрощает поддержку и обновление системы.

6. Безопасность: на каждом микросервисе можно организовать свою систему безопасности, делая вероятность взлома всей системы в целом более низкой.

При использовании микросервисной архитектуры возникают следующие проблемы [1]:

1. Сложность управления: Управление большим количеством сервисов может быть весьма затруднительно. Большое количество узлов порождает большое количество связей между ними, усложняет взаимодействие, делает систему в целом менее предсказуемой.

2. Сложность отладки: из-за высокой степени децентрализации локализовать ошибку может быть трудоемко.

3. Целостность данных: Децентрализация порождает проблему сложности поддержания целостности данных во всех микросервисах.

4. Тестирование: усложняется процесс тестирования и повышается ценность интеграционного тестирования, так как важно протестировать именно взаимодействие сервисов и работу всей системы в совокупности.

Монолитная архитектура – иной подход в разработки информационных систем. При таком подходе к разработке вся логика обработки данных инкапсулируется в едином приложении, что порождает как ряд преимуществ, так и ряд недостатков [2].

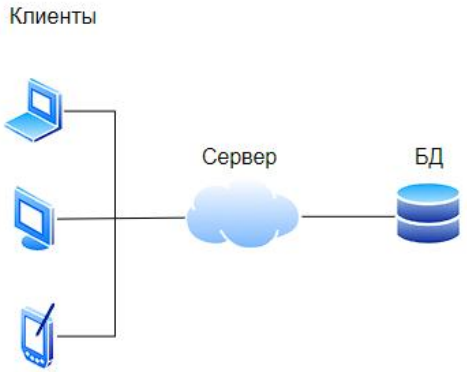


Рисунок 2 – Схема монолитной архитектуры

В использовании монолитной архитектуры есть несколько преимуществ:

1. Развертывание и обслуживание: так как приложение едино, не нужно тратить много усилий на развертывание проекта. Кроме того, инфраструктура не такая сложная и на ее поддержку так же не тратится много усилий.

2. Простота отладки и тестирования: из-за того, что весь код находится в одном месте, отладка и тестирование приложения проще.

3. Большая скорость разработки: построение приложения может быть быстрее за счет единой кодовой базы, однако разработку сложно масштабировать.

4. Меньше сложностей с сетевым взаимодействием: поскольку все компоненты приложения работают в пределах одного процесса, нет необходимости в сетевом взаимодействии между сервисами.

5. Единый стек технологий: разработка на базе единой технологической базы может упростить написание приложения, однако уменьшает гибкость разработки, так как разные технологии заточены под разные задачи, и реализация всех аспектов предметной области в рамках одной технологии может быть неэффективно.

При использовании монолитной архитектуры могут возникнуть следующие проблемы:

1. Масштабирование: в случае увеличения нагрузки на приложение, основной способ справиться с ней, увеличивать производительность электронно-вычислительных машин, на которых работает это приложение, что далеко не всегда удобно и реализуемо, а кроме того, недешево.

2. Снижение гибкости в использовании технологий: в монолитном приложении все компоненты связаны друг с другом, поэтому замена или обновление одного компонента может потребовать изменений во всей системе.

3. Сложность поддержки и разработки: так как приложение, написанное в монолитном стиле, сильно централизовано, добавление функционала со временем будет становиться все сложнее. Рефакторинг приложения так же будет непрост из-за сильной связанности приложения, наличия большого количества глобальных состояний в приложении. Решающим фактором становится квалификация разработчиков, их способность поддерживать чистоту кодовой базы, использование принципов SOLID, DRY, KISS, YAGNIи других.

4.Медленная скорость развертывания: при обновлении монолитного приложения может потребоваться перезапуск всего приложения, что может привести к временным простоям.

5. Сильная централизация: в результате возникновения ошибки в одном компоненте, приложение может прекратить работу полностью, что критично в случае, если приложение затрагивает большое количество клиентов или является социально значимым.

Таблица 1 – Сравнительный анализ

	Монолитная архитектура	Микросервисы
Простота разработки	Проще	Сложнее
Масштабируемость приложения	Небольшие возможности для масштабирования	Большие возможности для масштабирования

Отказоустойчивость	В случае возникновения проблемы перестает работать все приложение. Невысокая отказоустойчивость.	В случае возникновения проблемы перестает работать только затронутый сервис. Более высокая отказоустойчивость.
Скорость разработки	Более высокая	Более низкая
Сложность	Низкая	Высокая
Развертывание и обслуживание	Проще развертывать и поддерживать	Сложнее развертывать и поддерживать
Масштабируемость команды	Команду сложнее масштабировать	Команду сложнее масштабировать

Таким образом все вышеперечисленные факторы делают монолитную архитектуру хорошим выбором если проектнебольшой или средний, где скорость разработки и обслуживания имеют большое значение. Однако, это не самый удачный выбор при работе над крупными и сложными проектами. Для них лучше применять микросервисную архитектуру.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. «Монолитная и микросервисная архитектура. Сравнение.» [Электронный ресурс]: Хабр. [Электронный ресурс]. – URL: https://habr.com/ru/companies/haulmont/articles/758780/.

2. «Microservicesvsmonolithicarchitecture» [Электронныйресурс]: [Электронный ресурс]. – URL: https://www.atlassian.com /ru/microservices/microservices-architecture/microservices-vs-monolith/.

УДК 004.021

КОСТИН В.Ю., СКВОРЦОВ С.В.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

КЛАССЫ СЛОЖНОСТИ АЛГОРИТМИЧЕСКИХ ЗАДАЧ

В статье рассматриваются классы сложности задач, приводятся примеры задач соответствующих классов, их общие особенности и алгоритмы, применяемых для решения таких задач.

В теории алгоритмов классами сложности называются множества вычислительных задач, сходных по сложности вычисления [1]. Двумя наиболее важными классами сложности являются классы P и NP.

К классу Ротносятся задачи, решение которых считается «быстрым», то есть время решения полиномиально зависит от размерности входных данных. Примерами таких задач являются: поиск кратчайшего пути; поиск минимального остовного дерева; задача календарного планирования; сортировка конечного множества чисел; поиск в массиве; определение связности графов; умножение матриц и др.

Задачи классаP обладают следующими общими особенностями: для таких проблем существуют эффективные алгоритмы, которые работают с достаточно высокой скоростью даже на больших входных данных; точное решение может быть получено за относительно короткое время.

Теория классов сложности активно использует такую абстракцию как машина Тьюринга (МТ) – расширение конечного автомата. МТ оказалась очень удобным математическим аппаратом, позволяющим оценивать сложность задач.

Класс NP включает задачи, которые недетерминированная машина Тьюринга в состоянии решить за полиномиальное время [2]. Задачи в классе NP могут быть верифицированы на детерминированной машине Тьюринга за полиномиальное время. С точки зрения практики это означает, что проверка правильности решения задач NP класса может быть выполнена за полиномиальное время, но само нахождение оптимального решения может потребовать экспоненциального времени. Говоря в контексте О-нотации, задачи NPмогут обладать высокой асимптотической сложностью.

Определение класса NP с использованием МТ подходит и для класса P–очевидно, что упрощение задачи не приведет большим трудностям для вычислительной машины. Из этого следует, что класс NPвключает в себя P.

Примеры задачNP класса, не принадлежащих классу P:определение наличия гамильтонова цикла в неориентированном графе; задача коммивояжёра; задача выполнимости булевых формул (англ. SAT); задача о вершинном покрытии и др. Такие задачи имеют общие особенности: нет известного эффективного полиномиального алгоритма для их решения; экспоненциальный рост вычислительных затрат с увеличением размерности задачи; на практике чаще используются приближенные методы поиска решения.

NP-полные задачи. Среди всех задач класса NP можно выделить «самые сложные» — NP-полные задачи, или NPC (NP-complete), к которым можно свести любые задачи в пределах класса NP за полиномиальное время. Теория NP-полноты основана на строгих определениях из теории автоматов и формальных языков. В частности, теорема Кука – Левина утверждает, что задачаSAT выполнимости булевой формулы в конъюнктивной нормальной форме (КНФ) является NP-полной.

В доказательстве теоремы Кука каждая задача из класса NP в явном виде сводится к SAT.Доказательство этой теоремы, полученное Стивеном Куком в его фундаментальной работе [3] в 1971 году, стало одним из первых важнейших результатов теории NP-полных задач. Независимо в то же время эта теорема была доказана советским математиком Леонидом Левиным.

После появления результатов Кука была доказана NP-полнота для множества других задач. При этом чаще всего для доказательства NP-полноты некоторой задачи приводится полиномиальное сведение задачи SAT к данной задаче, возможно в несколько шагов, то есть с использованием нескольких промежуточных задач.

В качестве примеров NP-полных задач можно взять список из 21 NP-полных задач, опубликованный Ричардом Карпом в 1972 году в работе «Сводимость комбинаторных задач», в которой приведены как формулировки, так и доказательства NP-полноты для каждой: задача выполнимости булевых формул; бинарное целочисленное программирование; задача о клике; задача упаковки множества; задача о вершинном покрытии; задача о покрытии множества; задача о разрезающем циклы множестве вершин; задача о разрезающем циклы наборе дуг; задача ориентированного гамильтонова графа; задача неориентированного гамильтонова графа; задача выполнимости булевых формул с тремя литералами; задача раскраски графа; задача о кликовом покрытии; задача о точном покрытии; задача о вершинном покрытии в гиперграфах; задача Штейнера о минимальном дереве; задача о трёхмерном сочетании; задача о ранце; задачи из раздела теории расписаний; задача разбиения множества чисел.

Проблема равенства классов P и NP. Благодаря теореме Кука была сформулирована проблема равенства классов P и NP.Если для какой-то NP-полной задачи будет найден полиномиальный алгоритм решения, то и любую задачу из класса NP можно будет решить так же эффективно. В таком случае будет выполняться тождество P = NP.

Вопрос о равенстве этих двух классов считается одной из самых сложных открытых проблем в области теоретической информатики.

Математический институт Клэя включил эту проблему в список проблем тысячелетия, предложив награду размером в один миллион долларов США за её решение.

NP-трудные задачи. Говорят, что задача является NP-трудной (NP-сложной), если, подобно задаче выполнимостиSAT, она, по крайней мере, такая же сложная, как любая другая задача класса NP. При этом говорят, что задача является NP-полной, если она NP-трудная и также является членом класса NP.

Тем не менее, существуют задачи, которые кажутся NP-трудными, но при этом не являются членами класса NP. Такие задачи могут быть даже более сложными, чем NP-полные задачи. В качестве примера сложной задачи, не являющейся членом класса NP, можно привести игру в шахматы. Стратегия проигравшего описывается полным деревом всех его возможных ходов и лучших ответных ходов победителя. Количество узлов этого дерева будет экспоненциально зависеть от его высоты, равной количеству ходов, которые проигравший сможет сделать, перед тем как проиграть, применяя максимально эффективную защиту. Очевидно, что это дерево нельзя создать и проанализировать за полиномиальное время, поэтому задача не является членом класса NP.

Примером NP-трудной задачи может служить проблема остановки. Даны описание процедуры и её начальные входные данные. Требуется определить: завершится ли когда-либо выполнение процедуры с этими данными; либо, что процедура всё время будет работать без остановки. Алан Тьюринг доказал в 1936 году, что проблема остановки неразрешима на машине Тьюринга. Другими словами, не существует общего алгоритма решения этой проблемы.

Таким образом, NP-трудные задачи образуют множество наиболее сложных, актуальных проблем, в том числе неразрешимых.

Обобщая изложенный материал, можно визуализировать взаимоотношение между классами P, NP, NP-complete (NP-полными задачами), NP-hard (NP-трудными задачами) в случае, если P≠NP и если P=NP (рисунок 1).

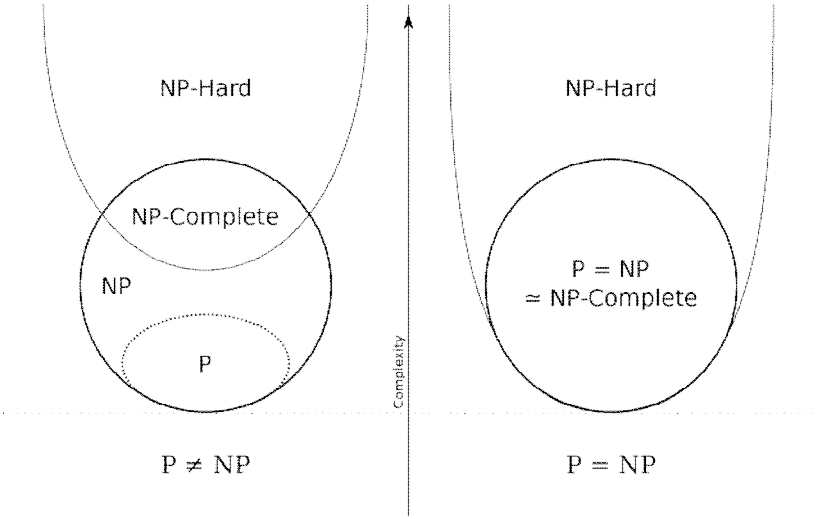


Рисунок 1 – Вычислительная сложность алгоритмических задач

В инженерной практике часто приходится сталкиваться со сложными задачами, которые оказываются NP-полными. Факт, что для решения задачи нет эффективного полиномиального алгоритма, не избавляет от необходимости решить проблему. Все, что известно, так это лишь невозможность создания программы для оптимального решения в наихудшем случае.

Однако для достижения цели остаются три варианта:

- алгоритмы, эффективные для средних случаев задачи. В качестве таких алгоритмов можно назвать алгоритмы поиска с возвратом, в которых выполняются значительные отсечения;

- эвристические алгоритмы, такие как имитация отжига, жадные алгоритмы, генетические алгоритмы, метод роя частиц, муравьиный алгоритм и др. Их можно использовать, чтобы быстро найти решение, но без гарантии, что это решение будет наилучшим;

- аппроксимирующие алгоритмы. Теория NP-полноты только оговаривает сложность для получения задачи. Но используя специализированные, ориентированные на конкретную задачу эвристические алгоритмы, скорее всего можно получить близкое к оптимальному решение для всех возможных экземпляров задачи.

Таким образом, практический интерес представляет изучение свойств различных алгоритмов на примере решения прикладных задач. Для этого целесообразно использовать практический метод оценки сложности исследуемых алгоритмов путем проведения

вычислительных экспериментов, направленных на оценку затрат машинных ресурсов. Получаемые экспериментальные данные позволяют выполнить сравнительный анализ алгоритмов и сформулировать рекомендации по их применению с учетом особенностей задачи и условий ее решения.

СПИСОКИСПОЛЬЗУЕМЫХИСТОЧНИКОВ

1. Крупский, В. Н. Теория алгоритмов. Введение в сложность вычислений: учебное пособие для вузов / В. Н. Крупский. – 2-е изд., испр. и доп. – Москва: Издательство Юрайт, 2024. – 117 с.

2. Kleinberg J. Algorithm Design /Kleinberg, Jon; Tardos, Éva- Addison-Wesley. – 2006. – С. 464.

3. Cook, Stephen. The Complexity of Theorem-Proving Procedures / Stephen, Cook // University of Toronto. – 1971. – С. 126-147.

4. Computational Complexity: A Modern Approach / Arora, Sanjeev; Barak, Boaz. – Cambridge. – 2009. – С. 534-560.

5. Johnson, David S. A Catalog of Complexity Classes. / David S, Johnson // Handbook of Theoretical Computer Science. – 1990 . – С. 67-161.

6. Turing A. On Computable Numbers, with an Application to the Entscheidungs problem / A. Turing // Proceedings of the London Mathematical Society. – London Mathematical Society. – Vol. s2-42, Iss. 1. – 1937. – С. 230-265.

УДК 004.021

КОСТИН В.Ю.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ ВЕТВЕЙ И ГРАНИЦ И ПОЛНОГО ПЕРЕБОРА ПРИМЕНИТЕЛЬНО К ЗАДАЧЕ КОММИВОЯЖЁРА

В статье рассматриваются особенности, преимущества и недостатки методов ветвей и границ и полного перебора, а также сравнительный анализ их эффективности на примере решения задачи коммивояжёра.

Задача коммивояжёра

Вход. Взвешенный граф G.

Задача. Найти цикл минимальной стоимости, проходящий через каждую вершину графа G ровно один раз (рисунок 1).

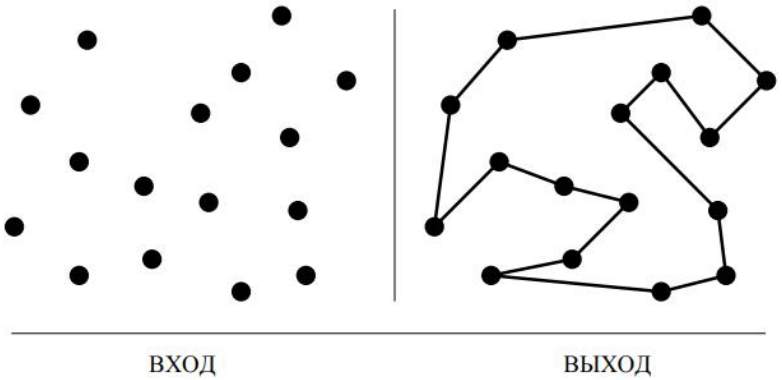


Рисунок 1 – Маршрут коммивояжёра

Задача коммивояжёра (Travelling salesman problem, сокращенно TSP) – наиболее известная изо всех NP-полных задач. В качестве причин такой известности можно назвать прикладную ценность задачи и легкость ее популярного изложения.

Неформальная формулировка - нахождение самого выгодного маршрута через указанные города хотя бы по одному разу с последующим возвратом в исходный город.

Существует несколько частных случаев TSP, большая часть которых принадлежит классу NP-трудных. На практике это означает, что полиномиальный алгоритм ее решения не найден, а рост вычислительных затрат связан экспоненциальной зависимостью с размерностью задачи. Уже при относительно небольшом количестве городов (более 66) она не может быть решена методом полного перебора никакими теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет. Хотя в реальных стандартных условиях использование метода полного перебора без модификаций становится нецелесообразным и при значительно меньшем количестве городов.

Метод полного перебора. В случае полного перебора TSP сводится к задаче поиска всех перестановок. Вычисление длины маршрута осуществляется пренебрежительно быстро по сравнению с генерацией всех перестановок, поэтому эффективность данного подхода будет определяться эффективностью алгоритма поиска перестановок.

Алгоритм на основе метода поиска с возвратом является самым неэффективным алгоритмом. Время работы такого алгоритма в худшем случае может заметно превышать величину n!.

Более эффективный алгоритм может быть получен на основе совместного использования методов декомпозиции и грубой силы. Идея заключается в следующем:

В соответствии с методом декомпозиции предположим, что задача меньшего размера (n - 1) успешно решена, т.е. все перестановки множества (n - 1) элементов сформированы (их число равно (n-1)!).

По методу грубой силы на основе каждой имеющейся перестановки из (n - 1) элементов получим n перестановок из n элементов путем вставки нового элемента в каждую из n позиций.

Все формируемые перестановки будут различны, а их общее число составит n(n-1)! = n!.

Такой подход легко реализовать в виде рекурсивного алгоритма, при этом возможны разные варианты, которые связаны со вставкой нового элемента в ранее сформированные перестановки. Например, элемент можно вставлять слева направо, справа налево, и иными способами. Это сильно влияет на эффективность такого алгоритма. В известных работах доказано, что наиболее эффективно с точки зрения временных затрат начинать вставку нового элемента справа налево и изменять направление каждый раз при переходе к новой, ранее сформированной перестановке.

Известен итерационный алгоритм (алгоритм Джонсона-Троттера), который формирует точно такую же последовательность перестановок, но работает заметно быстрее.

В данном алгоритме рекурсия заменяется итерацией, причем для этого с каждым элементом перестановки связывается не только его позиция, но и направление возможного перемещения.

Алгоритм Джонсона-Троттера является одним из наиболее быстрых алгоритмов, так как может быть реализован со временем, пропорциональным количеству перестановок: O(n!). Время работы данного алгоритма является постоянным. Данный алгоритм формирует не упорядоченную последовательность перестановок, однако в контексте задачи коммивояжёра это не имеет значения.

Метод ветвей и границ **(**Brach and bound, сокращенно BnB) – представляет собой одну из самых эффективных алгоритмических стратегий для нахождения точного решения задачи коммивояжера.

Метод ветвей и границ является модификацией метода полного перебора. Здесь строится дерево поиска решений, и оцениваются частично сформированные решения с точки зрения их корректности и

перспективности. Основное отличие метода ветвей и границ заключается в сущности задач, для решения которых он предназначен. Если метод полного перебора без модификаций больше ориентирован на решение комбинаторных задач, то метод ветвей и границ предназначен для решения оптимизационных задач, которой задача коммивояжёра и является. В таких задачах существует количественный критерий – целевая функция, что позволяет сравнивать решения. В задаче коммивояжёра такая оценка характеризует подмножество решений и определяет нижнюю или верхнюю границу целевой функции для данного подмножества решений.

Сравнительный анализ

Использование целевой функции в методе ветвей и границ позволяет более направленно стремиться к глобальному оптимуму по сравнению с полным перебором, но в худшем случае работа метода сводится к полному перебору вариантов решений. Более того, при недостаточном количестве опыта программирования и малой размерности задачи метод ветвей и границ иногда может показывать время работы хуже, чем метод полного перебора. Это обусловлено большой разницей в сложности разработки программы. Таким образом, в методе полного перебора минимизируются шансы утечки памяти, когда для метода ветвей и границ необходимо обладать продвинутыми навыками программирования. Однако при большой размерности задачи это преимущество метода полного перебора не имеет значения.

Также большую роль в методе ветвей и границ имеет то, насколько быстро в данном конкретном экземпляре задачи решение сходится к оптимуму. Как уже было сказано выше, в худшем случае BnB эффективен так же, как и полный перебор. В этом смысле метод очень непредсказуем, что является его недостатком. В среднем случае при хорошей реализации BnB относительно быстро сходится к оптимальному решению и заметно быстрее, чем в случае полного перебора.

Оптимизация реализации метода ветвей и границ в настоящее время является актуальной проблемой, а эффективности серьезных современных решений сравниваются между собой и обсуждаются в профессиональном научном сообществе. При умелом обращении такие алгоритмы могут решать задачи с тысячами вершин [3].

Экспериментальная часть. Для экспериментального сравнения эффективности методов ветвей и границ и полного перебора была разработана программа с таймером. Исходные данные для задачи были

получены с помощью генератора псевдослучайных чисел в диапазоне [1;99] в соответствии с распределением Гаусса. Время работы было рассчитано как среднее время между большим количеством экспериментов.

Результаты представлены в виде графика зависимости времени работы в наносекундах от количества городов (рисунок 2).

На графике можно наблюдать, что полный перебор эффективнее метода ветвей и границ при числе городов <10, но это обусловлено только отсутствием оптимизации используемой реализации метода ветвей и границ. Существующие программные решения лишены такого изъяна, что в контексте данного исследования позволяет воспринимать это как статистический выброс.

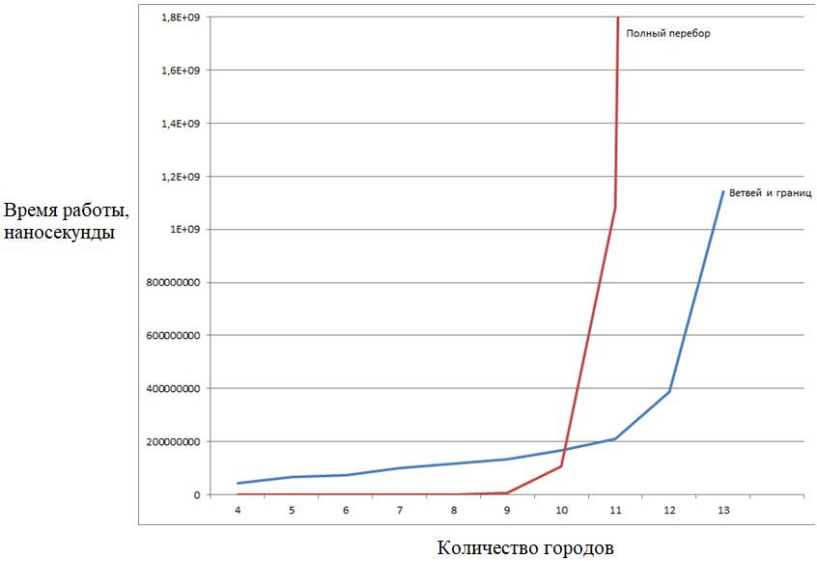


Рисунок 2 – Зависимость времени работы от размерности задачи для методов ветвей и границ и полного перебора

Возможности сравнительного анализа оказались сильно ограничены в силу того, что при уже 13 городах поиск решения методом полного перебора имеет значительные временные затраты. Однако это не мешает наблюдать, что метод ветвей и границ значительно эффективнее даже без оптимизации.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Коршунов Ю.М. Математические основы кибернетики: учеб. пособие. – М.: Энергия, 1980. – 424 с.

2. Крупский, В. Н. Теория алгоритмов. Введение в сложность вычислений: учебное пособие для вузов / В. Н. Крупский. – 2-е изд., испр. и доп. – Москва: Издательство Юрайт, 2024. – 117 с.

3. Steven S S. Skiena. The Algorithm Design Manual / Steven S S. Skiena. – Springer 2nd ed. – 2008. – 549 с.

УДК 004.94

КОШЕЛЕВ А.Д., КОШЕЛЕВА М.С., ОРЕШКОВ В.И.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ЭВОЛЮЦИЯ КОНЦЕПЦИИ ЦИФРОВЫХ ДВОЙНИКОВ

В данной статье рассматриваются этапы развития концепции цифровых двойников (ЦД). Дается определение когнитивного цифрового двойника, описываются его ключевые характеристики и основные отличия от ЦД.

Введение

Цифровой двойник (ЦД, Digital twin) – это виртуальная копия физического объекта, процесса или системы. Она создается на основе данных, полученных на стадиях разработки и изготовления продукта, или данных о текущем состоянии уже работающего процесса.

В научной литературе можно встретить различные объяснения термина, в том числе противоречащие друг другу. Это связано с тем, что понятие ЦД является достаточно широким и может трактоваться по-разному в зависимости от специфики деятельности компании или конкретного проекта.

В связи с этим возникла острая необходимость формирования подходов к интерпретации термина «цифровой двойник» с учетом следующих факторов:

− Стадии разработки. Разные определения подразумевают, что разработка цифрового двойника должна осуществляется до этапа создания физического объекта, либо до начала его эксплуатации, либо допускают оба подхода (классическая концепция, включающая в себя прототипы и экземпляры цифровых двойников).

− Адекватность модели. Необходимый уровень адекватности модели достигается до ввода реального объекта в эксплуатацию.

− Бизнес-цели. Различия в терминологии чаще всего связаны с преследуемыми целями применения данной концепции.

Кроме того, не существует универсальных методов построения цифровых двойников, подробных стандартов и типовых решений, но есть общие рекомендации:

Этап 1. Построение виртуальной модели продукта или бизнес-процесса, которая состоит из трех уровней:

– уровень компонентов — геометрическая и физическая модели объекта;

– уровень поведения — анализ продукта и взаимодействия с пользователем;

– уровень правил — оценка, оптимизация и прогностические модели, основанные на правилах взаимодействия с продуктом.

Этап 2. Обработка данных для облегчения принятия решений по проектированию (иногда при помощи искусственного интеллекта). На этом этапе данные, собранные из различных источников, подвергаются анализу, объединению и визуализации для более полного представления.

Этап 3. Моделирование поведения продукта в виртуальной среде с помощью технологий имитационного моделирования с последующей визуализацией в среде виртуальной реальности. Наглядность позволяет специалистам из разных областей знаний принимать эффективные решения в процессе коллективного обсуждения.

Этап 4. Управление продуктом или процессом для реализации рекомендуемого поведения. Датчики выполняют функцию по восприятию информации физического мира, а исполнительные механизмы позволяют вносить желаемые изменения, запрошенные цифровым двойником. Технологии дополненной реальности могут быть использованы для отображения некоторых частей виртуального продукта в физическом мире, например, для просмотра состояния отдельных параметров продукта в реальном времени.

Этап 5. Установка двустороннего и безопасного соединения между физическим и виртуальным продуктами в режиме реального времени. Для решения этой задачи используют облачные вычисления и хранение данных на удаленных серверах.

Этап 6. Сбор информации, связанной с продукцией или услугами, из различных источников (комментарии клиентов, записи о просмотре/загрузке и т.д.), сведения об окружающей среде, интерактивные данные о взаимодействии и т.п. Собранные материалы

отправляются на первый этап для внесения изменений в виртуальный продукт с целью его дальнейшего улучшения.

Цифровые двойники работают в «среде цифровых двойников» (Digital Twin Environment, DTE) – интегрированном пространстве, предназначенном для работы с виртуальными копиями в различных целях, например, для предсказания производительности физического объекта или запроса текущего состояния.

Концепция начала развиваться в производственной сфере. Она использовалась для имитации различных ситуаций, позволяющих в последствии подбирать наиболее адекватные сценарии проведения технических систем или машин в целях контроля качества и предотвращении сбоев.

Однако в последнее время, благодаря стремительному развитию искусственного интеллекта и Интернета вещей, цифровые двойники получили широкое распространение за пределами промышленности, например, в персонализации клиентов. Наметилась устойчивая тенденция использования, так называемых, когнитивных цифровых двойников (КЦД, Cognitive digital twins) – модифицированной версии ЦД.

В данной статье рассматривается эволюция цифровых двойников, а также новые направления развития данной концепции, в частности КЦД.

Эволюция цифровых двойников

Эволюцию цифровых двойников можно рассматривать с различных точек зрения, например, уровень взаимодействия прототипа и ЦД, функциональные возможности двойника или степень его автономности. Рассмотрим наиболее общие этапы развития концепции.

Предшественниками ЦД являются цифровая модель и цифровая тень:

– Цифровая модель – это виртуальное представление физического объекта, созданное в системе автоматизированного проектирования или моделирования, отображает геометрические, структурные и функциональные свойства прототипа, поддерживает только ручной обмен данными.

– Цифровая тень — это сохраненная копия данных о состоянии физического прототипа, с односторонним потоком данных между ним и цифровым аналогом.

Цифровые двойники были впервые представлены в начале 2000-х годов Майклом Гривзом в презентации курса по управлению жизненным циклом продукта. Однако первое описание его

использования было опубликовано несколько лет спустя НАСА в «Технологических дорожных картах», где ЦД использовался для проведения испытаний при подготовке к полету.

Изначально ЦД рассматривались с практической точки зрения, как

– Цифровой двойник объекта — на этапе проектирования продукта учитывается огромное количество его характеристик и ограничений, можно проводить виртуальные испытания, вносить необходимые изменения в конструкцию без изготовления реальных прототипов.

– Цифровой двойник процесса – моделирует реальные бизнес-процессы, разные режимы работы и конфигурации оборудования для оптимизации производства. Наглядно показывает актуальное состояние оборудования, а также позволяет накапливать и обрабатывать данные, поступающие с датчиков, для предсказания дальнейшего состояния процесса.

– Цифровой двойник систем – это виртуальная модель, описывающая работу отдельных производств или целых предприятий. Она включает большое количество данных, необходимых для имитации поведений сложной системы и управления ими. При этом цифровой двойник может быть построен как для уже существующего предприятия в целях оптимизации его бизнес-процессов, так и для проектирования новых.

После распространения концепции на обрабатывающую промышленность, примерно в 2012 году, появились следующие модифицированные версии ЦД:

– Цифровой двойник — прототип (Digital Twin Prototype, DTP). Содержит всю информацию об объекте, необходимую для его создания (начиная с этапа проектирования и заканчивая производством). Включает данные о требованиях к изделию, по эксплуатации и обслуживанию, 3D модели объектов, описание технологических процессов, условия утилизации, спецификации.

– Цифровой двойник — экземпляр (Digital Twin Instance, DTI). Это копия конкретного экземпляра продукта после его изготовления, с которым она будет связана на протяжении всего жизненного цикла изделия. Включает информацию о материалах и компонентах, рабочих процессах, результаты тестов, записи о проведенных ремонтных работах, операционные данные от датчиков, параметры мониторинга и многое другое. Экземпляр цифрового двойника обеспечивает возможность отслеживать и анализировать работу физического объекта в реальном времени.

– Агрегированный двойник (Digital Twin Aggregate, DTA) — система, объединяющая все цифровые двойники и их реальные прототипы. Она позволяет собирать данные со всех источников и обмениваться ими в режиме реального времени. Благодаря этому можно получить полное представление о состоянии объекта и его окружающей среды, принимать обоснованные решения и оптимизировать работу системы.

Следующим этапом развития стали персональные цифровые двойники, ориентированные на конкретного человека, получившее наиболее признание в области бизнес-аналитики. В настоящее время крупные компании нацелены на адаптацию своих продуктов или услуг к определённой, достаточно узкой целевой аудитории или отдельным клиентам. Для этого в продукцию вносят необходимые конструктивные и дизайнерские решения, формируют для клиентов специальные условия и предложения (например, по кредитам, вкладам или условиям обслуживания), т.е. осуществляют процесс персонализации, инструментом реализации которой стал цифровой двойник клиента.

Цифровой двойник клиента (ЦДК) – это виртуальное представительство, цифровой аватар клиента или группы клиентов. С его помощью можно изучать предпочтения потенциальных клиентов, моделировать их поведение и результаты работы. ЦДК могут использоваться для самых разных целей в бизнес-аналитике, основная из которых оптимизация пути клиента CJO (Customer Journey Orchestration) – процесс улучшения взаимодействия клиента с компанией на всех этапах его жизненного цикла.

Исходные данные и данные сторонних организаций, собранные с их фирменных сайтов и других источников, являются основой профиля клиента. Поскольку эти данные могут устаревать и их становится все труднее получить, организации используют синтетические данные, т.е. сами генерируют их для создания и расширения виртуальных профилей клиентов.

Используя ЦДК маркетологи могут:

– создавать персонализированные сообщения и предложения для каждого отдельного клиента, а не заставлять их идти по заранее определенному пути, предложенному компанией. ЦД могут быть включены в сценарии CJO, чтобы предоставить реальным клиентам наилучший план действий, основанный на сочетании профилей и поведения других клиентов;

– понять, насколько новый продукт или услуга будет интересен целевой аудитории за счет использования ЦДК для оценки новшеств до их запуска и продвижения.

В настоящее время, устойчивая тенденция внедрения ЦД столкнулась с существенными проблемами управления данными, так как различные компании могут использовать разные определения и отношения в своих моделях, увеличивая несовместимость систем. Необходимо разработать новую структуру для связанных двойников, основанную на онтологическом подходе, т.е. когнитивные цифровые двойники (КЦД, Cognitive gigital twin).

Когнитивные цифровые двойники

В литературе существуют различные объяснения термина, например, КЦД это:

– виртуальное представление физического объекта или системы на протяжении всего его жизненного цикла с использованием данных реального времени, получаемых при помощи Интернета вещей, обеспечивающее обучение, рассуждение и автоматическую корректировку для принятия оптимальных решений [3];

– цифровые двойники с расширенными семантическими возможностями для выявления динамики эволюции виртуальных моделей, способствующие пониманию взаимосвязей между ними [2];

– усовершенствованная версия ЦД за счет использования новых возможностей коммуникации и искусственного интеллекта на трех уровнях – доступа, аналитики и познания [4].

Таким образом можно выделить общие характеристики КЦД:

– содержит несколько моделей ЦД с унифицированными определениями топологии семантики;

– позволяет выполнять интеллектуальную деятельность, такую как рассуждение, прогнозирование, принятие решений, т.е. он предназначен для динамического распознавания сложного и непредсказуемого поведения с помощью стратегий оптимизации;

– должен состоять из цифровых моделей, охватывающих различные этапы всего ЖЦ системы (от проектирования до окончания срока службы), а также обеспечивать интеграцию и анализ получаемых данных и знаний с этих этапов, для поддержания когнитивной деятельности;

– должен быть автономным и минимизировать уровень человеческого вмешательства;

– должен иметь возможность развиваться вместе с реальной системой на протяжении всего жизненного цикла.

В качестве ключевых отличий между концепциями ЦД и КЦД можно выделить следующее:

1) КЦД базируется на ЦД, т.е. включает в себя все существенные характеристики цифрового двойника. Например, оба определяются тремя составляющими: физическим прототипом в реальном пространстве; виртуальным аналогом в цифровом пространстве; взаимодействием со своими прототипами и возможностью динамического обновления. С этой точки зрения концепцию КЦД можно рассматривать как подмножество ЦД – все КЦД представляют собой определенный вид цифрового двойника с расширенными характеристиками.

Основные различия между ними заключаются в сложности структуры и когнитивных способностях. ЦД, как правило, представляют конкретный продукт и сосредоточены на одном из его этапов ЖЦ. Напротив, КЦД должен состоять из множества цифровых моделей, соответствующих различным подсистемам и компонентам сложной системы, и фокусироваться на нескольких этапах жизненного цикла. Он может быть построен путем интеграции связанных ЦД с использованием онтологий и семантического моделирования.

2) Когнитивные способности необходимы для КЦД, но не обязательны для ЦД. Большинство ЦД применяются для мониторинга состояния, функционального моделирования, динамического планирования, обнаружения аномалий, профилактического обслуживание и т.д., что обычно реализуется с помощью алгоритмов, основанных на моделях, и данных, собранных от физических объектов.

Когнитивные способности предназначены для создания более высокого уровня автоматизации, за счет достижения семантической совместимости разнородной информации. Для решения этой проблемы необходимо использовать семантическое моделирование и системное проектирование. Важно отметить, что концепция КЦД не предлагается вместо ЦД, а рассматривается как один из возможных вариантов усовершенствования цифрового двойника.

Таким образом когнитивный цифровой двойник — это цифровое представление физической системы, включающее в себя набор семантически связанных цифровых моделей, относящихся к различным этапам жизненного цикла прототипа, а также дополненное определенными когнитивными возможностями, такими как поддержка выполнения автономных действий.

Главные отличия между концепциями цифрового двойника и когнитивного цифрового двойника заключаются в структуре и наличие когнитивных способностей. Цифровой двойник способен представлять

систему, например, машину, сборочную линию или весь завод в сфере производства. КЦД нацелены на более сложные сценарии, когда задействовано несколько систем, особенно если они имеют разные стандарты и спецификации данных.

Заключение

В данной статье авторами были рассмотрены основные этапы развития цифровых двойников. Зародившаяся в 2000-х концепция обрела особую популярность в настоящее время, как мощный инструмент для прогнозирования поведения и оптимизации работы физического объекта, за счет использования данных, получаемых на стадиях всего его жизненного цикла.

В связи с развитием информационных технологий процесс интеграции ЦД в сложные производственные системы является актуальной задачей, поскольку такие системы обычно содержит несколько подсистем или компонентов, имеющих собственные модели ЦД, созданные на основе различных протоколов и стандартов, следовательно, с неоднородной структурой с точки зрения синтаксиса и семантики. Решением данной проблемы является когнитивный цифровой двойник – следующий шаг в эволюции цифровых двойников.

Благодаря наличию расширенных возможностей коммуникации и искусственного интеллекта на трех уровнях – доступа, аналитики и познания, КЦД могут развиваться, объединяя различные источники информации для определенной цели.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Кошелева М.С., Тобратов Ю.М. Цифровые двойники: основные понятия и интерпретации/ Кошелева М.С., Тобратов М.Ю.// Новые информационные технологии в научных исследованиях: Материалы XXVIII Всероссийская науч.-техн. конф. студ., мол. уч. и спец. – Рязань: Book Jet, 2023. – С. 21-23.

2. Abburu, Sailesh, A. J. Berre, M. Jacoby, D. Roman, L. Stojanovic, and Nenad Stojanovic. 2020a. “Cognitive Digital Twins for the Process Industry.” In COGNITIVE 2020, The Twelfth International Conference on Advanced Cognitive Technologies and Applications, 68–73.

3. Adl, Ahmed El. 2016. The Cognitive Digital Twins: Vision, Architecture Framework and Categories. Technical Report. https://www.slideshare.net/slideshow/embed_code/key/JB6

4. Ali, Muhammad Intizar, Pankesh Patel, John G. Breslin, Ramy Harik, and Amit Sheth. 2021. “Cognitive Digital Twins for Smart Manufacturing.” IEEE Intelligent Systems, 36 (2): 96-100.

УДК 004.627

КУЗНЕЦОВ Н.А., СКВОРЦОВ С.В.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

АНАЛИЗ ОСОБЕННОСТЕЙ АЛГОРИТМОВ СЖАТИЯ ТЕКСТОВЫХ ДАННЫХ

Рассматриваются основные методы сжатия текстовых данных, их достоинства и недостатки, а также рекомендации по применению.

Сжатие данных обусловлено необходимостью сокращения затрат компьютерных ресурсов, таких как машинное время или машинная память, при решении различных прикладных задач. Например, при передаче информации по каналам связи вычислительной сети сжатие позволяет экономить машинное время. Если объем доступной памяти ограничивает возможности хранения информации, то сжатие позволяет увеличить количество сохраняемых данных.

Методы сжатия могут быть с потерями и без потерь. Сжатие без потерь позволяет после распаковки сжатых данных получить информацию, полностью идентичную исходной. При сжатии с потерями распакованная информация отличается от оригинала, но во многих случаях крайне незначительно.

Следует подчеркнуть, что при сжатии текста требуется использовать сжатие без потерь, так как отличие даже в одном бите может привести к искажению текста и его семантики. Эффективность алгоритмов сжатия оценивается величиной, которая называется степенью сжатия. Покажем, как вычисляется степень сжатия на примере алгоритма Хаффмана.

Алгоритм Хаффмана основан на жадном методе [1, 2]. Он выполняет построение бинарного дерева, описывающего структуру кодов переменной длины. Такая задача является примером прикладной задачи, для которой жадный метод всегда даёт оптимальное решение.

Пусть задан алфавит $\{A, B, C, D, \#\}$ с вероятностями (относительными частотами) символов, показанными в таблице 1.

Таблица 1

Символ	A	B	C	D	#
Частота	0,35	0,1	0,2	0,2	0,15

Предположим, что полученные по алгоритму Хаффмана коды символов приведены в таблице 2.

Таблица 2

Символ	A	B	C	D	#
Код	11	100	00	01	101

Тогда ожидаемую степень сжатия для рассмотренного примера можно вычислить следующим образом [1]. Очевидно, что при использовании кода фиксированной длины требуется не менее чем три бита для каждого символа. Применение кодов переменной длины с учетом заданных частот появления символов в тексте дает следующее среднее количество битов для кодирования одного символа:

$$2 * 0{,}35 + 3 * 0{,}1 + 2 * 0{,}2 + 2 * 0{,}2 + 3 * 0{,}15 = 2{,}25$$

Отсюда ожидаемая степень сжатия составит:

$$\frac{(3 - 2{,}25)}{3} = 0{,}25 = 25\%$$

На практике степень сжатия для кодирования Хаффмана обычно оказывается в диапазоне от 20% до 80% и зависит от характеристик сжимаемого файла.

Эффективность сжатия с помощью алгоритма Хаффмана пропорциональна объёму сжимаемого текста и зависит от разнообразия символов. Чем более объёмный текст и менее разнообразный, тем эффективнее сжатие и незначительнее увеличение объёма памяти, необходимого для хранения таблицы кодов.

Адаптивный алгоритм Хаффмана [3] является модификацией обычного алгоритма Хаффмана сжатия сообщений. Он позволяет ограничиться одним проходом по сообщению, как при кодировании, так и при декодировании. Сущность адаптивного алгоритма состоит в том, что при каждом сопоставлении символу кода изменяется логика вычислений так, что в следующий раз этому же символу может быть сопоставлен другой код, т.е. происходит адаптация алгоритма к поступающим для кодирования символам. При декодировании происходит аналогичный процесс.

В адаптивном алгоритме сжатия Хаффмана используется упорядоченное бинарное дерево. Бинарное дерево называется упорядоченным, если его узлы могут быть перечислены в порядке убывания веса. Перечисление узлов происходит по ярусам снизу-вверх и слева-направо в каждом ярусе. Узлы, имеющие общего родителя, находятся рядом на одном ярусе.

Метод LZW [4] реализует сжатие без потерь, и ориентирован на работу с текстовыми данными. В его основе лежит поиск повторяющихся последовательностей символов, сохранение их в словаре и кодирование текста с использованием индексов строк полученного словаря.

Метод LZW выполняет один проход по входным данным, как при сжатии, так и при распаковке. И в том, и в другом случаях, алгоритм строит словарь последовательностей просмотренных символов. Для их представления используются символы данного словаря. Словарь рассматривается как массив символьных строк произвольной длины, индексы элементов которого позволяют обращаться к найденным ранее последовательностям символов.

При сжатии текста и при его распаковке в словарь сначала записываются одно символьные строки, состоящие из символов используемого алфавита.

Для полного набора символов таблицы ASCII словарь будет начинаться с 256 односимвольных строк, где в i-й позиции словаря находится символ алфавита, ASCII-код которого равен значению i.

В процессе сжатия алгоритм создаёт новые строки, добавляет их в словарь и в качестве результата возвращает индексы этих строк в словаре. Например, пусть при сжатии текста первым считывается символ T. В словаре отыскивается T (он там обязательно есть). Каждый раз, когда программа сжатия находит в словаре некоторую строку, она считывает следующий входной символ и добавляет его к этой строке.

Эффективность метода LZW существенно зависит от скорости поиска данных в словаре. При распаковке требуется только установить факт наличия считанного индекса в словаре. Однако при упаковке требуется полноценный поиск строки в словаре. Простейший вариант – линейный поиск, который требует время $O(n)$. Для ускорения поиска обычно используются специализированные структуры данных, в частности хеш-таблицы.

Алгоритм Хаффмана, адаптивный алгоритм Хаффмана и LZW сжатие являются различными методами сжатия данных с их собственными особенностями. В частности, алгоритм Хаффмана имеет следующие отличительные черты: высокий коэффициент сжатия; простота реализации и эффективность.

Особенностями адаптивного алгоритма Хаффмана являются: адаптация к изменяющимся данным, что позволяет реализовать функцию добавления к исходному тексту дополнительной информации без необходимости полного перестроения дерева Хаффмана; улучшенная эффективность.

LZW сжатие имеет следующие особенности: эффективное сжатие повторяющихся фрагментов текста; независимость от словаря.

Все рассмотренные алгоритмы имеют свои достоинства и подходят для различных типов данных. Алгоритм Хаффмана эффективен для текстовых данных с неравномерным распределением символов, тогда как LZW сжатие хорошо работает с повторяющимися фрагментами данных. Адаптивный алгоритм Хаффмана подходит для данных с изменяющимся распределением символов. Выбор конкретного алгоритма зависит от особенностей входных данных и доступных ресурсов.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Кузнецов Н.А., Скворцов С.В. Применение алгоритма Хаффмана для сжатия текстовых данных // Информационные технологии в прикладных исследованиях. Рязань: ИП Коняхин А.В. (Book Jet), 2023. – 144 -148 с.

2. Левитин А.В. Алгоритмы: введение в разработку и анализ / Ананий Левитин; [пер. с англ. и ред. С. Г. Тригуб, И. В. Красикова]; Ун-т Вилланова. – Москва [и др.]: Вильямс, 2006. – 574 с.

3. Адаптивный алгоритм Хаффмана сжатия информации / М.А. Кудрина, К.А. Кудрин, О.А. Дегтярева, Е.В. Сопченко // Труды Международного симпозиума «Надежность и качество»: электронный журнал. – URL: https://cyberleninka.ru/article/n/adaptivnyy-algoritm-haffmana-szhatiya-informatsii.

4. Сжатие данных LZW // Хабр: сайт [Электронный ресурс]. – URL: https://habr.com/ru/companies/otus/articles/581728/.

УДК 004.42

ЛОГИНОВ В.И.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

СРАВНЕНИЕ ОСНОВНЫХ АЛГОРИТМОВ ПЛАНИРОВАНИЯ ПРОЦЕССОРНОГО ВРЕМЕНИ

В статье рассматриваются основные алгоритмы планирования процессорного времени, а также их достоинства и недостатки.

Алгоритмы планирования процессорного времени

Алгоритмы планирования процессорного времени решает проблему выбора процесса из очереди готовых к исполнению

процессов [1]. Выделяют несколько основных алгоритмов для равномерного распределения процессорного времени:

1) «FirstCome – FirstServed» (FCFS, т.е. первый зашёл – первый обслужен);

2) «Round-Robin» (Круговой алгоритм);

3) «ShortestJobNext» (SJN, сначала кратчайшая работа);

4) «Priority Scheduling» (Приоритетное планирование) [1];

5) «MultilevelQueue Scheduling» (Многоуровневое планирование очереди) [2].

Алгоритм FCFS

Наиболее простым и очевидным является алгоритм планирования FCFS. Суть его работы состоит в том, что процессор обслуживает очередь в прямом порядке. Новый процесс попадает в конец очереди. После того как процессор завершит обслуживание очередного процесса, он перейдёт к первому в очереди процессу. Для оценки эффективности работы алгоритма будем использовать среднее время ожидание в очереди.

Например, имеются 5 процессов Р. Их время выполнения будем считать в миллисекундах:

P1(17 мс), P2(3 мс), P3(2 мс), P4(5 мс), P5(1 мс).

Если эти процессы поступают на обработку в порядке *P1, P2, P3, P4, P5*,то процесс *P1* ожидает 0 мс, процесс *P2* – 17 мс, *P3* – 20 мс, *P4* – 22 мс и *P5* – 27 мс.

В таком случае их среднее время ожидания будет равняться 17,2 мс. Если же мы поменяем порядок в очереди на *P5, P2, P3, P4, P1*, тогда среднее время ожидания будет равным 4,4 мс. Как мы видим из примера, среднее время ожидания может сильно отличаться в зависимости от процессорного времени, нужного процессам.

Алгоритм FCFS – невытесняющий, то есть процессорудерживается процессом до тех пор, пока сам процесс не выполнится [1]. Данный алгоритм не рекомендуется использовать в системах, где каждый пользователь должен регулярно получать долю процессорного времени.

Круговой алгоритм Round-Robin

Алгоритм RR сильно напоминает FCFS, однако имеет одно существенное отличие. Round-Robin – *вытесняющий* алгоритм, то есть процессор может переключиться на следующий процесс, если прошло определенное время [1]. Это время называется *квантом времени.* Обычно квант имеет величину от 10 до 100 миллисекунд. Алгоритм Round-Robin выделяет каждому процессу по кванту времени, если процесс не завершился в течение этого кванта, то он переходит в конец

очереди, которая в данном случае является круговой. Если процесс завершается раньше выделенного ему времени, то он отпускает процессор, который переходит к обработке следующего процесса в очереди.

Рассмотрим его работу на примере:

P1(17 мс), P2(3 мс), P3(2 мс), P4(5 мс), P5(1 мс). Квант времени равен 4 мс.

Процесс P1 получит квант, длинной 4 мс, а затем будет приостановлен. Ему будет необходимо ещё 13 мс.

Процесс P2 получит квант, но выполнится быстрее на 1 мс, значит процессор будет передан P3.

Процесс P3 так же будет выполнен раньше на 2 мс.

Процесс P4 не успеет завершиться, т.к. ему будет необходимо ещё 1 мс.

Процесс P5 завершится за 1 мс. Осталось 2 ожидающих процесса.

Процесс P1 снова не успеет закончить работу, ему необходимо ещё 9 мс.

Процесс P4 завершится за 1 мс и освободит процессор.

Процессу P1 необходимо ещё 2 раза передать процессор самому себе, для того, чтобы завершить работу, рисунок 1.

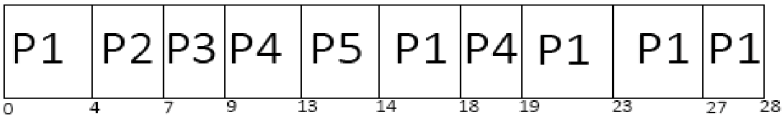


Рисунок 1 – Пример работы алгоритма

Среднее время ожидания будет равняться 10,4 мс.

Эффективность алгоритма Round-Robin во многом зависит от длины кванта времени. В случае, когда длина кванта равна бесконечности, алгоритм, по сути, превращается в FCFS. Если квант времени очень мал, тогда алгоритм принимает вид разделенного. Это можно представить, как многопроцессорную систему, в которой каждый процесс выполняется на своём процессоре. Так как мы рассматриваем случай, когда физически процессор один, то его мощность разделяется на n процессов. То есть, каждый процесс работает только со своим «процессором» со скоростью 1/n от реального, физического процессора.

Так же при выборе кванта времени, необходимо учитывать переключения контекста. Слишком маленькое значение кванта чревато большим количеством переключений, что приводит к снижению скорости. В рассмотренном выше примере видно, что при величине кванта в 4 мс процесс *P4(5 мс)* обрабатывается за 2 переключения, а процесс *P1(17 мс)* за 4. Если увеличить квант до 5 мс, тогда число переключений снизится, как и среднее время ожидания с 10,4 мс до 9,8 мс. Желательно, чтобы квант был на много больше времени переключения контекста. Рекомендуется использовать такой квант времени, чтобы 80% процессов были короче него.

Алгоритм Shortest Job Next

Алгоритм SJN распределяет задачи в порядке возрастания по их времени выполнения. Такой подход минимизирует среднее время ожидания, так как самые короткие задачи выполняются первыми, что уменьшает ожидание у последующих процессов. В нашем примере *P1(17 мс), P2(3 мс), P3(2 мс), P4(5 мс), P5(1 мс)* задачи будут располагаться в таком порядке: *P5(1 мс), P3(2 мс), P2(3 мс), P4(5 мс), P1(17 мс).*

Таким образом, среднее время ожидания станет минимальным и будет равняться 4,2 мс. Однако у алгоритма SJN есть существенный минус – в общем случае, его нельзя реализовать на уровне планирования, так как нет возможности точно узнать время выполнения процесса. Одним из способов реализации алгоритма является аппроксимация времени выполнения следующей задачи. Стоит заметить, что алгоритм SJN может быть как вытесняющим, так и невытесняющим. Это зависит от того, хотим ли мы вытеснять текущий процесс, если в очереди появилась более короткая задача. В случае с вытеснением алгоритм называют «Shortest – Remaining – Time – First», что означает: кратчайший оставшийся участок – первым.

Алгоритм планирования с приоритетом Priority Scheduling

Рассмотренные ранее алгоритмы FCFS и SJN являются частными случаями алгоритма Priority Scheduling. В FCFS у всех процессов был одинаковый приоритет, а в алгоритме SJN приоритетным считался кратчайший процесс. Приоритет - число в каком-то диапазоне от 0 до N, где N – какое-либо фиксированное число (обычно 7 или 4095). В разных источниках 0 соответствует, как и наивысший, так и наименьший приоритет. В данной работе будем считать, что числу 0 соответствует наивысший приоритет. На примере рассмотрим работу алгоритма планирования с приоритетом. Даны процессы с приоритетом:

P1 (Приоритет = 3; 17 мс);

P2 (Приоритет = 1; 3 мс);
P3 (Приоритет = 4; 2 мс);
P4 (Приоритет = 5; 5 мс);
P5 (Приоритет = 2; 1 мс);

В таком случае, порядком их обработки в соответствии с приоритетом будет являться *P2, P5, P1, P3, P4*. Среднее время ожидания равно 10,2 мс.

Приоритеты могут быть внутренними или внешними [1]. Для вычисления внутренних приоритетов используются какие-либо измеримые величины внутри операционной системы, например ограничение по времени или памяти. Внешние же приоритеты задаются критериями, не зависимыми от операционной системы. Планирование с приоритетом может быть вытесняющим и невытесняющим. Главной проблемой алгоритмов с приоритетом (в частности вытесняющих) является так называемая «неопределенная блокировка» (indefinite blocking), когда есть вероятность, что какой-либо процесс с низким приоритетом будет вытесняться процессами с более высоким приоритетом. Одним из способов решить данную проблему является «механизм старения», который постепенно повышает приоритет процесса, ожидающего долгое время.

Алгоритм Multilevel Queue Scheduling

В тех случаях, когда задачи можно разбить на некоторое количество групп, имеет место использование алгоритма многоуровневой очереди. Такими группами процессов могут быть приоритетные и фоновые процессы в системе. Алгоритм MQS разбивает начальную очередь на несколько отдельных очередей. Процессы навсегда привязываются к этим очередям. Каждая такая очередь может обслуживаться своим алгоритмом планирования. Так же, необходимо организовать планирование между новыми очередями. Когда приоритеты очередей заданы, очередь *Q0* с приоритетом 0 будет иметь абсолютный приоритет над всеми очередями *Q1, Q2, … , Qn-1, Qn* с меньшим приоритетом. В таком случае, если в момент обработки очереди *Q1* в очередь *Q0* поступит процесс, то задача из очереди *Q1* будет вытеснена, тогда начнёт выполняться задача из очереди *Q0*.

Можно разделить очереди по выделенному им процессорному времени. В таком случае, каждая очередь получит часть времени, которую она самостоятельно распределяет между своими процессами.

Достоинства и недостатки рассмотренных алгоритмов

Рассмотрим достоинства и недостатки, рассмотренных ранее алгоритмов планирования:

Алгоритм FCFS:

Достоинства**:**

Алгоритм FCFS очень прост в реализации, а также не накладывает расходов на переключение контекста.

Недостатки:

При использовании алгоритма в интерактивных системах, возможны возникновения проблем с отзывчивостью, так как долгие процессы могут существенно замедлить короткие в очереди. Не учитывает важности процессов.

Круговой алгоритм Round-Robin:

Достоинства:

Алгоритм RR прост в реализации, гарантирует равное распределение процессорного времени между задачами, а так же подходит для использования в интерактивных системах.

Недостатки:

Существует вероятность возникновения излишнего накладного расхода при частом переключении контекста. Не учитывает важности процессов.

Алгоритм SJN:

Достоинства:

Алгоритм SJN обеспечивает минимальное время ожидания и выполнения процессов.

Недостатки:

Главным недостатком алгоритма SJN является сложность сбора информации о длительности процессов.

Алгоритм планирования с приоритетом:

Достоинства:

Данный алгоритм обеспечивает приоритет для важных процессов. Является довольно гибким в настройке приоритетов в соответствии с требованиями ОС.

Недостатки:

Вероятность появления проблемы «неопределенной блокировки» процесса. Так же, при неоптимальном назначении приоритетов, процессорное время используется неэффективно.

Алгоритм многоуровневого планирования очередей MQS:

Достоинства:

Алгоритм позволяет разделять процессы на группы и применять к ним различные алгоритмы планирования. MQS довольно гибок в настройке.

Недостатки:

Из-за сложности алгоритма, могут возникнуть проблемы с производительностью и реализацией при большом числе уровней и неправильном балансе между ними.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Костров, Б.В. Основы теории вычислительных систем: методические указания к лабораторнымработам / Рязан. гос. радиотехн. ун-т.; Сост.: Б.В. Костров, В.Ю. Чамкин. – Рязань, 2006. – 32 с.

2. Попов, И.С. Операционные системы: планирование и выполнение процессов: методическиеуказания / И.С.Попов; МГУ им. М.В. Ломоносова. – Москва, 2015. – 50 с.

УДК 004.65

МЕЛИХОВ А.Ю., БАСТРЫЧКИН А.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ИССЛЕДОВАНИЕ РАБОТЫ БРОКЕРОВ СООБЩЕНИЙ ДЛЯ АСИНХРОННОЙ ПЕРЕДАЧИ НА ПРИМЕРЕ РЕАЛИЗАЦИИ RABBITMQ

Рассматривается реализация механизма асинхронной передачи сообщений RabbitMQ, её внутреннее устройство и работа

Использование брокеров сообщений даёт множество преимуществ в системе нескольких сервисов. Из этих преимуществ главным образом можно выделить:

- Непрерывность передачи сообщений. Сообщения, отправленные брокеру, сохраняются у него до тех пор, пока они не будут прочитаны другим сервисом, благодаря чему отправитель может отправлять сообщения, несмотря на то, проявляет активность потребитель в настоящее время или нет.
- Асинхронность передачи сообщений. За счёт асинхронной обработки задач можно увеличить производительность системы в целом.
- Надёжность передачи сообщений. Для обеспечения надёжности доставки сообщений: как правило, брокеры обеспечивают механизмы многократной отправки сообщений в тот же момент или через определённое время. Кроме того, обеспечивается

соответствующая маршрутизация сообщений, которые не были доставлены.

Эти достоинства безусловно повышают надёжность систем, но иногда их возможности могут превышать потребности. Так, например, для передачи изображения с камер реального времени может и не потребоваться надёжность передачи сообщений. При передачи таких изображений необходима своевременная и асинхронная передача данных, так как сервису, принимающему эти изображения, может потребоваться время для их обработки, а сама обработка должна производиться для актуальных кадров.

Для удовлетворения этих потребностей может послужить брокер RabbitMQ. Он обеспечивает асинхронную и высокопроизводительную передачу сообщений между сервисами.

Этот брокер при управлении сообщениями не делит их между партициями как это происходит в брокере Apache Kafka, в виду чего сообщения могли бы восстановиться при утрате. Однако RabbitMQ поддерживает основные стратегии передачи сообщений: «как минимум однократная доставка» и «как максимум однократная доставка»

В общем виде схема передачи сообщения выглядит следующим образом (рисунок 1):

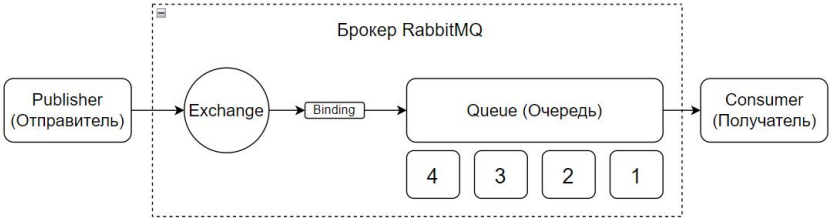


Рисунок 1 – Общая схема обмена сообщениями

Ключевые понятия необходимые для рассмотрения RabbitMQ:

- Publisher — публикует сообщения в Rabbit.
- Exchange — обменник. Сущность Rabbit, точка входа для публикации всех сообщений.
- Binding — связь между Exchange и очередью.
- Queue — очередь для хранения сообщений.
- Message — сообщение, атомарная сущность.

• Consumer — подписывается на очередь и получает от Rabbit сообщения.

Publisher. Первым звеном в цепочке передачи сообщения является publisher, он же отправитель. На этом этапе генерируется сообщение и инициируется его передача. Для этого создаётся соединение по протоколу AMQP с брокером и в его рамках создаётся канал.

Publisher может декларировать практически все сущности — exchanges, queues, bindings и др. На практике чаще применяется стратегия декларирования всех нужных сущностей consumer, но решать нужно для каждого проекта индивидуально.

Publisher всегда пишет в exchange с routing key, совпадающим с названием очереди.

Publisher определяет delivery_mode для каждого сообщения — так называемый «признак персистентности». Это значит, что сообщение будет сохранено на диске и не исчезнет в случае перезагрузки Rabbit.

• delivery_mode=1 —Режим без сохранения сообщений в ПЗУ. Ускоряет обработку.

• delivery_mode=2 — Режим с сохранением сообщений в ПЗУ. Увеличивает надёжность.

Также publisher определяет Routing Key для каждого сообщения — признак, по которому идёт дальнейшая маршрутизация в Rabbit.

У сообщений RabbitMQ существует такое понятие как флаг. Флаг может влиять на дальнейшую маршрутизацию сообщения. Флагами сообщения управляет Publisher. Например, publisher может выставлять confirm флаг — отправлять указания Rabbitmq через отдельный канал подтверждения об успешной приёмке сообщений. Также есть флаг mandatory — указание Rabbit складировать сообщения, не имеющие маршрута в какую-либо очередь в отдельный Exchange.

Exchange – это точка входа и маршрутизатор всех сообщений (как входящих от Publisher, так и перемещающихся от внутренних процессов в Rabbit). Exchange представляет собой неизменяемую сущность. Если необходимо изменить параметры Exchange, его необходимо удалить и заново объявить.

Выделяют четыре типа Exchange:

· Fanout; Exchange отправляет сообщения во все очереди, в которых есть привязки (bindings), игнорируя любые настройки привязок (routing key или заголовки). Это самый простой тип и наименее функциональный.

· Direct; Exchange отправляет сообщения во все очереди, в которых привязанный Routing Key полностью совпадает с Routing Key сообщений. Этот тип является наиболее популярным, имеет скорость сравнимую с fanout и при этом обладает необходимой гибкостью для большинства задач.

· Topic; Этот тип Exchange похож на Direct, но поддерживает Wildcard * и # в качестве параметров привязок. * - означает совпадение одного слова, # - совпадение любого количества слов.

· Headers. Этот тип Exchange является наиболее гибким, но наименее производительным типом. Его скорость очень сильно зависит от сложности условий и поэтому труднопрогнозируема. Он оперирует не Routing key, а заголовками сообщений и привязками. В привязках указываются ожидаемые заголовки, а также признак x-match, где x-match=all — для попадания сообщения необходимы все совпадения; x-match=any — необходимо хотя бы одно совпадение.

Binding. Базовая сущность Rabbit, статический маршрут от Exchange до Queue (от обменника до очереди).

Между парой обменника (exchange) и очереди (queue) может быть несколько привязок (bindings), но только с разными параметрами. Параметры привязки могут быть routing key или заголовками, в зависимости от типа Exchange.

Очередь (Queue) – основная сущность в RabbitMQ, которая представляет собой упорядоченное хранилище для необработанных сообщений.

Существует три типа очередей:

Classic – обычная очередь, которая используется в большинстве случаев.

Quorum – аналог классической очереди, но с гарантией консистентности, обеспечиваемой кворумом в кластере.

Stream – новый тип очередей (начиная с версии RabbitMQ 3.9), который пока мало кем используется и аналогичен принципам Apache Kafka.

Consumer – это процесс, который получает сообщения из очереди и выполняет необходимую логику.

Consumer подписывается только на одну очередь. Если необходимо получать сообщения из разных очередей, более правильным будет маршрутизировать их в одну очередь, чем разделять внутри приложения.

Из перечисленных возможностей RabbitMQ видно, что данный брокер позволяет настроить его под разные цели использования. Он может помочь во многих задачах и для большинства решений

RabbitMQ может стать правильным выбором. Его основным минусом остаётся сложное горизонтальное масштабирование. Для больших микросервисных систем с высокими нагрузками и высоким риском потери сообщений лучшим выбором остаётся Apache Kafka. Однако, во всех остальных случаях брокер RabbitMQ показывает себя не только как удобный инструмент для перевода синхронного взаимодействия в асинхронное, но и как очень оптимальный брокер с минимальной задержкой передачи сообщений.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. RabbitMQ: терминология и базовые сущности: [Электронный ресурс]. – 2022. – URL: https://habr.com/ru/companies/slurm/articles/703060/.

УДК 004.056.5

МОСКАЛЕВ И.С., ВЫЖИГИН А.Ю., СЕЛИН А.А., БУГАЕВ А.А.
МИРЭА – Российский технологический университет

СИСТЕМА ЗАЩИТЫ ИНФОРМАЦИИ ДЛЯ БЕЗОПАСНОЙ РАБОТЫ В СЕТИ ИНТЕРНЕТ

В статье рассматривается проблема открытого поиска информации по IP-адресу. Для ее решения разработан алгоритм безопасной работы в сети интернет, а на его основе – система защиты информации (СЗИ).

Введение

В настоящее время существует проблема открытого поиска информации по IP-адресу, поскольку в сети Интернет практически нет данных, содержащих точные руководства для проведения подобных работ. Таким образом, пользователю сложно выявить потенциально опасные IP-соединения. Для решения указанной проблемы проведено исследование, цель которого – применение технологии OSINT для получения информации об IP-адресе с одной стороны и разработка компонента защиты системы с другой.

В рамках исследования были решены следующие задачи:

- изучены материалы о поиске открытой информации об IP-адресе в сети Интернет;
- разработана система поиска, хранения и защиты информации по IP-адресу с применением технологии OSINT.

Технология OSINT

При разработке данной системы использовалась технология получения и анализа информации из открытых источников – Open Source INTelligence (OSINT). OSINT позволяет:

- получать максимально объективную и полезную информацию для принятия решений;
- находить недостатки и уязвимости в собственной системе безопасности, защите конфиденциальных сведений;
- понимать психологические особенности, потребности, привычки представителей целевой аудитории.

OSINT подразумевает получение данных из открытых источников и/или таких, доступ к которым возможен по запросу. Такими источниками являются:

- информационные материалы (статьи, новости, заметки) в СМИ;
- научные исследования, опубликованные в специализированных изданиях;
- книги, энциклопедии, справочники;
- публикации и комментарии в социальных сетях;
- документы из государственных и негосударственных архивов;
- публичные коммерческие данные;
- результаты публичных опросов;
- данные со спутников дистанционного зондирования Земли и самолетов аэрофотосъемки;
- документация судебных делопроизводств;
- другие источники [1].

Применение технологии OSINT

Для проведения эксперимента была применена поисковая система ipinfo.io. Этот сайт предназначен для получения сетевых и географических данных. В качестве примера произведем поиск информации по IP-адресу 98.27.68.1. Введем запрос – http://ipinfo.io/98.27.68.1/json. По ссылке понятно, что результаты будут выведены в формате JSON. JSON (JavaScript Object Notation) – формат данных для обмена сообщениями в веб-приложениях. Построен на паре «ключ-значение». Отличается от словаря тем, что ключи выделяются двойными кавычками (рис. 1).

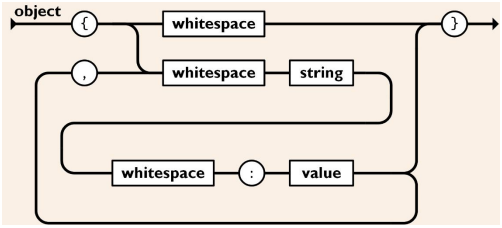


Рисунок 1 – Диаграмма Вирта JSON-файла

По данным сайта, мы можем узнать наименование организации, ее местоположение и сетевые настройки. Эта информация может быть использована на первом этапе определения потенциальных угроз. В дальнейшем потребуются дополнительные данные из других источников.

Разработка механизма получения информации по IP-адресу

Система защиты информации (СЗИ) была разработана с помощью библиотек языка программирования (ЯП) Python и основ нереляционной теории баз данных в виде текстовых документов. Один документ отвечает за список IP-адресов, с которыми отмечались IP-соединения, другой – за хранение данных по IP-адресам, которые уже анализировались. Система получения данных связывает два нереляционных объекта между собой. Ее функционал – получение данных по IP-адресу из специализированных сервисов, таких как IPInfo, Censys, Shodan, Criminal IP, Virus Total и другие [2] (рис. 2).

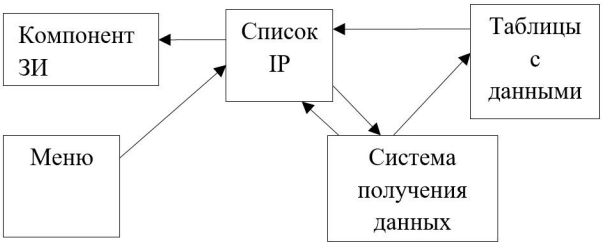


Рисунок 2 – Схема программного обеспечения

Работа данной системы происходит следующим образом: после запуска пользователь вводит IP-адрес, если он незащищен, система выдаст ему информацию об этом. Дальнейшие действия по работе с данными, которая выдала система, зависят от пользователя. Реализована возможность сохранения информации в системе.

Разработка компонента защиты информации

Отдельно стоит обратить внимание на функцию защиты информации, представленную в системе. Алгоритм защиты, как и работа всей системы, связан с IP-адресами. Защита информации от злоумышленников основана на использовании механизма блокировки IP-адресов. Алгоритм реализован с помощью Netsh [3]. Netsh (от англ. NetworkShell – «сетевая оболочка») – это программа для локального и удаленного управления параметрами сети, входящая в стандартный набор инструментов семейства операционных систем Microsoft Windows NT, начиная с Windows 2000. Netsh позволяет менять настройки сетевого соединения с использованием интерфейса командной строки (CLI). Для использования данной утилиты потребуется настройка брандмауэра [4] (межсетевого экрана).

На начальном этапе необходимо создать правило блокировки IP-адресов. Затем сформировать блок-лист IP-адресов, соединения с которыми необходимо блокировать для защиты информации. Работа данного компонента выглядит тривиально. Пользователь выбирает пункт «Защита от атак». Далее для блокировки вписывает IP-адрес в соответствующее пространство для записи, после чего будет запущен алгоритм защиты информации. Результат работы алгоритма отражен на рисунке 3.

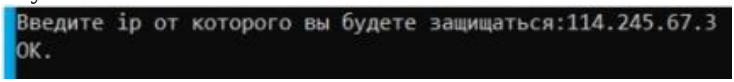


Рисунок 3 – Вывод работы алгоритма защиты информации

Важно учесть, что shell-скрипт, предназначенный для защиты информации, работает только в режиме администратора. Поэтому терминал ОС Windows при работе СЗИ необходимо запускать в режиме администратора. Практическая новизна разработанного алгоритма состоит в простоте выбора данного решения с одной стороны и его несложной реализации с другой. В сравнении с другими алгоритмами на различных языках программирования, алгоритм утилиты Netsh легок в использовании и понимании, что является немаловажным для разработчика (рис. 4).

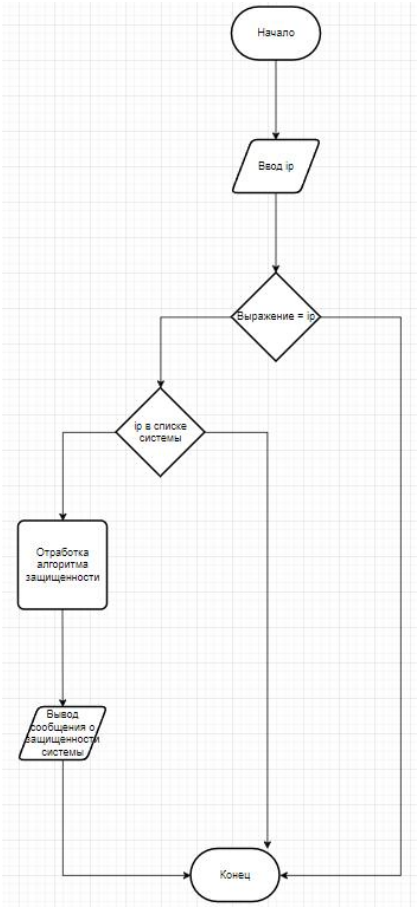


Рисунок 4 – Алгоритм утилиты Netsh

Заключение

В результате проведения экспериментов была создана СЗИ, пригодная для использования в государственных и коммерческих организациях. Базисом системы является получение информации об IP-адресе методами OSINT и, в последующем, принятие решения о необходимости его блокировки на основании априорно известных данных. Таким образом, был реализован алгоритм, повышающий защищенность системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Google Dorking. [Электронный ресурс]. – URL: https://habr.com/ru/company/postuf/blog/510766/.

2. Официальный сайт ipwhois. [Электронный ресурс]. – URL: https://pypi.org/project/ipwhois/.

3. Синтаксис, контексты и форматирование команд Netsh. [Электронный ресурс]. – URL: https://learn.microsoft.com/ruru/windowsserver/networking/technologies/netsh/netsh-contexts .

4. Что такое фаервол. [Электронный ресурс]. – URL: https://ddos-guard.net/ru/blog/chto-takoe-faervol .

УДК 004.891.2

МУШИК А.В.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

РАЗРАБОТКА БАЗЫ ЗНАНИЙ В СИСТЕМЕ PROTÉGÉ

В статье рассматривается процесс разработки базы знаний в системе Protégé, включая основные этапы создания онтологии.

В современном информационном обществе с невероятно быстрым ростом объёма доступной информации поиск действительно значимых знаний и фактов становится всё более сложной задачей. Одним из решений проблемы организации и структурирования такого огромного массива данных являются экспертные системы, основой которых выступают базы знаний. Такие системы – эффективный инструмент выполнения сложных задач и автоматизации принятия решений в различных областях, таких как бизнес, наука, образование, производство, медицина.

Базы знаний и онтологии

База знаний представляет собой хранилище данных, включающее в себя правила вывода и информацию, отражающую человеческий опыт и знания в определенной предметной области. В автономных системах база знаний также содержит информацию, полученную в результате решения предыдущих задач.

Понятие «базы знаний» часто отождествляют с понятием «онтология». Термин «онтология» имеет множество определений в области исследований искусственного интеллекта. В общем смысле онтология – это формальное описание концепций и отношений в предметной области, которое включает набор понятий, их

классификацию, иерархии и отношения между ними. В информационных технологиях онтологии выступают в качестве концептуальных словарей с вычислительной функциональностью, а в контексте экспертных систем – метазнания, описывающие все аспекты рассматриваемой области. Базу знаний формирует не только сама онтология, но и множество индивидуальных экземпляров содержащихся в ней классов и правил вывода, обеспечивающих поддержку принятия решений. Таким образом, онтология является не целью, а скорее средством в разработке базы знаний, помогающим структурировать имеющуюся информацию.

Элементы и этапы создания онтологии

Онтология включает в себя следующие элементы:

- классы (концепты, понятия) и их свойства;
- свойства, описывающие функциональные возможности и атрибуты концепции (слоты, роли);
- ограничения по слотам (аспекты, грани).

Единого универсального подхода к созданию онтологий, гарантирующего успешный результат, не существует. Процесс их разработки является итеративным: сначала создаётся черновой вариант, затем, если это необходимо, производится возврат с целью уточнения деталей. Эта процедура выполняется до тех пор, пока онтология не достигнет определенной степени отражения концепции заданной предметной области. В общем случае можно выделить основные этапы создания онтологии:

1. Анализ предметной области.
2. Определение концептов и их свойств.
3. Организация иерархии концептов (определение базовых классов и подклассов).
4. Определение слотов и их допустимых значений.
5. Создание экземпляров концептов путем заполнения значений слотов.

На каждом из этапов инженеру по знаниям предполагается решить ряд вопросов.

Анализ предметной области включает в себя глубокий анализ исследуемой сферы с целью выявления основных концептов, отношений и характеристик. Здесь важно определить ключевые источники информации, например, документы, экспертные мнения, базы данных или другие ресурсы. Также следует выявить основные потребности и требования пользователей к онтологии.

На втором этапе стоит тщательно подойти к выбору концептов, которые будут наилучшим образом отражать структуру и сущность

предметной области. Каждый из концептов должен быть чётко определён и описан с учетом его свойств, характеристик и взаимосвязей с другими концептами. Помимо прочего необходимо учитывать их атрибуты, типы значений и возможные ограничения.

На этапе организации иерархии следует определить, какие классы являются базовыми, а какие – подчинёнными. Каждый класс и подкласс должен быть правильно распределен и организован с учетом их взаимосвязей и структуры.

На четвёртом этапе организуются слоты, представляющие собой конструкцию, которая собственно и описывает атрибуты и отношения классов. Каждый слот должен иметь определённый тип значений и допустимые значения, обеспечивая таким образом согласованность и стандартизацию. Для слотов, имеющих более сложное представление стоит определить свойства, которые также называются аспектами или гранями. Они являются ограничениями слотов и гарантируют их корректное заполнение.

Последний этап – создание экземпляров классов – наполняет онтологию данными, которые делают из неё полноценную базу знаний. Здесь важно осуществить проверку структуры созданного проекта и внести изменения, если это необходимо. При изменении структуры после ввода данных имеет место проблема потери уже имеющейся в базе информации и дополнительные временные затраты на заполнение значений слотов для созданных экземпляров.

Protégé, созданный в 1999 году командой из Стэнфордского университета, представляет собой бесплатный и открытый редактор онтологий и фреймворк для разработки баз знаний. Платформа Protégé поддерживает два основных способа моделирования онтологий через редакторы Protégé-Frames и Protégé-OWL, позволяя экспортировать созданные онтологии в различные форматы, включая RDF (RDF Schema), OWL и XML Schema. Благодаря открытой и легко расширяемой архитектуре, поддерживающей модули расширения функциональности, Protégé пользуется популярностью в различных сферах, таких как биомедицина, сбор знаний и корпоративное моделирование, и поддерживается значительным сообществом разработчиков, ученых и корпоративных пользователей. Программа доступна для свободного скачивания с официального сайта вместе с плагинами и готовыми онтологиями.

Создание базы знаний в системе Protégé

1. Анализ предметной области.

В рамках статьи будет рассмотрен процесс создания базы знаний для предметной области «управление проектами в области

информационных технологий». Пусть имеется компания под названием «TechSolveSolutions», которая специализируется на управлении IT-проектами. Она предоставляет комплексные решения для разработки программного обеспечения, создания веб-приложений, а также внедрения и поддержки информационных систем. Компания работает с клиентами из различных отраслей, включая финансовый сектор, здравоохранение, образование и государственные учреждения.

В данной предметной области можно выделить положения, которые будут описываться системой:

- процессы управления проектами, включая планирование, выполнение, контроль и закрытие проектов;
- методологии разработки программного обеспечения, например, Agile, Scrum, Waterfall и другие;
- роли и ответственности участников проекта, включая менеджеров проектов, разработчиков, тестировщиков и аналитиков;
- инструменты и технологии, используемые в процессе разработки и управления проектами, такие как системы управления версиями, системы отслеживания ошибок, среды разработки и тестирования и прочее.

2. Определение концептов и их свойств.

Работа над онтологией начинается с создания проекта в Protégé. После выбора формата онтологиии папки, в которой она будет хранится, проект откроется на вкладке «Classes». Теперь можно создать основные классы предметной области: проекты, задачи, участники проекта, методологии разработки, инструменты и технологии, отчёты и документация (рисунок 1):

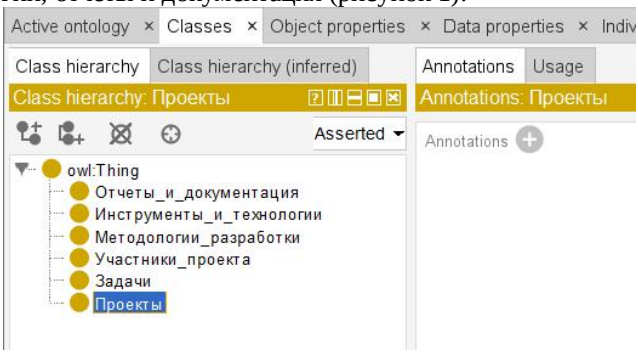


Рисунок 1 – Созданные классы онтологии

Классы онтологии создаются дочерними по отношению к классу «owl:Thing», который является системным. Классы можно создавать с

помощью пункта «Addsubclass» в контекстном меню, сочетанием клавиш Ctrl+E либо нажатием первой кнопки на панели инструментов.

3. Организация иерархии концептов.

Для основных классов будут созданы подклассы, которые их детализируют и описывают более конкретные категории для каждогоаспекта управления проектами (подклассы представлены на рисунке 2).

Выбирая любой из классов, в правом окне на вкладке «Usage» можно увидеть его связь с другими классами, а также его свойства, экземпляры и прочие характеристики.

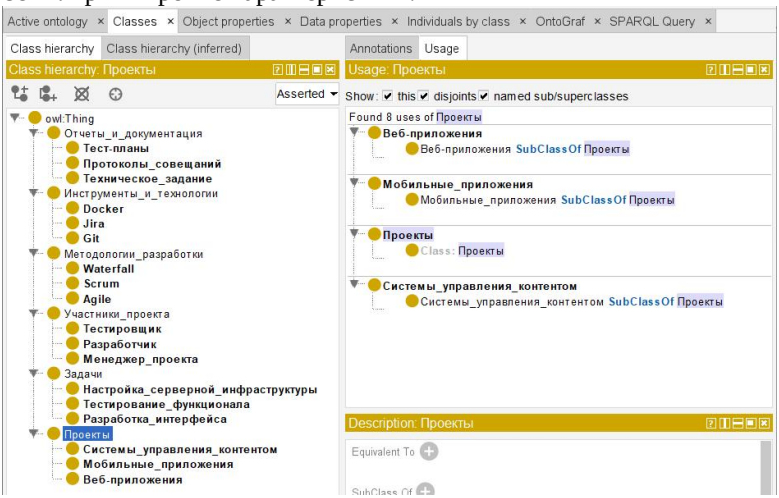


Рисунок 2 – Созданные подклассы и отображение их иерархии

4. Определение слотов и их допустимых значений.

Слоты удобно представлять, используябазовую концепцию языка представления знаний RDF– тройки. RDF-тройки состоят из трёх элементов: субъекта, предиката и объекта. Предикат – это свойство, устанавливающее бинарное отношение между субъектом и объектом. Существует два вида слотов: свойства-отношения и свойства-данные.

Свойства-отношения собственно и представляют собой предикаты, определяющие некоторые отношения между двумя классами. Эти слоты задаются на вкладке «ObjectProperties» и могут быть выражены в форме глагола. Для них указываются домены («Domains»)и диапазоны («Ranges»). В RDF-тройке домен равен субъекту, а диапазон – объекту.

Например, для предметной области «управление проектами в области информационных технологий» можно создать следующие слоты отношения (рисунок 3):

- участвует («Участник проекта» Участвует в «Проекте»);
- выполняет («Участник проекта» Выполняет «Задачу»);
- используется (в «Проекте» Используется «Инструмент и технология», в «Проекте» Используется «Методология разработки»);
- готовится (по «Проекту» Готовится «Отчет и документация»).

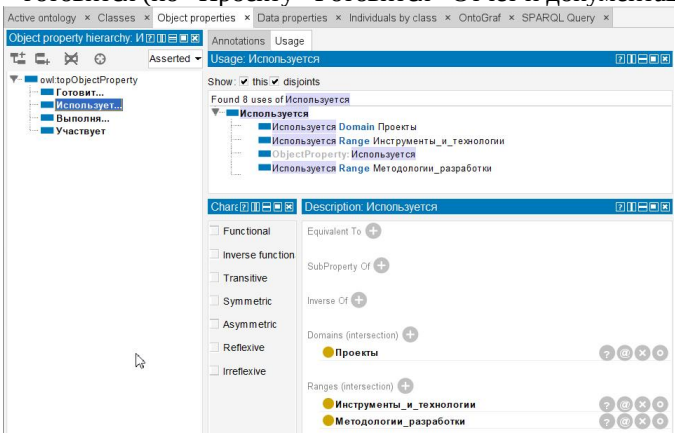


Рисунок 3 – Созданные свойства-отношения

Свойства-данные описывают характеристики экземпляров классов (рисунок 4).

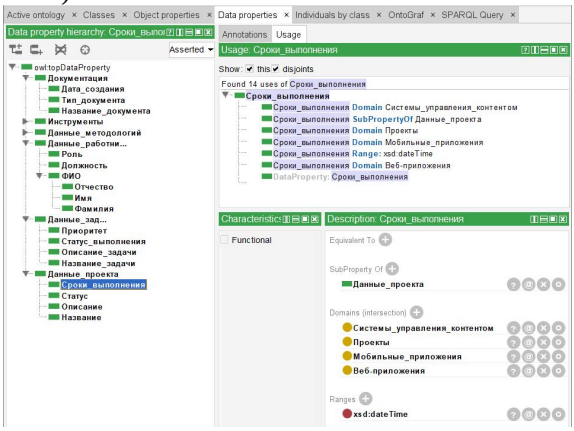


Рисунок 4 – Созданные свойства-данные

Здесь также предлагается указать домен и диапазон. Домен определяет экземпляры классов, для которых данное свойство может быть использовано. Диапазон указывает на область допустимых значений, то есть тип данных и ограничения.

5. Создание экземпляров концептов

Заполнение базы знаний экземплярами классов осуществляется на вкладке «Individualsbyclass» (рисунок 5).

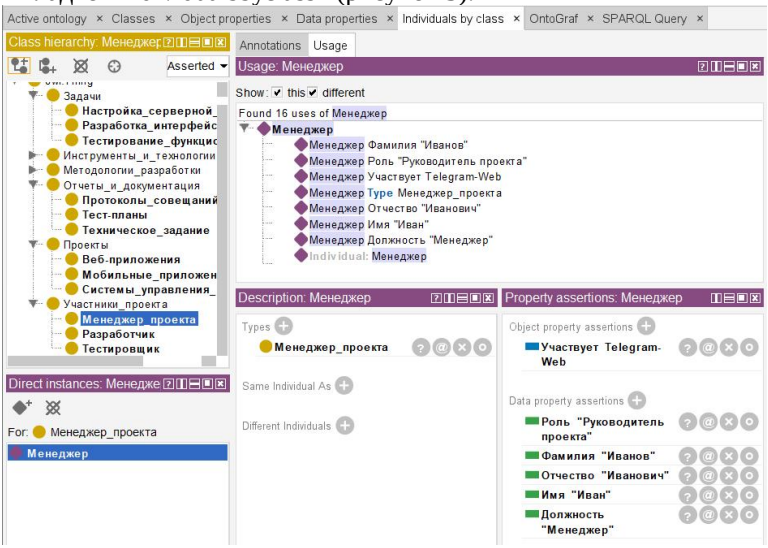


Рисунок 5 – Создание экземпляра класса менеджер проекта

При создании экземпляра можно задать ему существующие свойства-отношения и свойства-данные и ввести их значения.

Базы знаний играют ключевую роль в современном мире информационных технологий, обеспечивая хранение, организацию и доступ к ценной информации. Преимущества использования баз знаний включают в себя улучшение эффективности и точности принятия решений, повышение производительности, а также ускорение процессов поиска и анализа информации. Кроме того, базы знаний способствуют лучшему взаимодействию между системами и людьми, обеспечивая более эффективное использование информации для достижения поставленных целей и задач. Все это делает базы знаний неотъемлемой частью современной информационной инфраструктуры и ключевым инструментом для инноваций и развития в различных областях деятельности.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Дж. Джарратано, Г. Райли. Экспертные системы: принципы разработки и программирование, 4-е издание.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 1152 с.

2. Рубашкин, В.Ш. Онтологическая семантика [Текст]: знания, онтологии, онтологически ориентированные методы информационного анализа текстов / В.Ш. Рубашкин. – Москва: Физматлит, 2012. – 346 с.

3. Муромцев, Д.И. Онтологический инжиниринг знаний в системе Protege: учебно-методическое пособие / Д.И. Муромцев. – Санкт-Петербург: НИУ ИТМО, 2007. – 62 с.

УДК 004.4

НИКОНОВ К.А., САПРЫКИН А.Н.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ОПТИМИЗАЦИЯ ЗАПРОСОВ К БАЗЕ ДАННЫХ В ЦИФРОВОЙ СИСТЕМЕ ОБУЧЕНИЯ ДЛЯ ОБЕСПЕЧЕНИЯ БЫСТРОЙ ЗАГРУЗКИ СТРАНИЦ И РЕСУРСОВ

В статье представлено исследование оптимизации запросов, данная тема проиллюстрирована примерами реализации в области цифрового обучения.

В современном мире существует большое количество задач, которые требуют оперативной, полной и, главное, быстрой выгрузки данных из базы данных. Многие программисты упускают оптимизацию запросов, опираясь на аппаратные возможности устройств, но с ростом информации также растет время отклика системы.

В данной работе рассмотрим способы обеспечения быстрой выгрузки данных на примере спроектированной системы дистанционного обучения школьников. В качестве основного инструмента работы будем использовать язык SQL, который позволяет создать базу данных, а также обращаться к ней. Работа с данными осуществляется с помощью SQL-запросов – это код, который осуществляет различные операции с данными, такие как сортировка, извлечение, удаление и т.д.

Стоит заметить, что оптимизация запросов к базе данных не является единственным способом обеспечить быстрый отклик системы. Выделим несколько ключевых вариантов:

1. Оптимизация запросов к БД
2. Проектирование БД
3. Нормализация данных
4. Индексация БД
5. Кэширование данных
6. Параллельное выполнение запросов

Оптимизация на этапе проектирования

Рассмотрим некоторые способы подробнее: нормализация данных на этапе проектирования может привести к улучшению производительности запросов. Основная цель этого метода заключается в устранении избыточности данных и различных аномалий. Данный способ проектирования предполагает разделение больших таблиц на более мелкие и установление связей между ними. Рассмотрим основные нормальные формы [1]:

1. Первая нормальная форма (1НФ)
2. Вторая нормальная форма (2НФ)
3. Третья нормальная форма (3НФ)
4. Нормальная форма Бойса-Кодда (БКНФ)
5. Четвертая нормальная форма (4НФ)
6. Пятая нормальная форма (5НФ)
7. Шестая нормальная форма (6НФ)

Спроектировать базу данных, которая удовлетворяет нормальным формам, может помочь метод ER-моделирования. С ее помощью можно выделить ключевые сущности и связи между ними. Сущность – это объект предметной области, информация о котором должна храниться в базе данных. Атрибуты – это характеристики сущности.

Основная особенность такого метода заключается в том, что одну и ту же предметную область каждый проектировщик может представить в виде различных наборов сущностей, связей и атрибутов. Пример ER-модели образования на расстоянии представлен на рисунке 1.

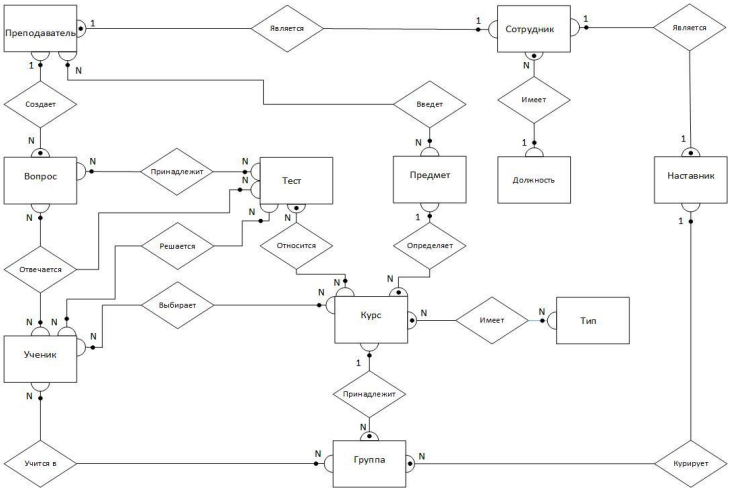


Рисунок 1 – ER – модель

Используя такую модель, можно спроектировать базу данных. Каждая сущность будет преобразована в виде таблиц с стандартным первичным ключом, а связи из ER-модели будут преобразованы в соответствии с правилами в различные таблицы, первичными ключами в которых может быть составной ключ или другие на усмотрение программиста и правил проектирования. Продемонстрируем на рисунке 2 схему базы данных, спроектированную по БКНФ и ER-модели.

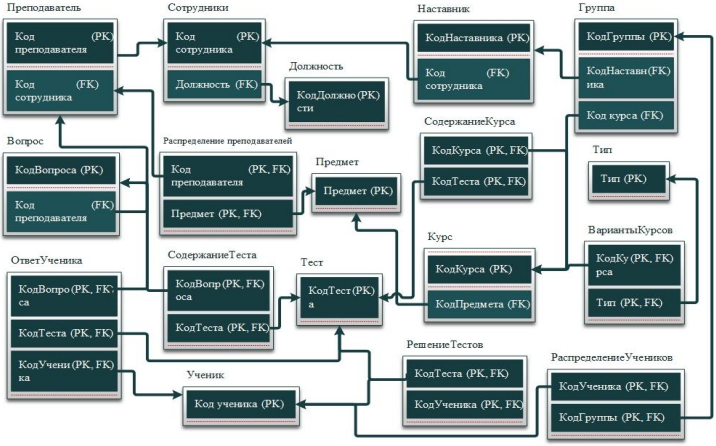


Рисунок 2 – Схема базы данных

Проверим соответствие одного из отношений к нормальным формам: Отношение «Сотрудник» (Код Сотрудника, ФИО, Телефон, Паспорт, Дата рождения, Должность, Дата найма на работу) находится в 1НФ, так как на пересечении столбца и строки находится только 1 значение. Отношение находится в 2НФ, так как оно находится в 1НФ и все не ключевые атрибуты функционально полно зависят от первичного ключа. Отношение находится в 3НФ, так как оно находится в 2НФ и в нём нет транзитивных зависимостей. Отношение находится в БКНФ, так как оно находится в 3НФ и детерминанты всех функциональных зависимостей являются потенциальными ключами (рисунок 2).

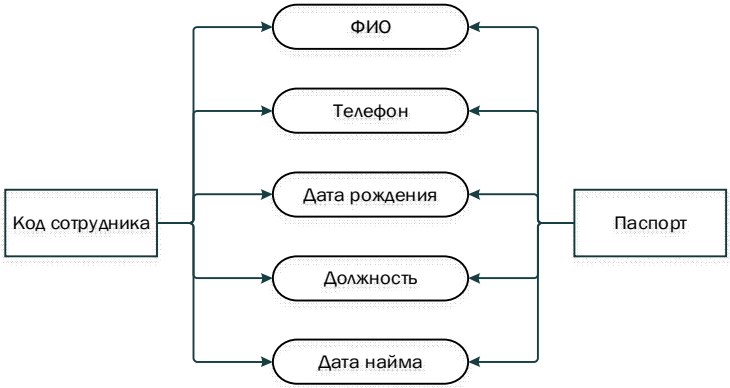


Рисунок 3 – Атрибуты отношения Сотрудник

Преимуществами такого проектирования являются:

1. Отсутствие противоречивых данных.
2. Уменьшение избыточности.
3. Повышение производительности запросов.
4. Уменьшение риска появления аномалий.
5. Обеспечение безопасности данных.

Недостатками такого подхода могут быть:

1. Увеличение количества составных ключей.
2. Потери производительности при выполнении сложных запросов, таких как объединение большого количества таблиц.
3. Сложности с реализацией

Оптимизация внедрением индексов в базу данных

Рассмотренный выше метод оптимизации возможно реализовать только на этапе проектирования, но во время создания БД перед нами также открываются возможность ее улучшения.

Рассмотрим оптимизацию путем индексации базы данных. Индексы представляют из себя указатели на строки в таблицах, они подходят для ускорения поиска информации. Рассмотрим пример оптимизации запроса к базе данных:

Пусть необходимо выгрузить информацию о сотрудниках с почтой на букву «V» и сроком работы более 3 лет:

Листинг 1. Запрос выгрузки информации о Сотрудниках

Данный запрос будет проверять каждую строку таблицы

```
SELECT *

FROM Сотрудники

WHERE Почта LIKE 'V%' AND DATEDIFF(NOW(),
ДатаНаймаНаРаботу) > 1095
```

«Сотрудники» на соответствие заданному условию. Его выполнение можно оптимизировать путем введения индекса для столбца с почтовыми адресами. Например, если столбец "почта" будет проиндексирован, то запрос будет выполняться более эффективно по следующему принципу:

1. Поиск среди индексов, почту на букву V
2. Проверка условия на срок работы

Таким образом, можем заметить, что большая часть данных о Сотрудника, которая нам не требуется даже не будет рассмотрена, это гораздо сокращает выполнение запроса.

Для создания индексов в языке SQL предусмотрен следующий синтаксис:

Листинг 2. Создание индекса

Таким образом, индексация значительно ускоряет выполнение

```
CREATE UNIQUE INDEX index_email

              ON Сотрудники (Почта)
```

запросов, но также имеет существенный недостаток. Индекс представляет собой отсортированную таблицу, т.е. для использования необходимо больше памяти, а также все изменения, происходящие в таблицы, должны производится в таблице индексов, что требует время на выполнения этих операций.

В результате использования такой оптимизации требуется значительный опыт работы с запросами, поскольку при неумелом

использовании базы данных могут возникнуть длительные задержки при обновлении и удалении данных.

Оптимизация за счет внедрения кэширования

Кэширование данных – это технология, которая снижает нагрузку на сервер. Ее работа заключается в сохранение динамического контента или прочей информации из базы данных, она размещается в памяти компьютера в виде текстового файла. Дальнейшая повторная генерация страницы будет выполнена быстрее так как запрос к БД не будет занимать время, а информация возьмется из кэша.

Такие данные определяются параметром актуальности кэша, т.е. временем, которое прошло от записи данных в файл. Некоторые данные никогда не изменяются потому их обновление не имеет смысла, но динамическая информация должна заносится в кэш с определённой периодичностью.

С помощью средств языка PHP, реализуем простой вариант кэширование данных. Функция для записи данных представим в листинге 3.

Листинг 3. Функция для записи кэша

```
Define (‘CACHE_FILE’, dirname(_FILE_). ‘/cache’);

Function CreateCache ($ObjectFile, $dataCache) {

      If ($f = fopen (CACHE_FILE. ‘//’. $ObjectFile,
‘w’)) {

            Fwrite ($f, serialize($dataCache));

            Fclose ($f);}

      } else {return False;}}
```

Использование кэш файла представляет из себя более сложную функции, так как необходимо проверить наличие файла и его актуальность, представим реализацию такой функции в листинге 4.

Листинг 4. Функция для выгрузки кэша

```
Function Use_Cache_in_Load ($ObjectFile,
$CurrentTime) {

  $file_name = CACHE_FILE. ‘/’. $ObjectFile;

    If (file_exists($file_name)) {

      If ($CurrentTime)

          If ((time ()
filemtime($file_name))>=$CurrentTime{

                    Return False;}}

     If ($f = fopen ($file_name, ‘r’)) {

              $temp = fread ($f, filesize
($file_name));

              Fclose ($f)

              Return (unserialize ($temp))}

       Else {return False}}

       Else {return False}}
```

Данный способ кэширования можно улучшить путем добавления некоторых функций:

1. Функция сжатия данных перед кэшированием и распаковка при выгрузке это уменьшит затраты на хранение кэша.

2. Функция удаления «мусора» из кэша это приведет к оптимальному размеру файла.

Рассмотрим пример, в котором необходимо сохранять данные о сотрудниках в кэш, с интервалом времени 20 минут, для наглядности оптимизации работы выберем не оптимальный запрос к базе данных, который осуществляет большую выборку.

Реализация такого примера с использованием вышеперечисленных функций опишем в листинге 5.

Листинг 5. Пример работы с кэшем

```
$query_select="SELECT * FROM Сотрудники";
$file=md5($query_select);
$res= Use_Cache_in_Load ($file, 20*60);
if ($res===false) {
    $sql_result=mysql_query($query_select);
    $res=array();
    while ($temp=mysql_fetch_array($sql_result)) {
   $res[]=$temp;
  }
  CreateCache ($file,$res);
}
// Операции с данными кэша
```

Таким образом, мы получили код, который сохраняет результат запроса в файл с проверкой актуальности кэша. Результаты можно использовать для выгрузки на сайт и в случае утери актуальности этой информации обновлять выполнение запроса. Данные, которые в ходе работы никак не изменяются могут оставаться в кэше неограниченное количество времени.

Правила оптимизированных запросов

Не смотря, на множество способов ускорить работу запросов, их написание также должно быть простым и оптимизированным. Рассмотрим некоторые основные правила написания такого кода [2]:

1. Избегать применение оператора DISTINCT и ORDER BY, данные операторы сильно увеличивают время запросов их использование лучше вынести за рамки обращения к БД.

2. Выбирать только необходимые столбцы в SELECT-запросе, лишние данные, которые возвращает нам БД, также увеличивают время.

3. Правильное использование JOIN, для такой операции лучше всего использовать равенство ключей, избегать связи N:N так как объединение такие сущностей занимает много времени.

4. Выбор правильных операторов для работы с данными, например оператора IN и EXISTS выполняют почти одинаковые функции, но разница между ними в том, что EXISTS, закончит выполнение запроса, как только найдет нужный элемент, а IN переберет все значения, что гораздо затруднит выполнение запроса.

5. Выбирать наиболее подходящий тип JOIN, при неправильном выборе данные будут представлять из себя декартово произведение строк таблиц.

В ходе нашего обзора мы изучили различные методы оптимизации производительности веб-сайтов. Необходимо отметить, что эффективная работа с платформами дистанционного образования является ключевой задачей, поскольку данная сфера характеризуется постоянным обновлением информации в базах данных, что подчеркивает важность вопроса оптимизации запросов в данном контексте.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Волк, В.К. Базы данных. Проектирование, программирование, управление и администрирование / В. К. Волк. – 4-е изд., стер. – Санкт-Петербург: Лань, 2023. – 244 с.

2. Скляр, А.Я. Системы управления данными: учебное пособие / А. Я. Скляр, А. А. Высоцкая, А. А. Горячев. – Москва: РТУ МИРЭА, 2022. – 163 с.

УДК 004.72

ПЕРЕПЕЛКИН Д.А., АНИСИМОВ К.В.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

АНАЛИЗ СОВРЕМЕННЫХ ТЕХНОЛОГИЙ ИНТЕРНЕТА ВЕЩЕЙ

Рассматриваются современные технологии в области Интернета вещей. Приводится обобщенная архитектура сети Интернета вещей, анализируются достоинства и недостатки различных способов подключения устройств Интернета вещей, рассматриваются наиболее популярные прикладные протоколы Интернета вещей.

Введение

Интернет вещей (Internet of Things, IoT) [1] – это концепция и совокупность технологий для организации сети передачи данных между множеством автономных устройств (вещей) с целью сбора данных о состоянии внешней среды и последующей автоматизации различных процессов на основе этих данных. В настоящее время IoT технологии стремительно развиваются [2 – 4]. Согласно исследованиям компании DemandSage [5] в 2023 году насчитывалось 15,14 млрд. IoT устройств. Прогнозируется, что это количество будет увеличиваться более чем на 2 млрд в год (рис. 1).

Рисунок 1 – Тенденция увеличения количества IoT устройств

Архитектура Интернета вещей

Технологии IoT разделяются на два сектора: потребительский и промышленный. К потребительскому сектору относятся бытовые устройства, используемые человеком в повседневной жизни: «умные» часы, розетки, телевизоры, чайники, роботы пылесосы и т.д. К потребительскому IoT также относят смартфоны.

К промышленному IoT относят технологии, применяемые для оптимизации бизнес-процессов: автоматизация производства, мониторинг состояния товаров, контроль качества продукции, мониторинг местоположения транспорта и т.д.

Архитектуру IoT можно представить в виде следующих трех уровней (рис. 2): уровень IoT устройств, уровень передачи данных, уровень приложений. В зависимости от конкретных решений эта архитектура может разбиваться на большее количество уровней [6].

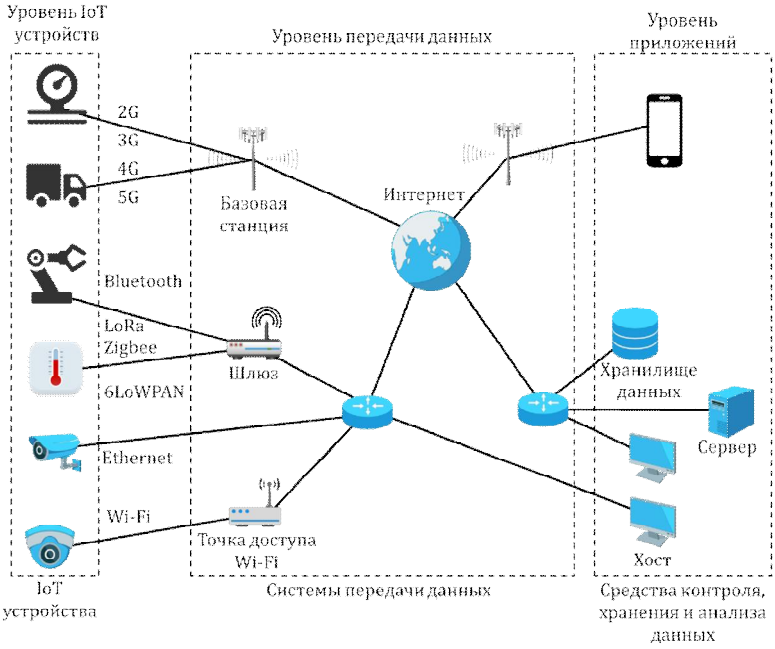


Рисунок 2 – Архитектура Интернета вещей

IoT устройства, расположенные на соответствующем уровне, подключаются к устройствам уровня передачи данных посредством различных беспроводных или проводных технологий (табл. 1). На уровне передачи данных располагаются устройства, позволяющие передавать данные между IoT устройствами в локальных или глобальных сетях: шлюзы, коммутаторы, маршрутизаторы, базовые станции и т.д. На уровне приложений расположены устройства, позволяющие хранить, обрабатывать, анализировать или отображать полученные данные с последующим прикладным применением.

Таблица 1 – Способы подключения IoT устройств

Способ связи	**Технология**	**Применение**
Беспроводной	Wi-Fi	Подключение мобильных или стационарных устройств в пределах локальной сети
	Bluetooth	Подключение мобильных или стационарных устройств в пределах частной сети
	Zigbee	
	Z-Wave	
	6LoWPAN	

	LoRa/LoRaWAN	Подключение мобильных или стационарных устройств с большим сроком автономной работы
	NB-IoT	Подключение стационарных устройств с большим сроком автономной работы к глобальной сети
Проводной	Ethernet	Подключение стационарных устройств к локальной сети, подключение локальных сенсорных сетей к глобальной сети
	RS-485	Подключение стационарных промышленных устройств
	RS-232	
	RS-422	
	EtherCAT	

Способы подключения IoT устройств

Wi-Fi

Одной из самых распространенных технологий подключения IoT устройств к локальной сети является Wi-Fi, работа которого основана на стандартах IEEE 802.11. Особенно распространен этот способ подключения в секторе потребительского IoT. Преимуществом такого способа подключения является то, что устройство, подключенное к Wi-Fi, сразу добавляется в локальную сеть и, при условии настроенного доступа в Интернет, имеет возможность принимать или получать данные из глобальной сети. Однако таким устройствам для информационного обмена необходимо тратить достаточно большое количество энергии, что накладывает ограничение на автономность устройств. В зависимости от функций устройств, емкости аккумуляторной батареи и частоты передачи данных, устройства, подключаемые к Wi-Fi, имеют длительность автономной работы несколько часов.

Bluetooth

Bluetooth, основанный на стандарте IEEE 802.15.1, также является одной из распространенных технологий обмена данными между IoT устройствами. Говоря о Bluetooth в контексте IoT часто имеют ввиду технологию Bluetooth с низким энергопотреблением Bluetooth Low Energy т.к. низкое энергопотребление позволяет увеличить время автономной работы, что является важным свойством IoT устройств. В отличии от Wi-Fi Bluetooth позволяет обмениваться

информацией устройствам, расположенным на небольшом удалении друг от друга – в пределах персональной сети. Также устройства, подключенные по Bluetooth, изначально не имеют доступа к локальной или глобальной сети. Для обеспечения таких функций необходим шлюз, роль которого, обычно, играет компьютер или смартфон.

Zigbee

Технология Zigbee, основанная на стандарте IEEE 802.15.4, позволяет строить ячеистые самоорганизующиеся сети устройств с низким энергопотреблением [7]. Различают три вида Zigbee устройств: координаторы, маршрутизаторы и конечные устройства. Координатор выполняет функции центра управления сетью. К его основным функциям относятся: выбор политики безопасности, разрешение или запрет подключения новых устройств, измерение уровня помех, определение частотного канала связи для всех устройств. Маршрутизаторы являются стационарными устройствами, которые отвечают за передачу данных межу устройствами. Используя специальные алгоритмы, они решают задачу определения наиболее оптимальных маршрутов межу конечными устройствами. Конечные устройства подключаются к маршрутизаторам или координатору и не участвуют в ретрансляции приходящего трафика. Эти устройства большую часть времени находятся в режиме «сна» и выходят из него только для получения или передачи данных. Виды топологий сетей Zigbee устройств представлены на рис. 3.

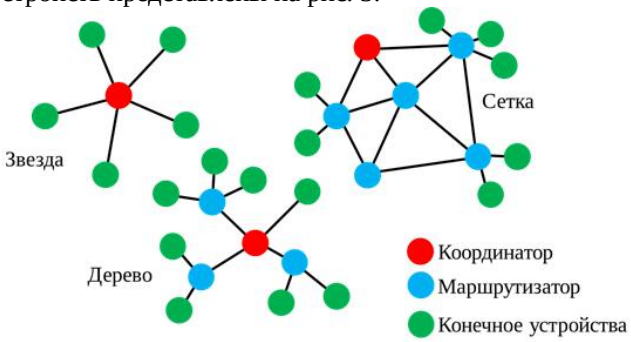


Рисунок 3 – Виды топологий сетей Zigbee устройств

Z-Wave

Z-Wave является запатентованной технологией беспроводной передачи данных IoT устройств, затрагивающей уровни модели OSI с сетевого до приложений [8]. Физический и канальный уровни определяются стандартом ITU-T G9959. Все устройства в сети Z-Wave объединяются в домены, которые могут частично перекрывать

радиочастотные диапазоны друг друга. Каждый домен имеет уникальный идентификатор HomeID. В сети Z-Wave выделяют следующие типы устройств: контроллеры и конечные устройства. Каждое устройство идентифицируется с помощью уникального NodeID.

Контроллеры отвечают за настройку и обслуживание сети. Они могут добавлять или удалять узлы. Также контроллеры знают топологию сети, что позволяет им определять возможные маршруты между любыми двумя узлами сети. Контролеры могут обмениваться данными о топологии сети друг с другом.

Контроллеры разделяются на первичные и вторичные. Первичный контроллер может быть только один в сети. Именно он может добавлять или удалять устройства из сети. Все остальные контроллеры – вторичные. Любой один контроллер в сети может быть SUC-контроллером - контроллером статических обновлений, который отвечает за обновление информации о топологии сети. Если SUC-контроллером является первичный контроллер, то он называется SUC ID Server.

Конечные устройства могут быть добавлены или удалены из сети только первичным контроллером. Они не вычисляют маршруты, а полагаются только на информацию о маршруте, предоставленную контроллерами. Конечные устройства могут отправлять команды другим узлам. Это позволяет узлам сети взаимодействовать друг с другом, даже если они находятся вне зоны прямой связи.

Конечные Z-Wave устройства могут работать в одном из трех режимов: «Всегда слушаю», «Часто слушаю» и «Не слушаю». Устройства, работающие в режиме «Часто слушаю» большее время находятся в выключенном состоянии, и включаются через некоторые интервалы времени для передачи и приема данных. Также они могут быть пробуждены устройствами, работающими в режиме «Всегда слушаю». Устройства в режиме «Не слушаю» не могут быть пробуждены другими устройствами.

Пример топологий сети Z-Wave представлен на рис. 4.

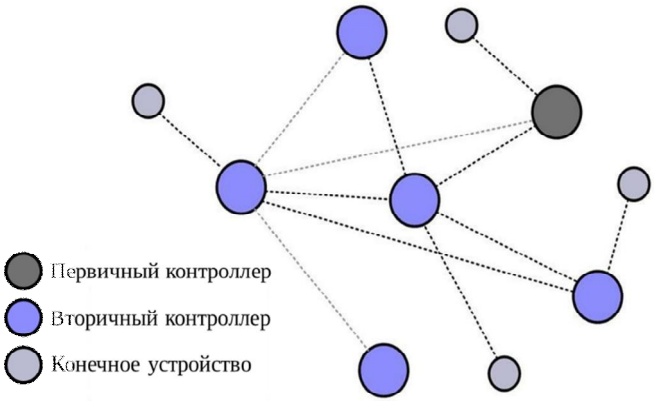


Рисунок 4 – Топология сети Z-Wave

LoRa/LoRaWAN

LoRa – это совокупность технологий для обеспечения беспроводной передачи данных между маломощными IoT устройствами, находящимися на достаточно большом удалении друг от друга [9]. Для передачи информации на физическом уровне используется специальный вид модуляции, называемый LoRa. На канальном уровне функционирует протокол LoRaWAN.

Устройства сети LoRa разделяются на шлюзы и конечные устройства, которые собой топологию «звезда». Конечные устройства делятся на три класса: A, B и C.

Связь с устройством класса A всегда инициируется самими устройством и является асинхронной. Каждая передача по восходящей линии связи может быть отправлена в любое время, за ней следуют два коротких окна по нисходящей линии связи, дающие возможность передачи команд управления. Все остальное время, когда устройство не отправляет данные, оно может находиться в режиме энергосбережения. Это позволяет устройствам класс A быть наиболее энергоэффективными.

В устройствах класса B в дополнение к окнам приема, инициируемым устройством класса A, устройства класса B синхронизируются с сетью с помощью периодических маяков и открывают «слоты пинга» нисходящей линии связи в запланированное время. Это обеспечивает возможность отправлять сообщения по нисходящей линии связи с определенной задержкой, но за счет некоторого дополнительного энергопотребления конечного устройства. Время ожидания программируется до 128 секунд.

В устройствах класса C приемник нисходящей линии связи остается включенным все время, когда устройство не отправляет данные. Исходя из этого, сетевой сервер может инициировать передачу по нисходящей линии в любое время при условии, что приемник конечного устройства включен. Класс C подходит для устройств, для которых доступно постоянное питание.

6LoWPAN

Сети IoT устройств делятся на IP сети и не-IP сети [10]. К IP сетям относятся сети Wi-Fi, а не не-IP – Bluetooth, Zigbee, Z-Wave, LoRa. IP сети удобны тем, что подключенные устройства сразу имеют доступ к локальной или глобальной сети. Также для организации IP сетей можно применять повсеместно-используемое сетевое оборудование. Стандарт 6LoWPAN был специально разработан с целью подключения маломощных устройств с низким энергопотреблением к IP сетям. Каждое устройство, подключаемое с помощью 6LoWPAN, имеет 16-битный короткий адрес, а также 64-битный расширенный адрес. Для маршрутизации в таких сетях используется протокол маршрутизации сетей с низким потреблением и потерями RPL (IPv6 Routing Protocol for Low Power and Lossy Networks). RPL представляет топологию сети как направленный ациклический граф, разделенный на один или несколько подграфов. Сеть управляется одним или несколькими центральными устройствами, например, пограничными 6LoWPAN маршрутизаторами.

NB-IoT

Широкая доступность и большая площадь сетевого покрытия делают технологии сотовых сетей очень привлекательными для их применения в IoT сфере. Однако, как и большинство других технологий, изначально не предназначавшихся для Интернета вещей, технологии передачи данных в сотовых сетях, не являются оптимальным с точки зрения энергоэффективности. NB-IoT – это технология энергоэффективного информационного обмена на основе стандартов передачи данных в сотовых сетях LTE [11].

Сравнение беспроводных IoT технологий представлено в табл. 2 и 3.

Таблица 2 – Сравнение беспроводных IoT технологий

Технология	**Частоты работы**	**Дальность связи**	**Максимальная скорость передачи данных**
Wi-Fi	2,4 ГГц 5 ГГц	до 300 м	более 100 Мбит/с
Bluetooth Low Enegry	2,4 ГГц	до 100 м	1 Мбит/с
Zigbee	868 МГц 915 МГц 2,4 ГГц (в России)	до 100 м	250 Кбит/с
Z-Wave	под 1 ГГц 869 МГц (в России)	до 100 м	100 Кбит/с
6LoWPAN	2,4 ГГц (в России) 915 МГц 868 МГц	до 100 м	250 Кбит/с
LoRaWAN	433 МГц 868 МГц 2,4 ГГц 5,7 ГГц	до 15 км	50 Кб/с
NB-IoT	450 – 2700 МГц	до 15 км	100 Кбит/с

Таблица 3 – Сравнение беспроводных IoT технологий

Технология	**Размер сети**	**Поддержка IP-технологий**	**Возможность организации ячеистой сети**
Wi-Fi	Зависит от Wi-Fi роутера	+	-
Bluetooth	Зависит от реализации	-	-
Zigbee	≈ 65000	-	+
Z-Wave	232 устройства в домене	-	+
6LoWPAN	2^{64}	+	+
LoRaWAN	2^{25}	-	-
NB-IoT	Зависит от базовой станции	+	-

Одним из преимуществ NB-IoT является способность передавать данные даже при низком уровне сигнала от базовой станции. В последнее время в NB-IoT внедряется режим Non-IP. Устройства, работающие в этом режиме, не получают IP адреса, данные передаются с помощью протокола NIDD (Non-IP Data Delivery). За взаимодействие с глобальной сетью отвечает оборудование оператора связи. Такое решение позволяет уменьшить объем передаваемых данных за счет отсутствия потребности в установлении TCP-соединения и отсутствия IP-заголовка.

Проводные технологии подключений IoT устройств

Помимо беспроводных технологий подключения устройств, с которыми ассоциируется термин «Интернет вещей» немаловажную роль имеют технологии проводного подключения, в особенности Ethernet. Технологии промышленных сетей, такие как RS-485, RS-232, RS-422 используются для объединения сенсорных устройств в единую сеть. Однако устройства в таких сетях не взаимодействуют между собой напрямую, а выполняют команды от ведущего устройства. Для подключения таких сетей к локальным сетям необходимо использовать шлюзы.

Технологии Ethernet используется как для непосредственного подключения IoT устройств к локальной сети, так и для подключения локальных IoT сетей к глобальным. Технологии Ethernet широко применяются в промышленном IoT, так как позволяют обеспечить высокую надежность и безопасность передачи данных.

Прикладные IoT протоколы

Как было сказано ранее, многие стандартные решения не подходят для IoT устройств. Тоже самое касается и прикладных протоколов передачи данных в IP-сетях. Протокол HTTP не подходит для информационного обмена между устройствами, ограниченными в вычислительных способностях и зарядом аккумулятора. Для взаимодействия с такими устройствами было разработано множество более энергоэффективных и простых протоколов. Наиболее популярные из них представлены в табл. 4.

Таблица 4 – Прикладные протоколы передачи данных в IoT сетях

Протокол уровня приложений	**Транспортный протокол**	**Модель**
MQTT	TCP	Издатель – подписчик
CoAP	UDP	Клиент – сервер
AMQP	TCP/UDP	Издатель – подписчик Клиент – сервер

MQTT

MQTT – это прикладной протокол обмена сообщениями для IoT устройств. Он разработан для обеспечения легкого способа обмена сообщениями в модели «Издатель – подписчик». MQTT подходит для подключения удаленных устройств с небольшим объемом передаваемого трафика, которые функционируют в сети с очень низкой пропускной способностью.

MQTT устройства разделяются на издателей, которые отправляют данные, подписчиков, которые принимают данные, и брокеров, которые принимаю данные от издателей и отправляют подписчикам. Издатель, отправляя данные, помечает их идентификатором, называемым topic. Брокер принимает данные и отправляет тем подписчикам, которые предварительно были подписаны на этот topic.

MQTT выделяет три приоритета сообщений для обеспечения разного качества сервиса (Quality of Service, QoS): QoS 0, QoS 1 и QoS 2. Сообщения QoS 0 отправляются только один раз, их доставка не гарантируется. Сообщения QoS 1 могут отправляться получателю несколько раз через некоторые промежутки времени, пока он не отправит сообщение о получении. Однако получатель может принять не только оригинальное сообщение, но и его дубликаты. Сообщения QoS 2 могут быть приняты получателем только один раз. Если получатель не отправил ответ о получении сообщения, то отправитель отправляет его снова, но уже помечает его как дубликат.

Принцип работы протокола MQTT представлен на рис. 5.

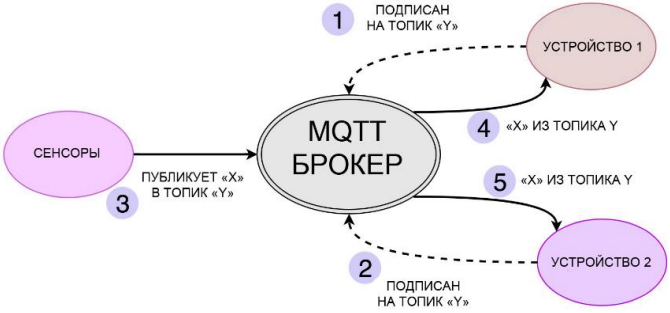


Рисунок 5 – Принцип работы протокола MQTT

CoAP

Другим прикладным IoT протоколом является CoAP. Он, работает на основе модели «Клиент – сервер», а устройства

обращаются к друг-другу с помощью стандартных HTTP методов. Для уменьшения количества вычислительных ресурсов, требуемых для работы, а также уменьшения энергопотребления, в качестве протокола транспортного уровня CoAP использует UDP. Несмотря на то, что UDP не гарантирует доставку данных, получатель может отправить подтверждение приема сообщения, тем самым оповестив отправителя об успешной передаче.

В CoAP можно выделить следующие устройства: клиент, сервер, посредник, сервер происхождения, и наблюдатель. Клиент, отправляет запрос на сервер, серверы выполняют запрос, приходящий от клиента. Посредники перенаправляют или изменяют запросы. Сервер происхождения – это сервер, на котором расположен запрашиваемый ресурс. Наблюдатель – это устройство, контролирующее состояние ресурса.

AMQP

Протокол AMQP способен работать как в модели «Издатель – подписчик», так и в модели «Клиент – сервер». В отличии от MQTT брокер сообщений AMQP разделен на две составляющие: точку обмена и очередей сообщений [12]. Издатели отправляют сообщения не напрямую в очередь, как в MQTT, а в точку обмена. Точка обмена маршрутизирует сообщение в необходимую очередь или очереди, если на тему этого сообщения подписаны несколько устройств. Сообщение остается в очереди до тех пор, пока не будет получено подписчиком. В качестве протокола канального уровня AMQP использует TCP. Также как и MQTT AMQP поддерживает три вида качества сервиса.

Заключение

В работе представлена обобщенная архитектура сети устройств Интернета вещей, состоящая из трех уровней: IoT устройств, систем передачи данных и средств контроля, хранения и анализа данных. Рассмотрены беспроводные и проводные способы объединения устройств Интернета вещей в единую сеть. Приведены их достоинства и недостатки. Рассмотрены наиболее популярные прикладные IoT протоколы и их особенности.

Работа выполнена при финансовой поддержке гранта Фонда содействия инновациям 19117ГУ/2023.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Введение в Интернет вещей: учебное пособие / Е.В. Глушак, А.В. Куприянов. – Самара: Издательство Самарского университета, 2023. – 104 с.: ил.

2. Перепелкин Д. А., Анисимов К. В. Разработка архитектуры и системы нечетких метрик каналов связи программно-конфигурируемой сети устройств Интернета вещей // Вестник Рязанского государственного радиотехнического университета. – 2022. – № 80. – С. 53-66. DOI 10.21667/1995-4565-2022-80-53-66.

3. Перепелкин Д. А., Анисимов К. В. Модифицированный алгоритм парных переходов в программно-конфигурируемых сетях на основе нечеткой логики // Вестник Рязанского государственного радиотехнического университета. – 2023. – № 86. – С. 54-62. DOI 10.21667/1995-4565-2023-86-54-62.

4. Перепелкин Д. А., Ткачев Д. Д. Разработка шлюза и облачной платформы программно-конфигурируемой сети устройств Интернета вещей // Вестник Рязанского государственного радиотехнического университета. – 2023. – № 84. – С. 88-98. DOI 10.21667/1995-4565-2023-84-88-98.

5. Rohit, Shewale 73 IoT Statistics on Market Growth, Usage & Trends (2024) / Shewale Rohit. — Текст: электронный // DemandSage: [сайт]. — URL: https://www.demandsage.com/internet-of-things-statistics/ (дата обращения: 31.03.2024).

6. Roberto Siagri Основа архитектуры «Интернета вещей» // Control Engineering Россия. - 2017. - №5 (71). - с. 52-55.

7. Pan M.S., Tseng Y.C. ZigBee and their applications //Sensor networks and configuration: Fundamentals, standards, platforms, and applications. – Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. – С. 349-368.

8. Danbatta S. J., Varol A. Comparison of Zigbee, Z-Wave, Wi-Fi, and bluetooth wireless technologies used in home automation //2019 7th International Symposium on Digital Forensics and Security (ISDFS). – IEEE, 2019. – С. 1-5.

9. Sundaram J. P. S., Du W., Zhao Z. A survey on LoRa networking: Research problems, current solutions, and open issues //IEEE Communications Surveys & Tutorials. – 2019. – Т. 22. – №. 1. – С. 371-388.

10. Yang, Zengxu and C. Hwa Chang. “6LoWPAN Overview and Implementations.” European Conference/Workshop on Wireless Sensor Networks (2019).

11. Ratasuk R. et al. NB-IoT system for M2M communication //2016 IEEE wireless communications and networking conference. – IEEE, 2016. – С. 1-5.

12. Якупов Д.Р. Обзор и сравнение протоколов интернета вещей: MQTT И AMQP // International Journal of Open Information Technologies. 2022. №9. URL: https://cyberleninka.ru/article/n/obzor-i-sravnenie-protokolov-interneta-veschey-mqtt-i-amqp (дата обращения: 31.03.2024).

УДК 004.65

ПЕЧЕНИН И.В.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

АНАЛИТИКА БОЛЬШИХ ДАННЫХ В СРЕДАХ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ

В данной статье рассматривается роль и значение аналитики больших данных в контексте облачных вычислений. Облачные вычисления предоставляют удобную среду для обработки, хранения и анализа больших объемов данных. Аналитика больших данных в облачных средах играет ключевую роль в извлечении ценной информации из данных и принятии обоснованных бизнес-решений. В статье также рассматриваются основные методы аналитики больших данных в облачных средах, проблемы и вызовы, а также потенциальные преимущества и перспективы развития.

Введение

С постоянным ростом объемов данных, с которыми сталкиваются организации, возникает потребность в эффективных методах их обработки и анализа. Облачные вычисления предоставляют среду, позволяющую управлять большими объемами данных гораздо эффективнее, чем традиционные вычислительные средства. Одним из ключевых аспектов облачных вычислений является возможность проведения аналитики больших данных прямо в облаке. Это открывает новые возможности для бизнеса в области анализа данных, прогнозирования, выявления тенденций и принятия решений на основе данных.

Методы аналитики больших данных в облачных средах

Аналитика больших данных в облачных средах включает в себя широкий спектр методов и инструментов для обработки и анализа данных. Среди основных методов можно выделить:

Хранение данных: Облачные сервисы предоставляют различные опции для хранения данных, включая хранилища данных с высокой отказоустойчивостью и масштабируемостью, такие как Amazon S3, Google Cloud Storage и Microsoft Azure Blob Storage.

Обработка данных: для обработки данных в облачных средах широко используются такие технологии как Apache Hadoop, Apache Spark, которые позволяют параллельно обрабатывать большие объемы данных.

Анализ данных: для анализа данных в облачных средах применяются различные методы машинного обучения, статистические методы, а также инструменты для визуализации данных, такие как Tableau, Power BI.

Масштабирование: Одним из основных преимуществ облачных вычислений является возможность масштабирования вычислительных ресурсов в зависимости от потребностей, что позволяет обрабатывать данные любого объема.

Преимущества, вызовы и проблемы

Облачные сервисы позволяют мгновенно масштабировать вычислительные ресурсы в зависимости от потребностей, что обеспечивает гибкость при обработке и анализе данных, а также способствуют снижению затрат на инфраструктуру и оборудование, так как предоставляется возможность оплаты только за фактически использованные ресурсы.

Облачные сервисы обеспечивают быстрый доступ к данным из любой точки мира, что позволяет сократить время на обработку и анализ данных.

Необходимо отметить, что аналитика больших данных в облачных средах также сталкивается с рядом вызовов и проблем, включая:

1.Конфиденциальность и безопасность данных

Перенос данных в облако может вызвать опасения относительно конфиденциальности и безопасности данных, особенно в чувствительных отраслях, таких как здравоохранение и финансы.

2.Интеграция существующих систем

Интеграция облачных решений с существующими системами и инфраструктурой может быть сложной задачей и требует дополнительных усилий и ресурсов.

3.Управление данными

С увеличением объемов данных возрастает сложность управления данными, и требуется разработка эффективных стратегий и методов для их управления.

Перспективы развития

С развитием технологий облачных вычислений и аналитики больших данных ожидается дальнейшее расширение возможностей и улучшение производительности и эффективности анализа данных.

Предполагается, что в будущем будут разработаны новые методы и инструменты для обработки и анализа данных в облачных средах, что позволит организациям еще эффективнее использовать данные для принятия стратегических решений.

Заключение

В настоящей статье была рассмотрена значимость и важность аналитики больших данных в средах облачных вычислений. Облачные вычисления предоставляют мощные инструменты для обработки, хранения и анализа огромных объемов данных, что позволяет организациям эффективно управлять информацией и принимать обоснованные решения на основе данных.

Методы аналитики больших данных в облачных средах, такие как хранение данных, обработка, анализ и визуализация, предоставляют широкий спектр возможностей для извлечения ценной информации из данных и прогнозирования тенденций. Гибкость и масштабируемость облачных сервисов, а также их экономическая эффективность делают их привлекательным выбором для организаций, стремящихся оптимизировать процессы анализа данных.

Однако необходимо учитывать вызовы и проблемы, такие как конфиденциальность и безопасность данных, интеграция с существующими системами и управление данными. Развитие технологий облачных вычислений и аналитики больших данных открывает новые перспективы для более эффективного использования данных и принятия более обоснованных и информированных решений в различных областях бизнеса и науки.

Таким образом, аналитика больших данных в средах облачных вычислений играет ключевую роль в современном информационном обществе, обеспечивая организациям возможность извлечения ценной информации из данных и принятия обоснованных стратегических решений для достижения успеха и конкурентных преимуществ.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Агревалио, А. Большие данные и облачные вычисления: технологии и методы анализа / А. Агревалио. – М.: ДМК Пресс, 2019. – 256 с.

2. Васильев, Д. А. Облачные вычисления и аналитика данных: современные тенденции и перспективы / Д. А. Васильев // Информационные технологии и вычислительные системы. – 2018. – № 3. – С. 45-52.

3. Гаврилов, В. А. Большие данные: технологии и методы анализа / В. А. Гаврилов, Е. Н. Соколова. – М.: ДМК Пресс, 2020. – 368 с.

4. Давыдов, А. С. Аналитика данных в облачных вычислениях / А. С. Давыдов // Облачные вычисления и цифровые технологии. – 2017. – № 2. – С. 31-36.

5. Клименко, С. Облачные технологии и аналитика больших данных / С. Клименко // Высокие технологии. – 2019. – № 5. – С. 18-22.

УДК 004.65

ПЕЧЕНИН И.В.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ИСПОЛЬЗОВАНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ОПТИМИЗАЦИИ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

В данной статье мы исследуем применение генетических алгоритмов для оптимизации вычислительных процессов. Мы обсуждаем основные концепции генетических алгоритмов, их механизмы и то, как их можно адаптировать и применить для повышения эффективности вычислительных процессов. Кроме того, мы рассматриваем примеры и подчеркиваем потенциальную пользу и проблемы, связанные с интеграцией генетических алгоритмов в вычислительные рамки.

Введение

Вычислительные процессы играют ключевую роль в различных областях, начиная от научных симуляций и заканчивая промышленной обработкой данных. Повышениеэффективности этих процессов является важным для улучшения общей производительности и снижения расхода ресурсов. Традиционные методы оптимизации часто сталкиваются с проблемами в обработке сложных многомерных пространств поиска и нелинейных целевых функций эффективно. Генетические алгоритмы (ГА) предлагают многообещающую альтернативу, имитируя принципы естественного отбора для поиска решений и определения оптимальных или приближенно оптимальных решений. В данной статье исследуется использование генетических алгоритмов для оптимизации вычислительных процессов, проясняя их механизмы, применение и потенциальные преимущества.

Основы генетических алгоритмов и их применение в вычислительных процессах

Гибкость генетических алгоритмов делает их подходящими для оптимизации различных вычислительных процессов, включая:

Настройка параметров: Генетические алгоритмы могут оптимизировать параметры сложных вычислительных моделей или алгоритмов для улучшения их производительности. Это включает настройку гиперпараметров моделей машинного обучения, оптимизацию численных симуляций и настройку алгоритмов для эффективного использования ресурсов.

Планирование и распределение ресурсов: В вычислительных средах с несколькими задачами и ресурсами генетические алгоритмы могут использоваться для оптимизации планирования задач и распределения ресурсов для максимизации пропускной способности, минимизации задержек или снижения энергопотребления.

Выбор признаков и снижение размерности: Генетические алгоритмы могут помочь в выборе релевантных признаков и уменьшении размерности данных, тем самым повышая эффективность вычислительных задач, таких как классификация, кластеризация и регрессия.

Оптимизация сети: Генетические алгоритмы были применены для оптимизации топологии сети, протоколов маршрутизации и распределения ресурсов в сетях связи, что привело к повышению пропускной способности, надежности и масштабируемости.

Алгоритмический дизайн: Генетические алгоритмы могут помочь в разработке и оптимизации алгоритмов для конкретных вычислительных задач, используя эволюционные принципы для изучения различных областей решений и выявления эффективных алгоритмов.

Тематические исследования и практические реализации

Несколько исследований реальных приложений демонстрируют эффективность генетических алгоритмов в оптимизации вычислительных процессов:

Генетические алгоритмы были использованы для оптимизации гиперпараметров моделей глубокого обучения, таких как количество слоев, скорость обучения и функции активации, что привело к повышению производительности и скорости конвергенции. А также использовались для оптимизации планирования заданий и распределения ресурсов в средах облачных вычислений с учетом таких факторов, как зависимости задач, доступность ресурсов и приоритеты пользователей, чтобы максимизировать пропускную способность системы и минимизировать время отклика.

ГА использовались для разработки цифровых схем с определенной функциональностью или ограничениями производительности, что позволяет автоматизировать проектирование

электронных систем с оптимизированной производительностью и сокращенным временем проектирования.

Проблемы и направления на будущее

Несмотря на то, что генетические алгоритмы обладают значительным потенциалом для оптимизации вычислительных процессов, существует несколько проблем, в том числе:

Вычислительная сложность: Генетические алгоритмы могут требовать значительных вычислительных ресурсов, особенно для задач многомерной оптимизации или крупномасштабных популяций.

Преждевременная конвергенция: Генетические алгоритмы могут преждевременно сходиться к неоптимальным решениям или застаиваться в локальных оптимумах, что требует изучения различных стратегий поиска и механизмов поддержания разнообразия.

Чувствительность к параметрам: Производительность генетических алгоритмов может быть чувствительна к настройкам параметров, таким как размер популяции, частота скрещиваний и мутаций, а также механизмы отбора, требующие тщательной настройки и экспериментов.

Будущие направления исследований включают разработку гибридных алгоритмов оптимизации, сочетающих генетические алгоритмы с другими методами метаэвристического или машинного обучения, исследование параллельных и распределенных генетических алгоритмов для масштабируемой оптимизации, а также изучение многоцелевых и ограниченных подходов к оптимизации в рамках генетических алгоритмов.

Заключение

В заключение, генетические алгоритмы предлагают надежный и универсальный подход для оптимизации вычислительных процессов в различных областях. Имитируя принципы естественного отбора и генетической изменчивости, генетические алгоритмы могут эффективно исследовать сложные пространства решений и выявлять оптимальные или близкие к оптимальным решения различных задач оптимизации.

С помощью тематических исследований и практических реализаций мы продемонстрировали эффективность генетических алгоритмов в повышении эффективности, масштабируемости и быстродействия вычислительных процессов. Несмотря на существующие проблемы, генетические алгоритмы по-прежнему остаются ценным инструментом для решения задач оптимизации в вычислительных областях, с многообещающими направлениями для будущих исследований и разработок.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1.Гольцова, М. Ю. Генетические алгоритмы и искусственные нейронные сети в задачах оптимизации и управления / М.Ю. Гольцова. – М.: Интеллект. – 2008. – С. 89-96.

2. Голдберг, Д.Э. Генетические алгоритмы в поиске, оптимизации и машинном обучении / Д. Э Голдберг. – М.: ДМК Пресс. – 2016.– С. 157-165.

3. Хафт, Р. Л.Хафт, С. Е. Практические генетические алгоритмы / Р. Л Хафт, С. Е Хафт. – М.: Техносфера.. – 2008. – С. 42-49.

4. Дэвис, Л. Руководство по генетическим алгоритмам / Л. Девис. – СПб.: Питер. – 2007.– С. 27-33.

УДК 004.021

ПОЛОВИНКИН А.В.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ РАСПАРАЛЛЕЛИВАНИЯ В ГЕНЕТИЧЕСКИХ АЛГОРИТМАХ

Рассматриваются способ эффективного нахождения оптимального значения функции с использование генетического алгоритма и технологии распараллеливания OpenMP.

Обычно, задача оптимизации включает в себя нахождение экстремума целевой функции, т.е. ее минимума или максимума. Если говорить простыми словами, то основной задачей оптимизации является нахождение таких входных параметров, при которых наблюдается ее минимум или максимум.

Одним из методов решения задач оптимизации является применение генетических алгоритмов.

Генетический алгоритм основан на принципах естественного отбора и относится к стохастическим методам. Одним из его наиболее важных понятий является функция приспособленности или функция полезности (fitness function). Данная функция представляет собой меру приспособленности данной особи в популяции. Она оказывает очень сильное влияние на работу алгоритма и поэтому должна иметь точное и корректное определение. В задачах оптимизации, функция приспособленности, как правило, оптимизируется и является целевой функцией [1].

Классический генетический алгоритм обычно имеет следующие шаги:

1) Инициализация или выбор исходной популяции хромосом.
2) Оценка приспособленности особей в популяции.
3) Проверка условий остановки алгоритма.
4) Селекция особей.
5) Применение генетических операторов.
6) Создание новой популяции.
7) Выбор наилучшей хромосомы

Но, как и в других алгоритмах оптимизации, данный метод может столкнуться с проблемой того, что нельзя точно сказать, достигнут ли экстремум функции или нет. В качестве инструмента, который помогает нейтрализовать этот недостаток, является мутация. Данный механизм позволяет нейтрализовать проблему однообразия популяции и решает проблему застревания в локальном оптимуме [2].

Если границы поиска оптимального значения слишком большие, и в их пределах находится несколько локальных оптимумов, то работа алгоритма может занять большое количество времени перед тем, как будет найден глобальный оптимум, даже несмотря на использование мутации.

В качестве решения данной проблемы может быть применение технологии технологий распараллеливания.

Основная идея такого подхода заключается в следующем: если нам заданы исходные границы поиска, в которых необходимо найти оптимальное значение, то промежуток поиска разбивается на несколько частей. После этого, используя технологии распараллеливания, мы создаем несколько потоков. Каждый поток получает одну из частей, полученных в результате разбивки исходных границ, и начинает нахождение оптимальных значений в собственных границах при помощи стандартных шагов генетического алгоритма. Данный процесс происходит на всех участках одновременно. После того, как в каждой из частей были найдены лучшие значения функции полезности, производится их сравнение, в результате чего находится наилучшее значение из всех.

Таким образом, мы избегаем попадания в локальный минимум, т.к. в одном из промежутков всегда находится глобальный. При этом точность нахождения фитнесс-функции будет выше, чем на более маленькие промежутки мы будем делить исходные границы поиска.

Сравним классический генетический алгоритм с алгоритмом, где используется технология OpenMP.

В качестве исходной функции возьмем $x^4 - 6x^2 + 5x + 6$, которая имеет два минимума в точках –1.912 и 1.465 (Рисунок 1). И попробуем найти глобальный минимум в промежутке от – 10 до 10 и запишем результаты нахождения в таблицу. Количество поколений – 150. Размер популяции – 20. Вероятность мутации – 0.1. Для алгоритма с OpenMP исходные границы разбиваются на 4 части. И 4 потока получают по 1 из частей.

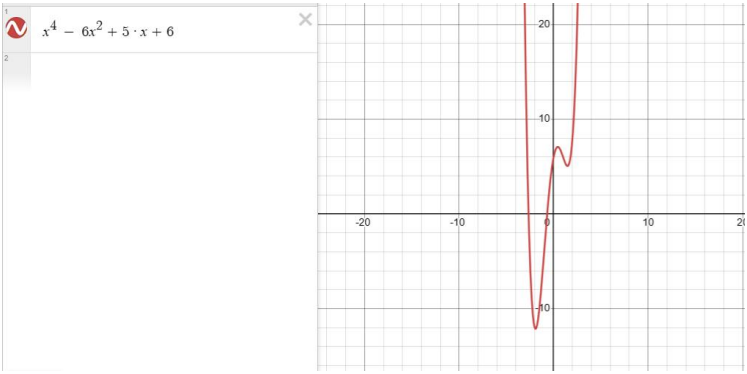


Рисунок 1 – График функции $x^4 - 6x^2 + 5x + 6$

Таблица 1 – Результаты нахождения глобального минимума

	Классический алгоритм	Алгоритм с использованием OpenMP (4 потока)
1	- 2.51	-2.02393
2	- 1.32	-2.00319
3	-0.829541	-1.81998
4	-2.00937	-1.9705
5	-2.25545	-2.02549
Среднее значение	-1.7848722	-2.011692

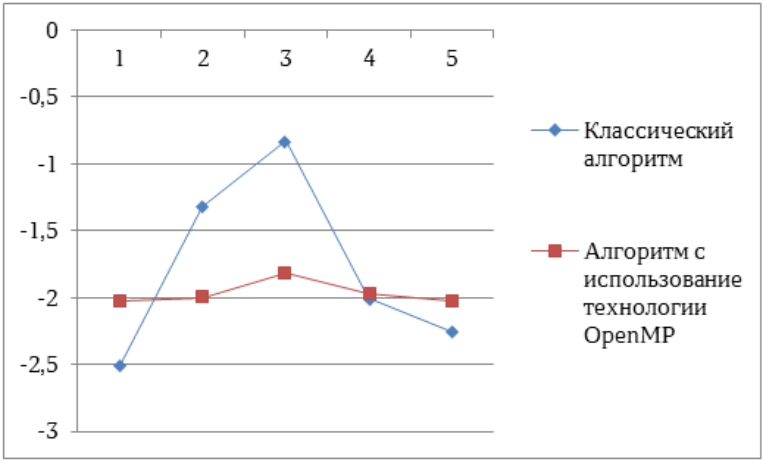


Рисунок 2 – График сравнения двух алгоритмов

Как видно из графика, что алгоритм с использование технологии OpenMP работает гораздо точнее, если функции предполагает наличие нескольких экстремумов, что говорит об эффективности данного подхода при решении задач оптимизации.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Гребенникова, И.В. Методы оптимизации: учебное пособие / И. В. Гребенникова. – Екатеринбург: УрФУ, 2017. – 148 с.

2. Панченко, Т.В. Генетические алгоритмы [Текст]: учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. – Астрахань: Издательский дом «Астраханский университет», 2007. – 87 [3] с.

УДК 81'322.2

ПОЛОВИНКИН А.В.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

СОВРЕМЕННЫЕ ИНТЕЛЛЕКТУАЛЬНЫЕ МЕТОДЫ ДЛЯ АВТОМАТИЧЕСКОГО АНАЛИЗА И ОБРАБОТКИ ТЕКСТОВЫХ ДАННЫХ

Рассматриваются интеллектуальных методы для автоматического анализа и обработки текстовых данных в современной информационной среде.

Исследование и разработка интеллектуальных методов для автоматического анализа и обработки текстовых данных являются важными направлениями в современной информационной технологии. Эта проблема актуальна по нескольким причинам: в современном мире происходит информационный взрыв. Под этим понимается, что объем текстовых данных в интернете и корпоративных базах данных постоянно растет.

Другой проблемой является тот факт, что текстовые данные часто представляют собой неструктурированную информацию, что делает их сложными для анализа традиционными методами. Интеллектуальные методы обработки текста позволяют извлекать смысл, выявлять паттерны и делать выводы из неструктурированных данных.

В области обработки естественного языка используется обширный арсенал методов и приемов анализа текстовой информации. От самых элементарных операций, таких как токенизация и удаление стоп-слов, до более сложных алгоритмов, включая рекуррентные нейронные сети. Каждый этап играет решающую роль в обеспечении эффективного анализа текста.

Первым шагом является токенизация, при которой текст сегментируется на отдельные компоненты — слова, фразы и знаки препинания, известные как токены. Вслед за этим удаляются стоп-слова, такие как предлоги и союзы, которые хотя и распространены, но не несут существенного смыслового значения. Этот шаг позволяет сосредоточиться на ключевых терминах и снизить размерность данных [1].

После предварительной обработки текста и удаления стоп-слов применяются методы лемматизации и стемминга, помогающие уменьшить разнообразие словоформ. При лемматизации существительные слова приводятся к базовой форме, т.е. существительные приводят к именительному падежу, все глаголы должны отвечать на вопрос «что делать?» и т.п. Лемматизация имеет важное свойство: нет необходимости хранить всевозможные словоформы.

Другим вариантом обработки является – стемминг. Это процесс обрезания слова до его основы или "стебля" с целью удаления суффиксов и/или окончаний. Цель стемминга - свести слово к его базовой форме, чтобы схлопнуть различные грамматические формы слова и сократить размерность данных. В результате слова с одинаковой основой будут представлены одним и тем же токеном, что может улучшить качество алгоритмов обработки естественного языка.

После выполнения лемматизации/стемминга, начинается парсинг. Это процесс, который включает в себя синтаксический анализ предложений для определения их структуры и зависимостей между словами. Т.е. парсинг позволяет определить, какие слова являются подлежащими, сказуемыми, дополнениями и т.д.

Когда заканчивается парсинг, происходит выделение сущностей. Т.е. происходит процесс автоматического извлечения и классификации именованных объектов, таких как лица, организации, локации и другие семантически важные элементы. Этот этап включает в себя применение методов машинного обучения, основанных на аннотированных корпусах текста, а также использование моделей, обученных на задачах именованной сущности.

Потом начинается этап извлечение информации из текста, который представляет собой процесс автоматического выделения структурированной информации, такой как даты, события и отношения между сущностями. Этот этап включает в себя анализ синтаксической и семантической структуры текста, а также применение методов машинного обучения для обнаружения и извлечения конкретных фактов.

Далее идет синтаксический анализ и синтаксический анализ, при котором выясняется структура предложений и зависимости между словами, что имеет решающее значение для понимания контекста текста.

В настоящее время выделяют следующие алгоритмы обработки текста: «Мешок слов», TF-IDF, N-граммы, скрытые модели Маркова, Word2Vec и GloVe.

Одним из простейших методов обработки естественного языка является метод «Мешок слов». Этот метод преобразует текст в неупорядоченный набор слов, игнорируя порядок и структуру предложений. Каждое слово считается отдельным признаком, и текст представляется в виде вектора, где каждая компонента соответствует частоте встречаемости слова. Важным недостатком этого метода является потеря порядка слов, что может стать проблемой в задачах, где важен контекст [2].

На основе «Мешка слов» реализована мера TF-IDF. TF-IDF – это статистическая мера, используемая для оценки важности слова в документе относительно всего корпуса текстов. Она вычисляется как:

$$IDF(t, D) = \log\left(\frac{N}{1+|\{d \in D : t \in d\}|}\right)$$

Где N – общее количество документов в коллекции, $|\{d \in D : t \in d\}|-$ количество документов, содержащих термин t (частота документа). Чем выше IDF – тем реже встречается слово в документе.

TF-IDF позволяет выделить ключевые термины в документе, подчеркивая их важность. В рекомендательных системах, где текстовая информация играет ключевую роль, TF-IDF помогает выделить наиболее релевантные термины и учесть их в процессе рекомендаций [3].

Для оценки наиболее часто встречающихся комбинаций слов используется метод N-грамм. В отличие от предыдущих методов, в N-граммах текст разбивается на последовательности из n элементов (например, слов, символов), взятых из текста без перекрытия. Наиболее часто используются такие последовательности, как биграммы (n = 2) и триграммы (n = 3). После чего, для каждой уникальной последовательности также подсчитывается частота ее встречаемости в тексте. Выявив наиболее частые комбинации, возможно предсказание следующих элементов в текущей последовательности.

Данный метод имеет ряд ограничений, таких как: игнорирование контекста и ограниченность порядка, т.е. более высокие порядки N-грамм, (например, триграммы) могут сталкиваться с проблемой разреженных данных [4].

Для моделирования последовательностей слов в тексте могут быть использованы скрытые марковские модели. Скрытые состояния моделируют различные лингвистические категории, такие как "существительное", "глагол", "прилагательное" и другие. Эти состояния, хотя и невидимы напрямую, предполагаются влияющими на выбор слов в тексте. Слова в тексте рассматриваются как наблюдаемые события. Каждое слово связано с конкретным лингвистическим состоянием, хотя мы видим только сами слова и не имеем точной информации о текущем состоянии системы. Вероятности переходов определяют вероятность перехода из одного лингвистического состояния в другое. Например, они могут выражать вероятность того, что после существительного последует глагол. Каждому лингвистическому состоянию присваивается распределение вероятностей для генерации слов. Например, для состояния "глагол" вероятность генерации слова "бегать" может быть выше, чем для слова "стол". Начальные вероятности отражают вероятность начала предложения с определенного лингвистического состояния. Это

подчеркивает, с какой лингвистической категории чаще всего начинаются предложения [5].

Метод Word2Vec основан на идеи обучения векторных представлений через анализ их контекста в тексте. Модель, используемая в Word2Vec, может быть реализована с использованием двух основных подходов: Continuous Bag of Words (CBOW) и Skip-Gram. В CBOW модель стремится предсказать целевое слово, исходя из контекста, в то время как в Skip-Gram модели пытаются предсказать контекст, исходя из целевого слова. Преимущества этой модели включают в себя создание плотных векторных представлений слов, которые отражают семантические отношения между словами. Этот метод обладает способностью выражать схожесть и контекстуальные зависимости в текстовых данных.

Метод GloVe основан на глобальной статистике совстречаемости слов в корпусе текста. Метод строит матрицу совстречаемости, в которой элементы отражают частоту встречаемости пар слов. Глобальные зависимости между словами выделяются, а векторы слов вычисляются с использованием этой информации. Достоинство GloVe - способность учитывать глобальные контексты [6].

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Гладилин П.Е., Боченина К.О., Технологии машинного обучения. – СПб: Университет ИТМО, 2020. – 75 с.

2. Обработка естественного языка. Python и spaCy на практике. – СПб.: Питер, 2021. – 256 с.

3. Орлов Г. М., Игнатьева О. А., Васин А. Г., Низомутдинов Б. А. Современные методы обработки и анализа данных. – СПб.: Университет ИТМО, 2021. – 147 с.

4. Орлянская Н.П., Янаева М.В. Информационный поиск и обработка естественного языка: конспект лекций / Кубанский государственный технологический университет. – Краснодар: изд. ФГБОУ ВО «КубГТУ», 2021. – 187 с.

5. Пучковская А.А., Зимина Л.В., Волков Д.А., Введение в цифровые гуманитарные исследования. – СПб: Университет ИТМО, 2021. – 61 с.

УДК 004.65

ПОПОВА Д.О.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ОСНОВЫ СЕТЕВОГО ПЛАНИРОВАНИЯ И УПРАВЛЕНИЯ

Статья посвящена основам сетевого планирования и управления. Сетевое планирование и управление – это метод управления проектами, который позволяет оптимизировать планирование, контроль и управление временем.

Основные инструменты для сетевого планирования

1. Диаграмма Ганта – инструмент для визуализации последовательности работ и управления временем проекта.

2. Сетевая диаграмма – инструмент для определения зависимостей между работами и выявления критических путей в проекте.

3. Метод оценки времени-техника определения продолжительности работ на основе исторических данных, экспертных оценок и других факторов.

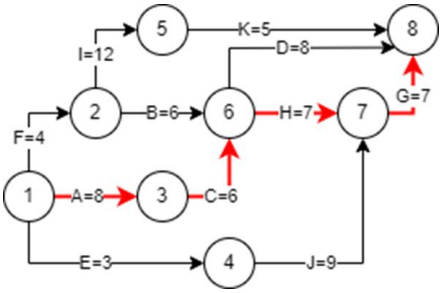


Рисунок 1 – Сетевой график

«Графический» способ. Для расчета параметров «графическим» способом узлы сетевого графика делятся на 4 сектора, в каждый из которых вносятся результаты вычислений:

− сверху отображается номер события (i);

− слева — ранний срок наступления события (T р i);

− справа — поздний срок наступления события (T п i);

− снизу — резерв времени события (Ri);

− в квадратных скобках под стрелкой — полный и свободный резерв каждой работы (R п ij; R с ij);

− над стрелкой – продолжительность работы и трудовой ресурс (tij).

Графический способ включает в себя создание диаграмм для представления компонентов сети, их взаимосвязей и потока данных. Этот метод может помочь выявить потенциальные проблемы, оптимизировать производительность сети и облегчить взаимодействие между членами команды

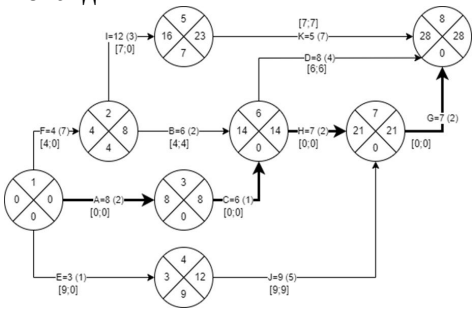


Рисунок 2 – «Графический» способ

Таблица 1 – Табличный способ

Кол-во предшествующих работ	**Код i-j**	**$T^{рн}ij$**	**tij**	**$T^{ро}ij$**	**Тпнij**	**tij**	**$T^{по}ij$**	**Rпij**	**$R^{с}ij$**
0	1-2	0	4	4	4	4	8	4	0
0	1-3	0	8	8	0	8	8	0	0
0	1-4	0	3	3	9	3	12	9	0
1	2-5	4	12	16	1	12	23	7	0
1	2-6	4	6	10	8	6	14	4	4
1	3-6	8	6	14	8	6	14	0	0
1	4-7	3	9	12	12	9	21	9	9
1	5-8	16	5	21	23	5	28	7	7
2	6-7	14	7	21	14	7	21	0	0
2	6-8	14	8	22	20	8	28	6	6
2	7-8	21	7	28	21	7	28	0	0
3	8-	28	-	-	-	-	-	-	-

Результаты «графического» и табличного способов имеют одинаковые результаты. Рассмотренные способы могут применяться самостоятельно, независимо друг от друга.

Анализ и оптимизация сетевого графика. Анализ и оптимизация сетевых графиков имеют большое значение при управлении проектами во многих областях. Этот подход позволяет представить проект в виде сети взаимосвязанных задач и определить зависимости между ними, а также оценить продолжительность проекта и прогнозировать возможные задержки.

Анализ и оптимизация сетевого графика могут производиться как вручную, так и при помощи MicrosoftProject или иных схожих продуктов, которые позволяют выявить критические операции и найти решения для сокращения продолжительности проекта и повышения его эффективности.

На рисунке 3 представлены первоначальные данные для построения графика (информация о сроках начала и окончания работ, о предшествующих работах, о назначенных на работу ресурсах).

	Название задачи	Длительность	Начало	Окончание	Предшественник	Названия ресурсов
1	1-2	3,43 дней	08.01.07	Чт 11.01.07		Рабочий1;Рабочий2;Рабочий6;Рабочий7;Рабочий8;Рабочий9;Рабочий3
2	1-3	8 дней	08.01.07	Ср 17.01.07		Рабочий1;Рабочий2
3	1-4	3 дней	08.01.07	Ср 10.01.07		Рабочий3
4	2-5	12 дней	11.01.07	Пн 29.01.07	1	Рабочий4;Рабочий5;Рабочий6
5	2-6	6 дней	11.01.07	Пт 19.01.07	1	Рабочий6;Рабочий7
6	3-6	6 дней	18.01.07	Чт 25.01.07	2	Рабочий7
7	4-7	9 дней	11.01.07	Вт 23.01.07	3	Рабочий8;Рабочий9;Рабочий10;Рабочий11;Рабочий1
8	5-8	5 дней	29.01.07	Пн 05.02.07	4	Рабочий2;Рабочий3;Рабочий4;Рабочий5;Рабочий6;Рабочий7;Рабочий8
9	6-7	7 дней	26.01.07	Пн 05.02.07	5;6	Рабочий2;Рабочий3
10	6-8	8 дней	26.01.07	Вт 06.02.07	5;6	Рабочий4;Рабочий5;Рабочий6;Рабочий7
11	7-8	7 дней	06.02.07	Ср 14.02.07	9;7	Рабочий8;Рабочий9
12	8-	0 дней	14.02.07	Ср 14.02.07	8;10;11	

Рисунок 3 – Данные для построения (до оптимизации)

На рисунке 4 представлен сетевой график, построенный по начальным данным, показаны взаимосвязи между работами.

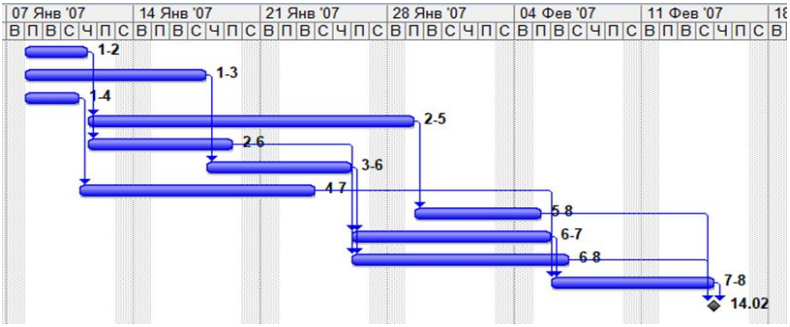


Рисунок 4 – Сетевой график (до оптимизации)

На рисунке 5 представлена статистика проекта до оптимизации.
Длительность проекта до оптимизации — 28 дней;
Стоимость проекта до оптимизации — 355`500 руб.

	Начало	Окончание
Текущее	Пн 08.01.07	Ср 14.02.07
Базовое	НД	НД
Фактическое	НД	НД
Отклонение	0д	0д

	Длительность	Трудозатраты	Затраты
Текущие	28д	1 896ч	р.355 500,00
Базовые	0д?	0ч	р.0,00
Фактические	0д	0ч	р.0,00
Оставшиеся	28д	1 896ч	р.355 500,00

Рисунок 5 – Статистика проекта (до оптимизации)

Анализ сетевого графика показал, что напряженной зоной работ является работа 1-6. Для оптимизации были предприняты следующие шаги:

1. Выявление резерва времени.

Путь 1-2-6 обладает резервом в 4 дня;

Путь 1-3-6 (лежит на критическом пути) обладает резервом в 0 дней.

2. Вычисление трудоемкости (Q). Q(1-2-6) = 4*7+6*2 = 40 чел/дн; Q(1-3-6) = 8*2+6*1 = 22 чел/дн.

3. Вычисление количества исполнителей, доступных для перевода между работами.

X(1-2-6) = 9-40/(10+2) = 9-40/12 ~ 5 чел.

4. Вычисление времени для работ. t(1-2-6) = 40/(9-5) = 10 дн;

t(2-5-7) = 22/(3+5) ~ 3 дн.

5. Распределение трудовых и временных ресурсов.

Оптимизированный сетевой график с новым критическим путем длительностью в 24 дня представлен на рисунке 6.

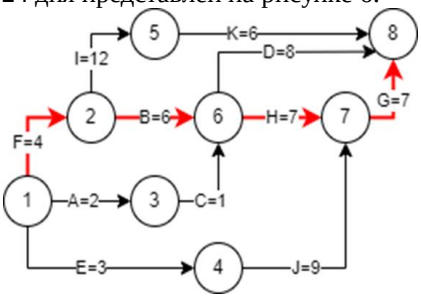


Рисунок 6 – Первая оптимизация

Дальнейшая оптимизация проекта была произведена в Microsoft Project. На рисунке 7 представлены данные для построения графика.

Оптимизация была произведена при помощи перераспределения трудовых ресурсов и сроков выполнения работ.

	Название задачи	Длительность	Начало	Окончание	Предше	Названия ресурсов
1	1-2	4 дней	Пн 08.01.07	Чт 11.01.07		Рабочий1;Рабочий2
2	1-3	1 день	Пн 08.01.07	Пн 08.01.07		Рабочий3;Рабочий4
3	1-4	3 дней	Пн 08.01.07	Ср 10.01.07		Рабочий5
4	2-5	8 дней	Пт 12.01.07	Вт 23.01.07	1	Рабочий1;Рабочий2;Рабочий3;Рабочий4;Рабочий5;Рабочий6
5	2-6	6 дней	Пт 12.01.07	Пт 19.01.07	1	Рабочий7;Рабочий8
6	3-6	1 день	Вт 09.01.07	Вт 09.01.07	2	Рабочий3
7	4-7	6 дней	Чт 11.01.07	Чт 18.01.07	3	Рабочий9;Рабочий10;Рабочий11
8	5-8	7,5 дней	Ср 24.01.07	Пт 02.02.07	4	Рабочий1;Рабочий2
9	6-7	7 дней	Пн 22.01.07	Вт 30.01.07	5;6	Рабочий7
10	6-8	5,33 дней	Пн 22.01.07	Пн 29.01.07	5;6	Рабочий8;Рабочий9;Рабочий10
11	7-8	3,5 дней	Ср 31.01.07	Пн 05.02.07	9;7	Рабочий11;Рабочий6
12	8-	0 дней	Пн 05.02.07	Пн 05.02.07	8;10;11	

Рисунок 7 – Данные для построения (после оптимизации)

На рисунке 8 представлен сетевой график, построенный по оптимизированным данным, показаны взаимосвязи между работами.

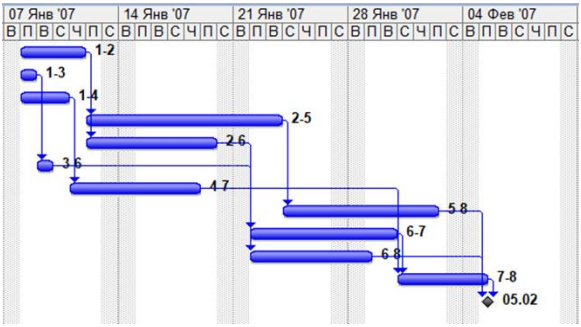


Рисунок 8 – Сетевой график (после оптимизации)

На рисунке 9 представлена статистика проекта после оптимизации. Длительность проекта после оптимизации — 21 день;

Стоимость проекта после оптимизации — 210`000 руб.

	Начало	Окончание
Текущее	Пн 08.01.07	Пн 05.02.07
Базовое	НД	НД
Фактическое	НД	НД
Отклонение	0д	0д

	Длительность	Трудозатраты	Затраты
Текущие	20,5д	1 096ч	р.210 000,00
Базовые	0д?	0ч	р.0,00
Фактические	0д	0ч	р.0,00
Оставшиеся	20,5д	1 096ч	р.210 000,00

Рисунок 9 – Статистика проекта (после оптимизации)

Заключение

Основы сетевого планирования и управления являются критически важными для обеспечения эффективной работы и развития сетей любого масштаба. В современном мире, где зависимость от компьютерных сетей и Интернета постоянно растет, ответственность за их функционирование и состояние лежит на сетевых администраторах и специалистах по информационным технологиям.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ 7.32-2017. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научноисследовательской работе. Структура и правила оформления (введен в действие Приказом Росстандарта от 24.10.2017 N 1494-ст).

2. Управление ИТ-проектами. Лекции // Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Рязанский государственный радиотехнический университет имени В.Ф. Уткина». Автор: Дудко И.С. – 2023.

УДК 004.4

РАНОВСКИЙ М.В.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

СРАВНЕНИЕ АЛГОРИТМОВ ПОИСКА ПУТИ

В статье проводится сравнение популярных алгоритмов поиска пути с анализом их возможностей, преимуществ и недостатков.

Введение

Поиск пути – это фундаментальная задача в области компьютерных наук, имеющая множество применений в робототехнике, картографии, играх, и других областях. Существует множество алгоритмов поиска пути, каждый из которых имеет свои преимущества и недостатки. В общем случае их можно разделить на две группы:

- алгоритмы, предназначенные для поиска кратчайшего пути;
- алгоритмы, предназначенные для поиска субоптимального пути.

Оптимальный путь — это путь, который обеспечивает минимальные затраты для достижения цели. Это означает, что оптимальный путь является наилучшим из возможных вариантов и гарантированно обладает наименьшей стоимостью или наименьшими затратами. В то время как субоптимальный путь — это путь, который, возможно, не является наилучшим, но все еще приемлем для достижения цели.

К первой группе относятся алгоритмы, которым необходимо полностью просмотреть область поиска для нахождения оптимального пути. Такие методы в большинстве случаев неприменимы из-за больших расходов вычислительных мощностей, так как необходимо полное исследование всей карты и ее хранение.

Алгоритмы второй группы позволяют находить приемлемое решение за разумное время, не требуя исследования всей области. Примером таких алгоритмов являются эвристические алгоритмы, которые, оценивая текущую позицию и используя эвристические функции, стремятся к конечной точке, даже если не гарантируют нахождение оптимального пути. Это позволяет существенно сократить время и ресурсы, затрачиваемые на поиск решения, при сохранении достаточного качества полученного маршрута.

Выбор эвристики

В большинстве разбираемых в этой статье алгоритмов поиска пути должна быть определена эвристическая функция, которая определяет общую стоимость достижения целевой точки из текущей позиции. Она используется для приблизительной оценки оставшегося расстояния до цели, обуславливая выбор направления движения алгоритма и определяя, какой путь следует выбрать на основе предположений о том, какая часть пути будет более перспективной.

Эвристика должна быть информативной, то есть она должна давать хорошее приближение к реальной стоимости достижения цели, но не обязательно точной. Эффективность алгоритма увеличивается с повышением точности эвристики, но очень точная эвристика будет более вычислительно сложной.

Для алгоритмов поиска пути эвристика обычно выражается в виде функции оценки, которая принимает на вход текущую точку и возвращает приближенную цену достижения целевой точки через из данной.

Существует несколько типов эвристических функций, которые могут быть использованы в зависимости от конкретной задачи и характеристик среды:

1. Манхэттенская эвристика

Манхэттенская эвристика вычисляет приближенное расстояние между текущей позицией и целевой позицией, используя только горизонтальные и вертикальные перемещения (без диагональных).

Формула для вычисления манхэттенского расстояния: $|dx| + |dy|$,

где *dx* - разница между горизонтальными координатами текущей и целевой позиции, а *dy* - разница между вертикальными координатами.

2. Эвристика Евклида

Эвристика Евклида вычисляет прямое расстояние между текущей позицией и целевой позицией, используя геометрическую формулу расстояния.

Формула для вычисления расстояния Евклида: $\sqrt{dx^2 + dy^2}$

3. Octile эвристика

Octile эвристика учитывает возможность движения по диагонали, но при этом считает диагональное расстояние немного дороже, чем горизонтальное или вертикальное.

Формула для вычисления Octile расстояния:

$$max(|dx|, |dy|) + (\sqrt{2} - 1) * min(|dx|, |dy|)$$

4. Эвристика Чебышева

Эвристика Чебышева вычисляет расстояние как максимальную разницу между горизонтальными и вертикальными координатами текущей и целевой позиций.

Формула для вычисления расстояния Чебышева: $max(|dx|, |dy|)$

Каждая из этих эвристических функций имеет свои особенности и может быть более или менее подходящей в зависимости от конкретной ситуации и характеристик среды, в которой происходит поиск пути.

*Алгоритм A**

Алгоритм A* (A-star) — один из самых эффективных алгоритмов поиска пути, объединяющий в себе эвристический информированный поиск и поиск по первому наилучшему совпадению. A* часто используется для нахождения кратчайшего пути в графах или на сетках.

Принцип работы A* заключается в том, что он оценивает стоимость достижения цели через каждую вершину. Это делается с использованием эвристической функции, оценивающая расстояние от текущей точки до целевой. Алгоритм находит оптимальный путь,

выбирая на каждой итерации следующие вершины для исследования на основе их стоимости пути, которая состоит из фактической стоимости пути от начальной вершины до текущей и эвристической оценки стоимости пути до цели через эту вершину.

A* может проводить поиск как от начальной вершины к цели, так и от цели к начальной. Это позволяет ему эффективно находить оптимальный путь, уменьшая количество вершин, которые необходимо просмотреть.

Формула для оценки стоимости пути от начальной вершины до текущей выглядит так: $f(n) = g(n) + h(n)$ - общая оценка стоимости пути через вершину n, $g(n)$ - стоимость пути от начальной вершины до вершины n, а $h(n)$ - эвристическая оценка стоимости достижения цели через вершину n.

Преимущества A* включают его оптимальность при правильно выбранной эвристике, его эффективность за счет использования эвристики для ускорения поиска, а также его широкое применение в различных областях, таких как робототехника, игровая разработка и планирование маршрутов.

Однако алгоритм A* подвержен влиянию выбранной эвристики - неправильно выбранная эвристика может привести к неправильному пути или потере оптимальности. Также он требует дополнительной памяти для хранения списков вершин и может иметь вычислительную сложность из-за сложности некоторых эвристик.

Рассмотрим построение пути алгоритмом A* с использованием эвристик, описанных ранее (Рисунок 1-2). Как видно из рисунков, при подобном расположении стартового и целевого узла все алгоритмы, использующие не манхэттенскую эвристику, тратят больше времени и операций на просмотр лишних ячеек. Поэтому для рассмотрения следующих алгоритмом будем использовать манхэттенскую эвристику, как более эффективную.

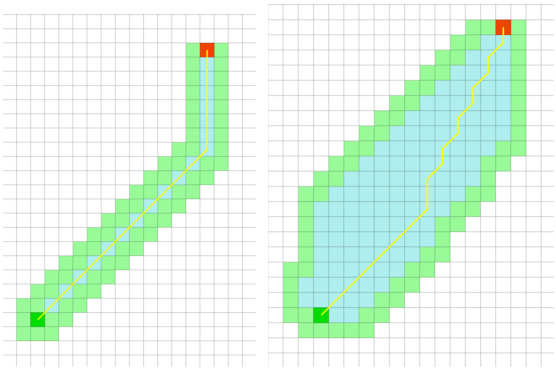


Рисунок 1 – Иллюстрация работы алгоритма A* с манхэттенской эвристикой (слева) и эвристикой Евклида (справа)

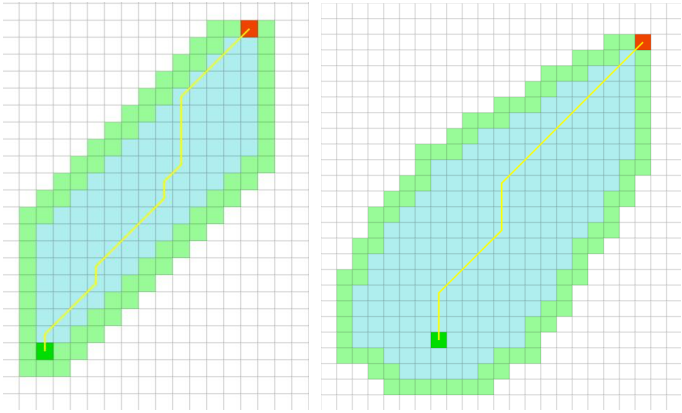


Рисунок 2 – Иллюстрация работы алгоритма A* с Octile эвристикой (слева) и эвристикой Чебышева (справа)

Также рассмотрим, как будет справляться алгоритм с учетом препятствий (Рисунок – 3). Будем использовать данный лабиринт при анализе следующих алгоритмов.

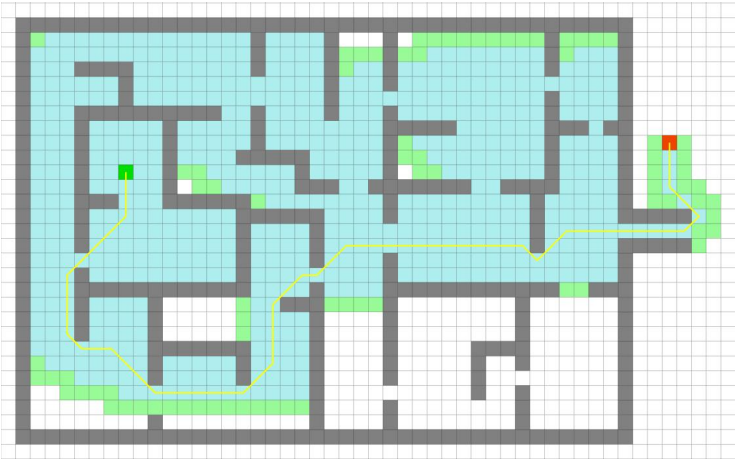


Рисунок 3 – Иллюстрация работы алгоритма A* в лабиринте

Поиск в ширину

Поиск в ширину (BFS - Breadth-First Search) — это алгоритм поиска пути, исследующий все соседние вершины на каждом уровне графа, начиная с начальной вершины, прежде чем переходить к исследованию вершин на следующем уровне.

Принцип работы BFS заключается в том, что алгоритм начинает свою работу с исходной вершины и добавляет все ее соседние вершины в очередь. Затем он переходит к следующей вершине в очереди и добавляет все ее неисследованные соседние вершины в конец очереди. Процесс повторяется, пока не будет исследовано все пространство графа или не будет найдена конечная вершина.

Одно из важнейших преимуществ BFS заключается в том, что несмотря на большое количество итераций и просмотренных позиций, он всегда выстраивает оптимальный путь от начальной вершины ко всем другим вершинам в невзвешенном графе или сетке. Однако взвешенный граф или граф с циклами может привести к тому, что BFS не сможет найти оптимальный путь из-за того, что продолжит исследовать вершины на каждом уровне в порядке, который не обязательно будет оптимальным.

Поиск в ширину хранит все вершины, которые нужно исследовать, в очереди. Это обеспечивает равную возможность исследования всех вершин на текущем уровне, что делает BFS полезным при поиске кратчайших путей или при обходе графа в определенном порядке. Однако это также требует дополнительной памяти для хранения очереди вершин, особенно для больших графов.

Как видно на рис.4, алгоритму поиска в ширину пришлось просмотреть все ячейки в лабиринте чтобы добраться до цели и потратить почти в два раза больше операций чем алгоритму A*. Это связано с тем данный алгоритм попросту не имеет целенаправленного движения.

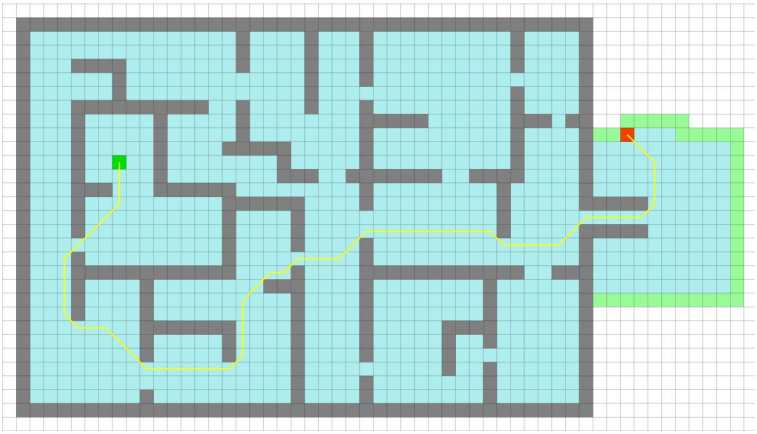


Рисунок 4 – Иллюстрация работы поиска в ширину в лабиринте

Поиск по первому наилучшему совпадению

Поиск по первому наилучшему совпадению, поиск «лучший — первый» (Best-First Search) — это алгоритм поиска пути, который в своей работе выбирает для перехода одну из соседних вершину на основе их эвристических оценок. В отличие от поиска в ширину, который исследует все соседние вершины, данный алгоритм выбирает для просмотра вершину, которая, кажется, наиболее близко к цели.

Принцип работы поиска «лучший — первый» заключается в том, что алгоритм оценивает стоимость достижения цели через каждую вершину с помощью эвристической функции. Затем алгоритм выбирает вершину с наилучшей эвристической оценкой и исследует ее соседей. Процесс повторяется, пока не будет найдена конечная точка или не останется вершин для исследования.

Одним из основных преимуществ поиска по первому наилучшему совпадению является его способность находить путь к цели, основываясь на текущей информации. Это позволяет алгоритму сосредоточиться на областях, которые возможно ведут к цели, и игнорировать менее перспективные ветви. Кроме того, алгоритм может работать более эффективно, чем поиск в ширину, если цель находится на значительном удалении от начальной вершины.

Однако алгоритм не гарантирует нахождение оптимального пути, поскольку он может сосредоточиться на одной известной эвристике и игнорировать другие аспекты графа. Это может привести к тому, что алгоритм найдет путь, который якобы является близким к цели, но не является оптимальным. Кроме того, эвристическая функция играет ключевую роль в успехе поиска, и неправильно выбранная эвристика может привести к неэффективному поведению алгоритма.

Из рис.5 видно, что алгоритму получилось достичь цели через субоптимальный путь, но для этого ему понадобилось просмотреть намного меньше узлов, чем остальным алгоритмам.

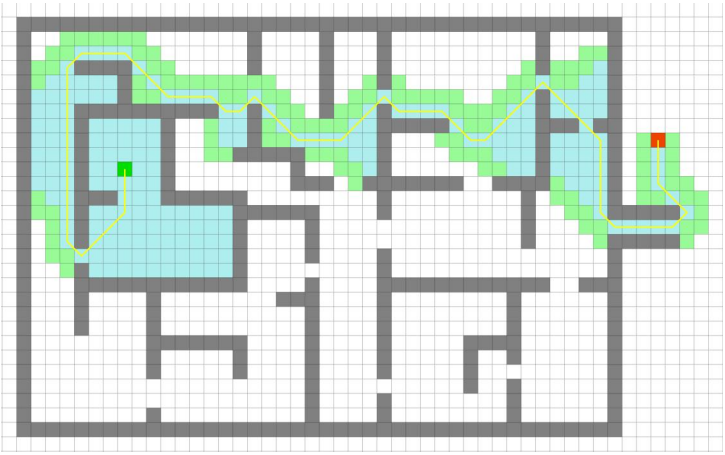


Рисунок 5 – Иллюстрация работы поиска по первому наилучшему совпадению в лабиринте

Метод Дейкстры

Метод Дейкстры (Dijkstra's algorithm) — это алгоритм, предназначенный для поиска кратчайшего пути от одной вершины к каждой другой вершине во взвешенном графе без отрицательных весов ребер. Алгоритм был разработан нидерландским ученым Эдсгером Дейкстрой в 1956 году и является одним из основных алгоритмов в теории графов и области алгоритмов поиска пути.

В основе метода Дейкстры лежит итеративное обновление кратчайших путей от начальной вершины ко всем остальным точкам в графе. На каждом шаге алгоритма выбирается вершина с наименьшим временем достижения среди тех вершин, которые еще не были исследованы. Затем для каждой соседней вершины, связанной с

выбранной, обновляется кратчайший путь, если новый путь короче текущего.

Важнейшим достоинством метода Дейкстры является его способность находить кратчайший путь от начальной вершины ко всем остальным вершинам во взвешенном графе без отрицательных весов. Он может использоваться для поиска оптимального пути между двумя конкретными вершинами посредством прерывания алгоритма после достижения целевой вершины.

Однако метод Дейкстры не применим к графам, содержащим ребра с отрицательными весами, так как не учитывает возможность уменьшения общей стоимости пути при прохождении через отрицательные ребра. Также метод Дейкстры требует, чтобы все веса ребер были неотрицательными, и он может терять эффективность при работе с большими графами из-за его временной сложности.

Из рис.6 видно, что в данном случае метод Дейкстры схож с поиском в ширину.

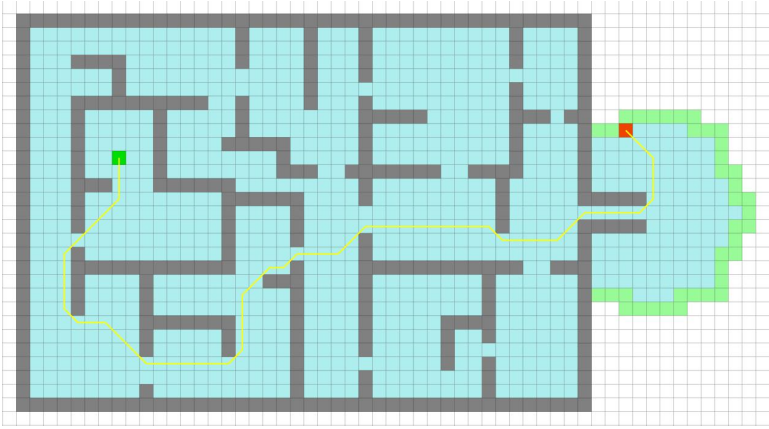


Рисунок 6 – Иллюстрация работы алгоритма Дейкстры в лабиринте

Поиск точки перехода

Поиск точки перехода (Jump Point Search, JPS) — это эффективный алгоритм поиска пути, который ускоряет поиск на сетках посредством предварительного пропуска несущественных участков. Он был разработан в 2011 году и является доработкой алгоритма A*.

Принцип работы JPS заключается в том, чтобы искать ключевые точки, называемые точками перехода, которые можно пропустить при поиске пути. Эти точки перехода определяются в результате анализа

структуры карты и выявления "препятствий", которые можно обойти без потери оптимальности пути.

Когда алгоритм достигает точки перехода, он делает прыжок через эту точку в направлении, которое позволяет ему достичь следующей ключевой точки или цели более эффективно. Это позволяет уменьшить количество вершин, которые необходимо исследовать, и ускорить процесс поиска пути.

Одним из главных преимуществ JPS является его способность ускорять поиск пути на картах сетки, особенно когда препятствия образуют длинные прямые линии. Алгоритм также обеспечивает оптимальность найденного пути, так как он основан на алгоритме A*. Однако алгоритм JPS может быть более сложным в реализации и требовать более глубокого понимания структуры карты и правил поиска. Кроме того, он может потреблять больше памяти для хранения дополнительной информации о точках перехода на карте (Рисунок 7).

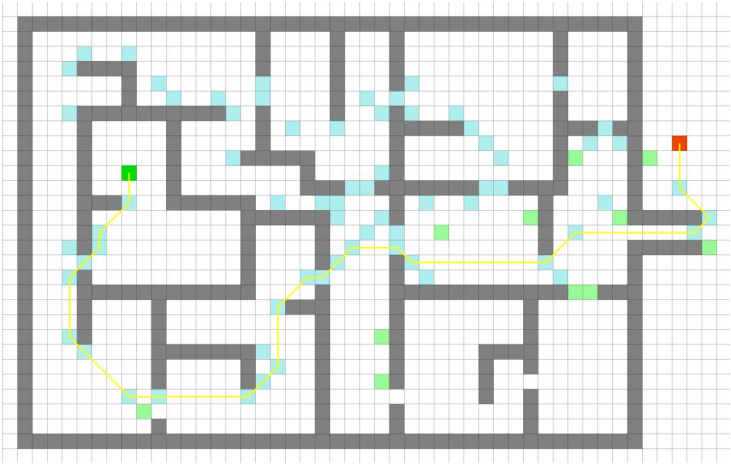


Рисунок 7 – Иллюстрация работы алгоритма поиска точки перехода в лабиринте

Заключение

В заключение можно составить сравнительную таблицу всех рассмотренных алгоритмов, чтобы проиллюстрировать их основные характеристики:

Таблица 1 – Сравнительный анализ

Алгоритм	**Основные преимущества**	**Основные недостатки**	**Сложность**	**Область применения**
A*	Оптимальность при правильной эвристике Эффективность в больших графах Широкое применение	Влияние выбранной эвристики Дополнительная память	$O(b^d)$	Игровая разработка, маршрутизация, робототехника
Поиск в ширину	Гарантированная оптимальность в невзвешенных графах	Неэффективность взвешенных графов Потребление памяти	$O(V+E)$	Графовая аналитика, обход графов, топологическая сортировка
Поиск по первому наилучшему совпадению	Быстрое нахождение пути к цели Эффективность при правильной эвристике	Неоптимальность при неправильной эвристике	Переменная	Робототехника, автономные системы, игровая разработка
Метод Дейкстры	Нахождение кратчайшего пути во взвешенных графах без отрицательных весов ребер	Неэффективность при наличии отрицательных весов ребер	$O((V+E)*$ $*\log(V))$	Маршрутизация, транспортное планирование, сетевая аналитика
Поиск точки перехода	Ускорение поиска пути на картах сетки Гарантированная оптимальность пути	Сложность реализации Дополнительное потребление памяти	Переменная	Робототехника, игровая разработка, планирование пути

Каждый из этих алгоритмов имеет свои особенности, преимущества и ограничения, и выбор конкретного алгоритма зависит от требований конкретной задачи и характеристик окружающей среды.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Введение в алгоритм A* [Электронный ресурс]. – URL: https://habr.com/ru/articles/331192/.

2. Heuristic pathfinding algorithm [Электронный ресурс]. – URL: https://programmersought.com/article/20106391396/.

3. Алгоритм поиска пути Jump Point Search [Электронный ресурс]. – URL: https://habr.com/ru/articles/162915/.

4. Pathfinding.js github [Электронный ресурс]. – URL: https://github.com/qiao/PathFinding.js.

УДК 004.65

РУСАЛЕЕВ И.В.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ИССЛЕДОВАНИЕ МЕТОДОВ РАСПОЗНАВАНИЯ РЕЧИ И СОВРЕМЕННЫЕ РЕЧЕВЫЕ ТЕХНОЛОГИИ

Рассматриваются основные существующие методы систем распознавания речи, история изучения, их ситуативное применение, преимущества и недостатки существующих решений, а также сферы применения речевых технологий.

Введение

Исследование методов распознавания речи и инновационных речевых технологий является ключевым направлением в развитии компьютерной лингвистики и обработки естественного языка. Современные алгоритмы машинного обучения в сочетании с глубокими нейронными сетями позволяют достичь высокой точности и скорости распознавания речи, превосходя человеческие способности.

Цель данной статьи заключается в рассмотрении основных методов и подходов к распознаванию речи, оценке их эффективности и перспектив в области речевых технологий. Будут рассмотрены как классические, так и современные подходы, включая использование нейронных сетей, глубокого обучения и искусственного интеллекта в целом.

Данное исследование будет иметь важное практическое значение, поскольку развитие речевых технологий может значительно улучшить коммуникацию между людьми и машинами, обеспечить более удобный и эффективный доступ к информации, помочь людям с ограниченными возможностями и повысить качество жизни общества в целом.

История появления технологии распознавания речи

Автоматическое распознавание речи — технология обработки голоса и перевода аудио в текст. Она появилась в 1952 году, но стала доступной и эффективной только с развитием машинного обучения.

Далее, в 70-х годах программа DARPA Speech Understanding Research (SUR) с 1971 по 1976 год была одной из самой большой в истории распознавания речи, она отвечала за систему «Hagpy» Университета Карнеги Меллона. «Hagpy» понимала 1011 слов, что является средним словарным запасом трехлетнего ребенка.

В следующей декаде благодаря новым подходам и технологиям словарный запас подобных систем вырос с нескольких сотен до нескольких тысяч слов и имел потенциал распознавания неограниченного количества слов.

В 90-х годах приложения «научились» распознавать обычную человеческую речь в обычном темпе (100 слов в минуту). Но для этого было необходимо тренировать программу перед использованием в течение 45 минут.

После 2000-х годов, технологии распознавания речи вышли на новый уровень, благодаря облачным сервисам хранения данных, происходил крупномасштабный анализ данных для поиска совпадений между словами, тем самым улучшилось качество распознавания речи.

Сферы применения технологий распознавания речи

1. Здравоохранение.

Медицинская документация: может использоваться для автоматического преобразования речи медицинских специалистов в электронную медицинскую документацию. Это позволяет сократить время, затрачиваемое на заполнение форм и отчетов, и повысить точность записей.

2. Бытовая техника и персональный компьютер.

Сегодня можно управлять голосом различными устройствами: выключателями, голосовым помощником «Алиса», системами освещения и гаджетами. Также можно обучить свой компьютер распознавать ваш голос

3. Образование.

Транскрипция лекций: позволяет автоматически преобразовывать речь преподавателей в текстовую форму, что упрощает создание транскрипций лекций. Это полезно для студентов, которые могут быстро получить текстовую версию лекций для повторения и изучения материала.

4. Транспорт и логистика.

Голосовые помощники в автомобилях: используется в автомобильных системах, позволяя водителям управлять функциями автомобиля голосовыми командами. Это обеспечивает удобство и безопасность, позволяя водителям сосредоточиться на дороге.

5. Расшифровка аудиозаписей и видеозаписей.

Используется для подготовки отчетности по результатам собеседований, расшифровки записей встреч, выступлений.

Существующие методы распознавания речи

1. Метод на основе шаблонов

Методы на основе шаблонов в распознавании речи используют заранее записанные шаблоны для сравнения с входящим звуком и определения наиболее подходящего соответствия. Эти методы часто используются в системах распознавания ограниченного словаря или ограниченного набора голосовых команд.

Общий процесс работы методов на основе шаблонов:

1. Обучение
2. Извлечение
3. Хранение шаблонов
4. Распознавание
5. Решение о сопоставлении

2. Скрытые марковские модели (HMM)

Скрытые марковские модели (HMM) – это статистические модели, используемые для моделирования последовательностей наблюдений, где каждое наблюдение связано с некоторым скрытым состоянием. Они нашли широкое применение в области распознавания речи, распознавания рукописного текста, биоинформатики и других областей.

HMM состоит из нескольких компонентов:

1. Набор скрытых состояний
2. Наблюдения
3. Матрица переходов между скрытыми состояниями
4. Матрица вероятностей наблюдений
5. Начальное распределение

Путем обучения HMM на размеченных данных можно научить модель распознавать шаблоны и закономерности в последовательностях наблюдений.

Одним из применений скрытых марковских моделей является распознавание речи, где модель используется для определения наиболее вероятной последовательности фонем или слов на основе входных аудиофрагментов. В биоинформатике HMM применяются для предсказания структуры белков, анализа геномных последовательностей и поиска генных паттернов.

Скрытые марковские модели являются мощным инструментом для моделирования последовательностей данных и находят применение в различных областях, где необходимо анализировать временные или последовательные зависимости между наблюдениями.

3. Глубокие нейронные сети

Глубокие нейронные сети (DNN) – это класс нейронных сетей, состоящих из множества слоев нейронов, каждый из которых связан с нейронами предыдущего и последующего слоя. Они являются ключевым инструментом в области машинного обучения и искусственного интеллекта, и нашли широкое применение в распознавании речи, компьютерном зрении, обработке естественного языка и многих других областях.

Применение глубоких нейронных систем в распознавании речи позволяет значительно повысить точность распознавания и снизить уровень ошибок по сравнению с традиционными методами. Это открывает новые возможности для разработки голосовых помощников, систем автоматизации обслуживания клиентов и других инновационных технологий.

Одним из наиболее известных и широко используемых примеров глубоких нейронных сетей в распознавании речи является система распознавания голоса от компании Google, которая использует технологию глубокого обучения для обеспечения более точного и эффективного распознавания речи.

Благодаря непрерывному развитию и совершенствованию глубоких нейронных сетей, возможности в области распознавания речи становятся все более широкими, открывая новые перспективы для различных отраслей и повседневной жизни людей.

4. Реккурентные нейронные сети

Рекуррентные нейронные сети (RNN) – это класс нейронных сетей, который обрабатывает последовательности данных с учетом их последовательной природы. Они имеют возможность запоминать информацию о предыдущих входах и использовать эту информацию при обработке последующих входов. Это делает их особенно подходящими для работы с последовательными данными, такими как тексты, временные ряды и речь.

Одним из основных применений RNN в распознавании речи является моделирование языка. RNN способны адаптироваться к сложным языковым структурам и контекстам, что делает их идеальным инструментом для задач автоматического распознавания речи. Они могут использоваться для распознавания слов, фраз и даже целых предложений.

Кроме того, рекуррентные нейронные сети могут быть использованы для улучшения качества распознавания голоса и синтеза речи. Они способны учитывать контекст и интонацию, что позволяет

им работать более эффективно в условиях шума и различных акцентов.

В целом, использование рекуррентных нейронных сетей в задачах распознавания речи позволяет получить более точные и надежные результаты, обеспечивая высокую производительность и эффективность системы распознавания речи.

5. Энергетические методы

Энергетические методы часто используются в комбинации с другими методами распознавания речи, такими как скрытые марковские модели или нейронные сети, для улучшения точности распознавания. Использование энергетических методов может помочь учесть различные особенности речи и акустическую среду, что может повысить эффективность системы распознавания речи в различных условиях.

Энергетические методы являются одним из ключевых подходов к анализу речи. Они основаны на измерении энергии звука в различных частотных диапазонах и временных интервалах, что позволяет выделить основные характеристики речи, такие как интенсивность, тембр, частота и длительность звуков.

Благодаря использованию энергетических методов в комбинации с другими техниками, такими как скрытые марковские модели или нейронные сети, можно добиться более точного и надежного распознавания речи. Например, скрытые марковские модели могут использоваться для моделирования последовательности фонем в речи, а нейронные сети - для обучения системы на большом объеме данных и улучшения ее способности к обобщению.

Энергетические методы также могут быть приспособлены под конкретные условия использования, что делает их эффективными в различных средах и задачах. Например, адаптация параметров алгоритмов к шумным или реверберирующим средам может помочь улучшить качество распознавания в таких условиях.

Таким образом, энергетические методы играют важную роль в современных системах распознавания речи, позволяя учитывать разнообразные особенности звучания и контекста, что в конечном итоге способствует повышению качества и эффективности таких систем.

Применимость рассмотренных методов

Метод на основе шаблонов применяется в системах голосового управления, распознавании речи в мобильных устройствах. Преимущества метода на основе шаблонов включают быструю обработку данных, высокую точность распознавания и устойчивость к

шуму в окружающей среде. Такой метод также не требует больших вычислительных ресурсов и может быть использован для распознавания речи в реальном времени.

Метод на основе скрытых марковских моделей применяется в областях, где требуется моделирование последовательности событий с помощью вероятностных моделей.

Такой подход является эффективным, т.к. имеет небольшое количество параметров, но его использование требует тщательного подбора параметров и большого объёма данных для обучения.

Метод на основе глубоких нейронных сетей часто применяется в системах управления. С помощью такого подхода получается высокая точность распознавания речи, но необходимы высокие требования к вычислительному устройству.

Метод на основе реккурентных НС применяется в таких задачах как распознавание речи в реальном времени. Такой подход является более точным, но также имеет недостатки: при обработке длинных последовательностей происходит потеря информации.

Энергетические методы используются для измерения и оценки энергии звуковых колебаний, что позволяет выделять основные особенности речевого сигнала и улучшать точность его распознавания.

Кроме того, энергетические методы используются для устранения шума и искажений в речевых сигналах. Алгоритмы фильтрации и преобразования позволяют улучшить качество записи и повысить точность распознавания речи даже в условиях высокого уровня шума.

Заключение

Таким образом, распознавание речи играет важную роль в различных областях, таких как: информационные технологии, автоматизация, медицина. В настоящее время, благодаря нейронным сетям, технологии распознавания речи всё больше совершенствуются.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Зверев Д.В., Цыкунов Ю.В. Распозавание и синтез естественного языка / Зверев Д.В., Цыкунов Ю.В. // 2017. – С. 27-38.

2. Иванов И.И. Распознавание речи / Иванов И.И. // Высшая школа. – 2000. – С. 131-145.

3. Комиссаров В.А. Распознавание и синтез речи. Методы и модели / В. А. Комиссаров // Машиностроение. – 1991. – С. 81-88.

4. Литвинов В.А., Петров С.К. Моделирование систем распознавания и синтеза речи / Литвинов В.А., Петров С.К. // – 2013. – С. 72-79.

УДК 004.8

САБЛИНА В.А., ПОГОДИН А.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ПОДХОДЫ К АНАЛИЗУ ДЕЙСТВИЙ ПОЛЬЗОВАТЕЛЕЙ ВЕБ-САЙТОВ ДЛЯ ВЫЯВЛЕНИЯ НЕСТАНДАРТНЫХ ШАБЛОНОВ ПОВЕДЕНИЯ

Предлагаются подходы, позволяющие выявить нестандартные шаблоны поведения пользователей веб-сайтов, которые могут указывать на работу автоматизированных программ.

В мире веб-разработки и онлайн-бизнеса веб-боты играют все более значительную роль. Однако не все веб-боты действуют в благих целях. Автоматизированные программы, взаимодействующие с веб-сайтами, могут нанести серьезный ущерб как владельцам веб-ресурсов, так и пользователям. Обнаружение таких ботов становится все более важной задачей для защиты веб-сайтов и их пользователей от различных видов мошенничества, включая похищение данных, DDoS-атаки, неправомерные авторизации, «скликивание» и другие несанкционированные виды активности.

В работе исследуются методы определения подозрительной активности пользователей на веб-ресурсах с акцентом на анализ их поведения. Основное внимание уделяется описанию подходов к созданию алгоритмов, которые могут автоматически выявлять аномальные или необычные шаблоны поведения, несвойственные реальным пользователям.

На сегодняшний день существуют различные способы защиты веб-ресурсов. Данные способы можно разделить на две группы. Первая группа ограничивает вход на сайт, вторая группа анализирует активность после предоставления доступа.

Яркий пример способов защиты первой группы – использование CAPTCHA® – полностью автоматизированного публичного теста Тьюринга для различения компьютеров и людей. В наиболее распространенном варианте CAPTCHA® – пользователь вводит символы, изображенные на рисунке, зачастую указанные с добавлением помех или полупрозрачности с целью создать затруднения для машинного распознавания текста. Таким образом, на этапе входа происходит фильтрация реальных пользователей и автоматизированных программ. Несмотря на высокую эффективность данного способа защиты, различные исследования приводят

статистику, согласно которой использование CAPTCHA® снижает конверсию веб-сайта [1]. Многие пользователи отказываются от посещения страницы, которая защищена данным тестом.

Учитывая недостаток использования CAPTCHA® можно сделать вывод о том, что предпочтительнее сделать сайт открытым для доступа и использовать методы защиты второй группы, нацеленные на анализ поведения, а также на анализ информации о пользователе, которую можно получить с использованием функций языка программирования JavaScript [2]. Для проведения экспериментальных исследований различных подходов к анализу действий пользователей веб-сайтов для выявления нестандартных шаблонов поведения в ходе проведения работы было разработано специальное программное обеспечение на указанном языке программирования. Разработанное программное обеспечение использовалось для анализа рассмотренных далее событий, связанных с возможными действиями пользователей веб-сайтов.

Для того чтобы разобраться в механизмах работы веб-ботов и впоследствии разработать методы защиты, необходимо определить с помощью каких инструментов и технологий они создаются. Наиболее популярными программами, способными создать автоматические запросы для веб-сайтов являются программы автоматизации веб-браузеров, такие как Selenium WebDriver [3], Puppeteer, Katalon Studio и другие программы, наиболее популярной из которых является Selenium. Данный инструмент поддерживает многие языки программирования, такие как Java, Python, C#, Ruby. Selenium способен выполнять следующие действия на веб-странице: поиск элементов, ввод текста, очистка форм, заполнение форм, имитация событий связанных с управлением мышью, прокрутка страницы и другие действия. Запросы, совершенные с помощью подобного программного обеспечения, схожи с запросами реальных пользователей, однако существуют различия, которые и являются своего рода маркерами сомнительной активности.

Все события, происходящие во время работы на странице веб-сайта можно разделить на 3 группы: события авторизации, события навигации и события ввода данных.

Событие авторизации: после входа пользователя на сайт с помощью кода JavaScript может быть получена информация о том, с какого устройства осуществлен вход, – параметры разрешения экрана устройства, IP-адрес устройства, язык, информацию об операционной системе.

События навигации: к этой группе можно отнести события перемещения курсора мыши, событие прокрутки страницы, событие кликов по странице и события перемещения между окнами.

События ввода данных представляют собой события изменения полей, которые можно редактировать.

Маркеры сомнительной активности событий авторизации: в большинстве случаев вход на сайт, который осуществляют автоматизированные программы, происходит с помощью Headless Chrome [3] – браузера без графического интерфейса. Информация о разрешении экрана, полученная с помощью кода JavaScript будет отсутствовать. В некоторых случаях информация о пользователе ограничена лишь одним IP-адресом. Пользователям с отсутствием информации о разрешении экрана, названии операционной системы, языка для устройства с которого осуществлен вход, присваивается маркер сомнительной активности.

Рассмотрим маркеры сомнительной активности событий навигации. Перемещение курсора мыши, клики по элементам, прокрутка страницы являются неотъемлемой частью взаимодействия с веб-ресурсом. Для того чтобы выявить нестандартные шаблоны поведения, стоит описать действия, характерные для реальных пользователей:

– траектория движения курсора реального пользователя не является прямолинейной;

– во время движения курсора переход по координатам экрана происходит последовательно, отсутствуют дискретные переходы из одной области в другую;

– события перемещения курсора происходят в разное время (нет повторяющихся временных интервалов);

– время между кликами различно;

– клики по элементам происходят в разных точках элемента (не только по центру элемента).

Учитывая вышесказанное, можно выявить следующие основные маркеры сомнительной активности и алгоритмы их определения.

Анализируем траекторию движения курсора мыши и события кликов. Для Selenium характерно моментальное перемещение курсора от элемента к элементу. В качестве примера запрограммируем Selenium на перемещение курсора от одного элемента веб-страницы к другому, запишем координаты перемещения X и Y в массив. Количество элементов в данном массиве составляет 2 значения – вектора с координатами X и Y центров элементов веб-страницы. В случаи перемещения курсора реальным пользователем массив будет

заполнен всеми координатами, через которые проходит курсор. Фиксация значений координат с некоторым дискретным шагом больше единицы определенно говорит о необходимости присвоения маркера сомнительной активности. Запросы можно усложнить, используя функцию Selenium MoveByOffset(x,y), которая позволяет автоматизировать перемещение курсора с определенным установленным смещением по осям *x* и *y*. С помощью выполнения данной функции в цикле можно добиться того, что траектория движения курсора будет похожа на траекторию действий реальных пользователей. Визуализация траектории представлена на рисунке 1.

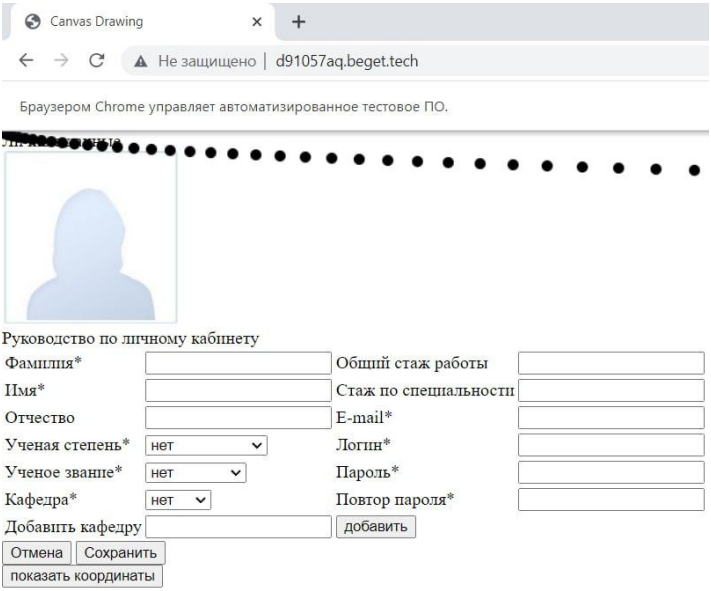


Рисунок 1 – Дискретное перемещение курсора мыши

Точками на рисунке указаны координаты, в которые переместился курсор. Во время анализа становится понятным, что перемещение курсора было дискретным, а не поступательным. Маркер сомнительной активности может быть присвоен. С помощью функции MoveByOffset (x,y) [3] можно создать поступательное движение курсора с шагом единица для *X* и с шагом единица для *Y*, однако в таком случаи движение курсора станет прямолинейным, что хорошо видно на рисунке 2. Такая траектория курсора также свидетельствует о

том, что шаблон поведения нестандартный для реального пользователя.

Важным фактором в оценке поведения является время. Фиксируя время между событиями, можно установить, что для автоматизированных программ свойственно выполнять действия с одинаковыми интервалами по времени. Например, если запрограммировать Selenium на перемещение от одного объекта к другому, выполняя при этом клики, то время между кликами будет одинаковым (с погрешностью до 100 мс). Такую активность можно признать сомнительной. Однако в этом примере стоит отметить, что время легко можно сделать различным, добавив задержку, задаваемую функцией от случайной величины, в выполняющий поток.

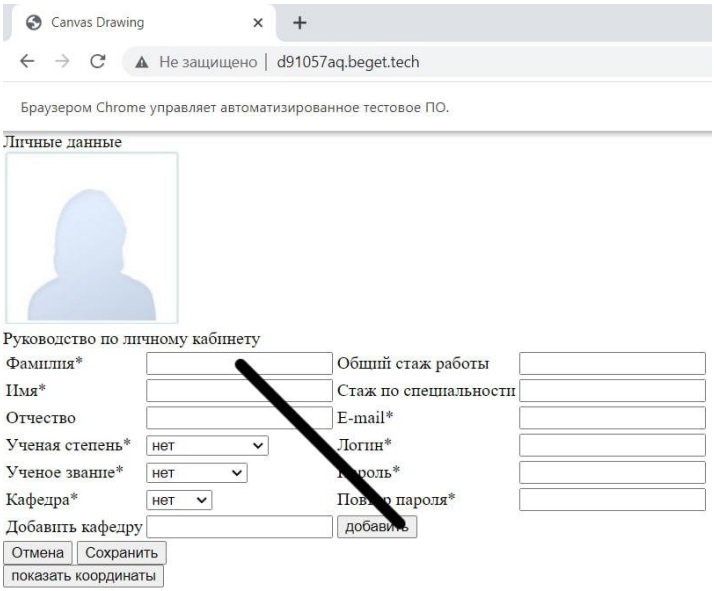


Рисунок 2 – Прямолинейное перемещение курсора мыши

Следующим важным фактором, который позволит определить сомнительную активность, являются координаты кликов, которые производят программы автоматизации по элементам веб-страницы. Клик приходится исключительно по центру элемента с точностью до одного пикселя, что, в свою очередь, практически невозможно для человека. С помощью кода JavaScript можно выявить подобные

события и присвоить маркер сомнительной активности для такого пользователя.

Далее разберем события третьей группы – события ввода данных. Анализ таких действий пересекается с событиями навигации. Так, для заполнения текстовых полей необходимо обратиться к полю, выполнив одно из трех действий: навести курсор и нажать левую кнопку мыши, использовать клавишу «Tab» для перехода, выполнить переход, используя клавиши управления курсором «Up», «Down», «Left» и «Right». Если запрограммировать с помощью Selenium заполнение форм без дополнительных настроек, поля будут заполнены автоматически. Фиксируя событие «Change» [2] с помощью JavaScript и проверяя отсутствие событий перемещения курсора и нажатий клавиш, необходимых для перехода, можно сделать вывод о том, что активность такого пользователя сомнительная.

Важным событием является событие очистки формы. Для его реализации реальный пользователь должен выполнить одно из нескольких действий: выделить весь текст и нажать на клавишу «Backspace» или клавишу «Delete» или вырезать выделенный текст. Возможно многократное нажатие на клавишу «Backspace», выполняя при этом также обращение к форме. В этом примере задействовано множество событий, определяемых кодом JavaScript, таких, как «Dblclick», «Mousedown», «Mousemove», «OnKeyDown», «Change» [2] и другие события. Selenium, запрограммированный на очистку форм, выполнит автоматическую очистку, генерируя только одно событие «Change». Конечно, можно усложнить запросы, однако анализируя подобные действия, можно с легкостью обнаружить автоматизированную программу с базовыми настройками.

Существуют и другие события, отличающие реальных пользователей от автоматизированных программ. Объединяя функции, написанные на языке JavaScript, определяющие маркеры сомнительной активности, можно значительно снизить опасность взаимодействия веб-сайта с автоматизированными программами, блокируя их действия.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. CAPTCHA: убивая конверсию [Электронный ресурс]. – URL: https://habr.com/ru/company/variti/blog/495902/.

2. Современный учебник JavaScript [Электронный ресурс]. – URL: https://learn.javascript.ru/.

3. Selenium Web Driver для начинающих [Электронный ресурс]. – URL: https://smartiqa.ru/blog/selenium-webdriver-basics.

УДК 378.14

САПРЫКИНА А.О.
Рязанский государственный университет имени С.А. Есенина

ЭЛЕКТРОННОЕ ПОРТФОЛИО В КОНТЕКСТЕ ТЕОРИИ ПРЕОБРАЗУЮЩЕГО ОБУЧЕНИЯ МЕЗИРОВА

В данной статье рассматривается технология электронного портфолио применимо к образовательному процессу, анализируется ее интеграция с теорией преобразующего обучения Мезирова. Рассматривается специфика процесса оценивания образовательных достижений с использованием электронного портфолио.

Текущее состояние исследований портфолио и их использования в онлайн-образовании предполагает, что анализ использования портфолио в дистанционном обучении требует применения сложных теоретических основ. При этом одним из наиболее релевантных подходов к исследованию является анализ с точки зрения теории преобразующего обучения Мезирова. Теория преобразующего обучения Мезирова – это конструктивистская теория, которая фокусируется на когнитивной реструктуризации и интеграции опыта, действия и рефлексии.

Согласно Мезирову [7], преобразующее обучение – это процесс, который влияет на изменения посредством изменения системы отсчета. Обучающиеся развивают и приобретают целостный опыт, который служит основной системой отсчета, определяя смысл мира. Другими словами, системы отсчета – это эмпирическая линза, через которую обучающиеся развивают свое видение окружающей реальности. Они оказывают избирательное воздействие на чувства, восприятие и действия, приводя к отказу от идей, которые не отражают предубеждения о мире, и принятию тех идей, которые позволяют реализовать ментальные и поведенческие намерения для собственной выгоды. Основываясь на теории преобразующего обучения Мезирова, люди используют критическую рефлексию для преобразования своих систем отсчета, они склонны критически относиться к предположениям, сделанным другими людьми или сформированным под влиянием изменений в политической, социальной, экономической или культурной среде. Следовательно, саморефлексия является фундаментальной движущей силой личностных преобразований. Здесь следует объяснить связь между саморефлексией, личностной трансформацией и использованием оценивания посредством портфолио в дистанционном образовании. Поскольку обучение призвано вызвать позитивные изменения в

мировоззрении обучающихся, и эти изменения, согласно Мезирову, происходят посредством критической саморефлексии, то портфолио идеально вписываются в современные условия предоставления знаний в сфере высшего образования. Оценивание посредством портфолио может стать тем самым средством саморефлексии, которое способствует личностной трансформации.

Согласно Мезирову, существует четыре основные деятельности внутри образовательного процесса.

Первый вид деятельности состоит в стремлении подкрепить свою точку зрения, находя и используя доступные доказательства. Эти доказательства в конечном итоге укрепят существующие этноцентрические предубеждения и усилят интенсивность точки зрения.

Второй вид деятельности состоит в поиске нового. Третий вид деятельности заключается в трансформации имеющихся взглядов и знаний такими способами, которые устраняют существующие неправильные представления. Четвёртый вид деятельности достаточно редок и заключается в полной трансформации убеждений и отношения к объективной реальности.

Теория преобразующего обучения Мезирова имеет далеко идущие последствия для понимания процесса дистанционного образования, роли и места оценивания портфолио, а также влияния, которое портфолио может оказать на самоэффективность учащихся и понимание содержания. Обучение – это многогранный процесс, который принимает множество форм; в результате анализ взаимосвязи между обучением и оценкой может стать чрезвычайно сложным. Чтобы обеспечить эффективность обучения, преподаватели должны помогать обучающимся осознавать и критически относиться к своим собственным и чужим предубеждениям, верованиям и предположениям. Обучающимся необходимо осознать свои знания, сильные стороны и недостатки, а затем переосмыслить проблемы, с которыми они сталкиваются в жизни, посредством преобразующего обучения и саморефлексии. В этом смысле портфолио как форма оценивания может способствовать развитию критического самоанализа.

Теория Выготского добавит актуальности теоретической базе Мезирова. Основываясь на теории обучения Выготского, разум – это не сложная сеть общих способностей, таких как наблюдение, внимание, память, суждение и так далее, а набор специфических способностей [1]. Выготский предполагает, что каждая из этих специфических способностей обособлена от других и, более того,

развивается независимо. Для Выготского обучение – это нечто большее, чем развитие мыслительных способностей. Суть обучения заключается в приобретении многочисленных мыслительных способностей по многим различным вопросам. Обучение не меняет направленность и внимание индивидов, но учит их расширять свое мировоззрение и сосредотачиваться более чем на чем-то одном. Следовательно, любое улучшение, вызванное обучением одной способности, повлияет на другие способности только в той мере, в какой они взаимосвязаны. Теория Выготского подразумевает, что оценивания посредством портфолио могут измерять изменения в индивидуальных способностях в результате обучения, но вряд ли могут измерить их взаимосвязь, поскольку считается, что эти способности не связаны между собой. Одновременно именно с помощью оценивая портфолио можно организованно измерить и оценить изменения в различных способностях под влиянием обучения. История портфолио как формы оценивания и его популярность в аудиторном обучении позволяют предположить, что портфолио является примером уникального, сложного и чрезвычайно эффективного механизма оценки прогресса учащихся в достижении целей обучения.

Множество типов портфолио и приложений создает представление об их универсальной применимости ко всем курсам и моделям обучения. Однако подобное заявление, по меньшей мере, не совсем верно, поскольку использование портфолио в качестве инструмента оценки обусловлено его природой, происхождением и первоначальной целью. Объективно, оценки портфолио отражают и вписываются в атмосферу совместного конструктивизма в образовании, которая смещает акцент с индивидуальной ответственности за обучение на совместный взгляд на образовательный процесс [6]. Портфолио лучше всего использовать в условиях, когда учащимся предоставляется свобода определять наиболее важные аспекты, влияющие на прогресс обучения, и действовать, чтобы внести изменения и улучшить свой прогресс. Портфолио как инструмент оценивания вписывается в среду, где критическое исследование, саморефлексия, реорганизация и переосмысление знаний являются важнейшими компонентами [6]. Портфолио облегчают связь между индивидуальными значениями и контекстами, в которых они создаются.

Оценивание достижений обучающихся, уровня их реальных знаний и степени усвоения учебного материала является одной из самых сложных задач образования. Традиционная система оценивания

основана на выставлении отметок как в течение всего срока обучения, так и итоговых при сдаче итоговых испытаний. Зачастую успехом образовательной программы и показателем ее эффективности становится получение обучающимся отметки выше некого среднего значения, но такой взгляд на задачу оценивания во многих является недостаточно проработанным и всеобъемлющим.

Оценивание представляет собой определение результативности обучения в целом и преподавания в частности. Оно направлено на определение эффективности образовательного процесса. Оценивание заключается не только в проведении тестов для проверки знаний учащихся, но и в выполнении заданий, позволяющих оценить методы обучения и полученные результаты, сформированность тех или иных практических навыков. На основе этих оценок принимается решение об уровне качества образования и верности выбранной траектории личного и профессионального роста для обучающихся. Полученные таким образом результаты могут и должны быть использованы для улучшения учебного процесса и повышения качества обучения.

Электронное портфолио на данный момент не всегда рассматривается как надежный инструмент образования. Например, согласно опросу ВЦИОМ [3] 2021 года, больше двух третей родителей в России (~69%) считают электронное портфолио ненадежным и опасаются, что хранящиеся в нем данные могут быть украдены или стерты.

Использование в образовательном процессе электронного портфолио способно повысить мотивацию обучающихся посредством структурирования и прозрачности устанавливаемых целей и задач. Можно выделить два компонента оценки [4] способностей, включая фактические данные, сохраненные в портфолио, и критические комментарии к достигнутым результатам и предполагаемым результатам обучения. Использование при их анализе формирующего оценивания может способствовать повышению эффективности обучения в целом [2, 5]. Электронное портфолио даёт пользователям возможность анализировать и иллюстрировать свой рост в рамках дискурса и стандартов сообщества. Однако несмотря на большой потенциал, наглядность и эффективность данной технологии, необходимо решить множество проблем, связанных с внедрением в образовательный процесс данной технологии, разработкой методик работы с ней и обеспечением ее надежности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Выготский Л.С. Мышление и речь. – Москва: Лабиринт, 2007. – 350 с.
2. Сапрыкина А.О. Формирующее и комплексное оценивание как инструмент поддержания успеваемости в высшей школе // Сборник научных трудов XI Международной конференции школьников, студентов, аспирантов, молодых ученых «Ресурсоэффективные системы в управлении и контроле: взгляд в будущее». – Томск, ТПУ, 2022. – С. 224-228.
3. Школьное портфолио: что думают родители? https://wciom.ru/analytical-reviews/analiticheskii-obzor/shkolnoe-portfolio-chto-dumajut-roditeli
4. Baume D. Enhancing Staff and Educational Development. – London: Routledge. – 2004. – 224 p.
5. Fuller E. J., Hollingworth L. Evaluating Principal Preparation Programs Based on Placement Rates: Problems and Prospects for Policy Makers // Journal of Research in Leadership Education. – 2016. – Vol. 11, Issue 3. – pp. 237-271.
6. Klenowski V., Askew S. Portfolios for Learning, Assessment and Professional Development in Higher Education // Assessment & Evaluation in Higher Education. – 2006. – Vol 31, Issue 10.
7. Mezirow J. Transformative Learning: Theory to Practice // New Directions for Adult and Continuing Education, 1997. – Vol 74. – Pp. 5-12.

УДК 004.4

СЕЛЕЗНЕВ А.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

РАЗРАБОТКА ГРАФИЧЕСКИХ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ jPCT

В статье рассматривается программная реализация трехмерной компьютерной игры с использованием библиотеки jPCT на языке программирования Java. Приведен пример работы программы.

Компьютерная графика стала неотъемлемой частью нашей повседневной жизни. Во многих сферах мы можем встретить ее различные проявления: интерфейсы в ПО, видеоролики, компьютерные игры и т.д. В современном обществе игры стали больше, чем развлекательный элемент. Кто-то делает их для заработка

денег, кто-то для реализации своих возможностей. В сюжет игр стали вкладывать больше смысла, управление и графика также выросли за долгое время. Но все еще остаются, так называемые, казуальные проекты, которые нужны лишь для развлечения и не несут в себе особой культурной ценности. Однако они развивают твои различные навыки, например, точность, логическое мышление и т.д.

В настоящий момент разработку видеоигр можно вести на множестве языков программирования, причем какого-то универсального не существует. Базовыми считают C++, C# и Java. Для начинающих больше подойдут два последних языка. Это языки ООП, а значит они отлично подходят для разработки крупномасштабных приложений, включая игры, однако большее предпочтение лучше отдать Java, так как C# в основном зависит от Unity – довольно старого движка для разработки приложений.

Java имеет достаточное количество различных движков для создания игровых проектов. Одним из них является jPCT - это свободный 3D-движок, который позволяет использовать OpenGL и Software рендеры на выбор. Он подходит для разработки мощных 3D-приложений для компьютеров, Интернета и Android. jPCT разрабатывается с 2002 года.

Основные функции, предоставляемые библиотекой:

- загрузка файлов типа 3DS, OBJ, MD2, ASC и XML
- поддержка анимации
- рассеянное и зеркальное освещение
- встроенные примитивы, такие как конусы, кубы, сферы и т.д.
- обнаружение столкновений (коллизия)
- рендер текстур

В начале создания видеоигры всегда ставятся ряд задач, определяющих все дальнейшую разработку. Первостепенную важность имеет вопрос выбора жанра игры и основных идей, которые будут реализованы в проекте. На данный момент существует множество жанров, начиная с Adventure, заканчивая ролевыми и файтингами. Выбор остановился на Adventure, который был одним из ранних жанров игр, и не требует от игрока постоянно сражаться с противниками и погружаться в сложные истории. Основной же задачей будет исследование особняка (в примере будет представлено две комнаты). Управление будет поддерживать только клавиатуру.

Следующим шагом стал выбор типа камеры. Восприятие игр во многом зависит от ракурса, с которого наблюдает игрок за происходящими событиями. В основном различают три типа игровых камер:

- с видом от третьего лица с фиксированным ракурсом: неподвижная камера, закрепленная на одной позиции, однако способная поворачиваться в стороны или приближать изображение;
- от третьего лица с динамическим ракурсом: подвижная камера, непривязанная к конкретной точке: она либо следует за персонажем, либо ей можно свободно управлять;
- от первого лица.

После некоторого изучения, был выбран первый тип камеры, который часто использовался в старых играх, а в частности в жанре Adventure. Ориентиром при разработке камеры в своем проекте послужила игра Fatal Frame 2, в которой была прикреплена к определенной позиции, однако постоянно следила за местоположением игрока (рисунок 1).

Рисунок 1 – Пример камеры из игры Fatal Frame 2

Следующим этапом разработки является создание игровых локаций и персонажа. Для этого был выбран графический редактор Cinema 4D. Хотя это и коммерческий продукт, однако есть бесплатная пробная версия на 2 недели, которую можно использовать для разработки объектов. Cinema 4D является универсальной комплексной программой для создания и редактирования двух- и трехмерных эффектов, и объектов. Позволяет рендерить объекты по методу Гуро.

После краткого изучения функционала программы были разработаны основные локации и различные объекты на них (рисунок 2).

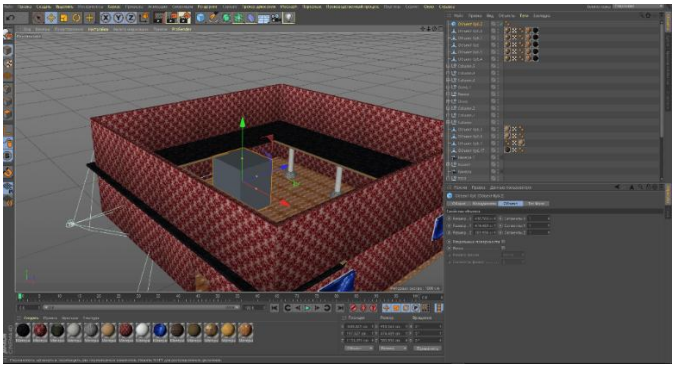


Рисунок 2 – Создание локации

Модель персонажа, как и текстуры, были взяты из Интернета. Все они имеют свободное распространение. Однако потребовалась полная доработка анимаций для модели персонажа. Обработка текстур была минимальна и проделана в бесплатном графическом редакторе Paint.

Следующий шаг – разработка программного обеспечения. По итогу написания кода получилось 4 класса: Core.java, Cam_obj.java, InputCore.java, Objects.java. Диаграмма полученных классов представлена на рисунке 3.

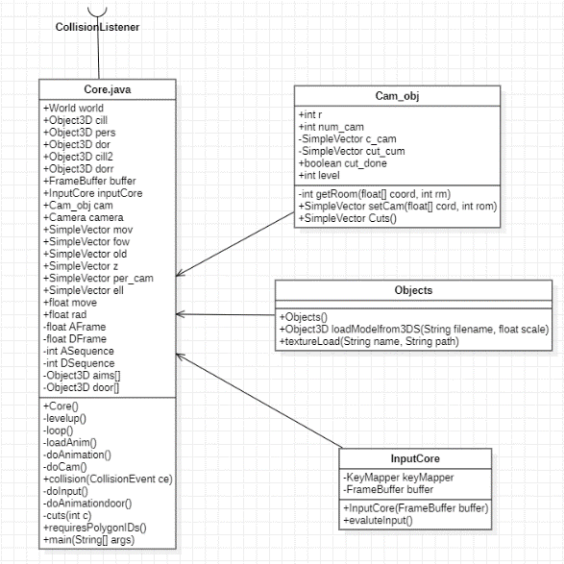


Рисунок 3 – Диаграмма классов

Опишем каждый класс более подробнее.

В классе InputCore.java реализовано основное управление персонажем. Для этого используются методы специального класса KeyMapper, который позволяет считывать нажатие клавиш. Метод класса InputCore опрашивает нажатие клавиатуры и изменяет значение логических переменных, отвечающих за определенные клавиши.

В классе Objects.java реализована загрузка моделей и текстур. Для этого используются методы нескольких классов jPCT:

- Matrix - базовый класс jPCT для работы с матрицами 4x4;
- Texture – управление текстурами - растровыми изображениями шириной / высотой 2^x и глубиной цвета 24 bpp;
- TextureManager – объект для хранения и извлечения текстур;
- Loader - несколько статических методов загрузки файлов объектов и анимации, а также некоторые очень простые методы загрузки текстовых файлов;
- Object3D – класс для работы с трехмерными объектами: задание коллизии, анимации, видимости и т.д.

В классе Cam_obj.java реализована работа с камерой, ее перемещение и работа в катсценах. Реализовано следующим образом: в зависимости от положения персонажа, получается номер комнаты, который в дальнейшем используется для получения вектора камеры в зависимости от нахождения игрока и помещения. Сама же камера представляет отдельный класс jPCT – Camera, которой для перемещения требуется вектор (координаты по X, Y и Z).

В классе Core.java реализована вся основная работа игры. Подробнее рассмотрим метод loop(), реализованный в нем. После загрузки всех объектов в мир, буфер остается пустым, поэтому на экране ничего нет. Первым делом вызывается метод cuts(0), во время которого помимо передвижения камеры также подгружаются некоторые модели на экране. После проигрывания катсцены начинается цикл с предусловием while (`!org.lwjgl.opengl.Display.isCloseRequested()`), который смотрит, запрошено ли закрытие окна или нет.

Далее вызываются методы, обрабатывающие положение камеры в мире, передвижение персонажа и его анимацию. После этого буфер очищается, заполняя себя черным цветом, затем происходит рендер сцены и его зарисовка в буфер. Буфер обновляется и вызывается метод `buffer.displayGLOnly()`, который позволяет OpenGL выводить изображение на экран. При выходе из цикла мы запрещаем буферу рендерится при помощи метода `buffer.disableRenderer (IRenderer.RENDERER_OPENGL)`, после чего мы избавляемся от

буфера и выходим из программы. Код программы выглядит следующим образом:

```
private void loop() throws InterruptedException
{
        final long delay = 50;
        long previous = System.currentTimeMillis();
        inputCore = new InputCore(buffer);
        cuts(0);
        //Цикл, в котором происходит рендер
         while
(!org.lwjgl.opengl.Display.isCloseRequested())
     {
       doInput();
       doCam();
       doAnimation();
       doKey();
       buffer.clear(java.awt.Color.BLACK);
       world.renderScene(buffer);
       world.draw(buffer);
       buffer.update();
       buffer.displayGLOnly();
       Thread.sleep(10);
     }
     buffer.disableRenderer(IRenderer.RENDERER_OPENGL);
     buffer.dispose();
     System.exit(0);
}
```

После запуска программы появляется катсцена, реализованная на игровом движке. После ее окончания игроку дается управление над персонажем. В распоряжение дается изучение локации, а также переходить из одной комнаты в другую. Результат работы представлен на рисунках 4 - 6.

Рисунок 4 – Вступительная катсцена

Рисунок 5 – Первая локация

Рисунок 6 – Вторая локация

В будущем планируется улучшение и доработка программы, можно добавить: новые локации, звуковые эффекты, начальное меню, оптимизацию, а также больше интерактивных объектов.

Стоит отметить, что на Java существуют куда более лучшие библиотеки для работы с графическими приложениями, однако jPCT можно считать хорошим началом для погружения в разработку трехмерных игр.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Разработка игр с использованием JPCT. – [Электронный ресурс.] – URL: https://jpct.ucoz.ru.

2. 7 Лучших языков программирования для разработки игр в 2024 году – [Электронный ресурс.] – URL: https://hackr.io/blog/best-programming-language-for-games#best-programming-languages-for-game-development

3. Chris Crawford. The Art of Computer Game Design (англ.). – Berkeley, Calif.: Osborne/McGraw-Hill, 1984.

4. jPCT. – [Электронный ресурс.] – URL: https://www.jpct.net.

5. Kent McQuilkin, Anne Powers. Cinema 4D: The Artist's Project Sourcebook. – Taylor & Francis, 2011.

УДК 004.021

СТЕПЧЕНКОВ А.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ИСПОЛЬЗОВАНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ В ОБУЧЕНИИ НЕЙРОННЫХ СЕТЕЙ

Статья рассматривает использование генетических алгоритмов в обучении нейронных сетей, представляя их эффективные методы оптимизации параметров. Обсуждаются принципы работы алгоритма, его преимущества и недостатки.

За последние десятилетия нейронные сети прошли путь от концепта до мощного инструмента для решения различных задач в области ИИ. Однако их обучение остается достаточно сложной задачей, требующей подбора оптимальных значений, таких как вес ссылки и параметры сети. В последнее время генетические алгоритмы (ГА) стали одним из методов оптимизации параметров нейронных сетей.

ГА основаны на идеях естественного отбора и генетики. Популяции создаются при помощи эволюции и используют операторы: выбор, скрещивание и мутация, для генерации новых поколений. Процесс обучения нейронных сетей с использованием генетического алгоритма можно описать следующим образом:

1. Инициализация популяции – создаем случайную совокупности отдельных нейронных сетей, которая представлена в виде набора параметров, таких как вес ссылки и параметры сети.

```
    private NeuralNetwork[] population;
        public Population(int size, int inputSize, int
outputSize) {
            population = new NeuralNetwork[size];
            Random rand = new Random();
            for (int i = 0; i < size; i++) {
                population[i] = new
NeuralNetwork(inputSize, outputSize);
                population[i].initializeWeights(rand);
            }
        }
```

2. Оценка приспособленности – каждая нейронная сеть из популяции оценивается на основе ее показателей производительности на выборке из данных. В примере использована (MSE) в качестве функции приспособленности.

$$F(x) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Где x – вектор параметров нейронной сети, N – количество обучающих примеров, y_i – истинное значение для i-го примера, $\hat{y}_i$ – предсказанное значение для i-го примера.

3. Селекция – выбираются лучшие индивидуумы из выборки по показателю их приспособленности. Чем он выше, тем больше шанс быть выбранным для создания следующего поколения.

```
public NeuralNetwork[] select(
NeuralNetwork[] population,
 double[] fitnessScores,
 int numSelected) {
        NeuralNetwork[] selected = new
NeuralNetwork[numSelected];
        double[] normalizedFitnessScores =
    normalizeFitnessScores(fitnessScores);
        Random rand = new Random();
        for (int i = 0; i < numSelected; i++) {
            double randomValue = rand.nextDouble();
            double cumulativeProbability = 0.0;
            for (int j = 0; j < population.length;
j++) {
                cumulativeProbability +=
normalizedFitnessScores[j];
                if (randomValue <=
cumulativeProbability) {
                    selected[i] =
population[j].clone();
                    break;
                }
            }
        }
        return selected;
    }
```

4. Кроссовер – выбранные индивидуумы скрещиваются для создания новых потомков. Их веса и структуры сетей будут скомбинированы.

Родитель 1: ABC|DEF

Родитель 2: GHI|JKI

После кроссовера получается потомство:

Потомок 1: ABG\DEF

Потомок 2: GHJ\IKL

5. Мутация – некоторые параметры новых индивидуумов изменяются случайным образом, чтобы обеспечить разнообразие в популяции и предотвратить застревание.

$$w_{новый} = w_{старый} + \Delta$$

Где $w_{новый}$ – новый вес, $w_{старый}$ – старый вес, Δ – случайное приращение.

6. Условие остановки – процесс повторяется до достижения критерия остановки, лимита количества поколений или достижение заданной производительности.

7. Выбор решения – по завершении обучения выбирается лучшая нейронная сеть из последнего поколения как итоговое решение.

Использование генетического алгоритма несет в себе ряд преимуществ:

1. Глобальный поиск – способен обнаруживать глобальные оптимумы в параметрах нейронных сетей.

2. Применимость к большим пространствам параметров – т.к. генетические алгоритмы не требуют аналитических методов для вычисления градиентов.

3. Менее подвержены застреванию – т.к. пользуются естественным отбором и эволюцией.

Не смотря на все преимущества так же присутствуют недостатки:

1. Вычислительная сложность – при работе с большими популяциями и нейронными сетями очень вычислительно затратно.

2. Необходимо ответственно подойти к выбору функции, она должна быть достаточно информативной, чтобы отличать более приспособленных, но не слишком сложной, чтобы не замедлить поиск.

3. Требуется больше времени для сходимости.

В качестве примеров применения, можно выделить:

1. Оптимизация гиперпараметров нейронных сетей, таких как количества слоев, количества нейронов в каждом слое, скорости обучения и т.д.

2. Проектирование архитектуры нейронных сетей включая выбор оптимальной структуры сети и оптимальных функций активации.

3. Обучение нейронных сетей на малом количестве данных. Для этого используется информацию о приспособленности индивидуумов для обучения.

Можно подвести итог, что генетические алгоритмы крайне действенный метод оптимизации при обучении нейронных сетей и

если придерживаться специфики алгоритма, то его применение сильно оптимизирует алгоритм.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Вебер Р., Райдер Д. Генетические алгоритмы в науке, технике и искусстве. – Москва: Солон-Пресс, 2006.
2. Джордж Д. Генетические алгоритмы: исследование и разработка. – Санкт-Петербург: Питер, 2005.
3. Дэвис Л. Адаптивный поиск и распознавание образов. – Москва: Техносфера, 2011.
4. Комаровский Д. И., Комаровская Е. П. Генетические алгоритмы и нейронные сети. – Москва: Московский центр качества, 2010.

УДК 004.021

СТЕПЧЕНКОВ А.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ПРИМЕНЕНИЕ ГИБРИДНЫХ АЛГОРИТМОВ ОПТИМИЗАЦИИ

В статье описаны основные формулы и методы применения гибридных алгоритмов с анализом эффективности и дальнейших перспектив развития.

В наши дни оптимизация играет важную роль практически во всех областях. В особенности, где необходимо найти оптимальные решения для сложных задач. Для этого отлично подходят гибридные алгоритмы, в последние годы они пользуются особенной популярность в связи с тем, что они могут комбинировать различные алгоритмы, тем самым достигая лучшей оптимизации.

Таким образом гибридные алгоритмы преодолевают ограничения методов по отдельности. В качестве комбинирования подойдут разные методы, такие как генетические алгоритмы, методы оптимизации на основе поколений, методы оптимизации на основе роя частиц, методы оптимизации на основе симуляции отжига и другие.

Главным преимуществом таких алгоритмов будет использование комбинации различных стратегий, тем самым обеспечивая наиболее глубокий и эффективный поиск. Это позволяет достичь лучших показателей и быстродействия.

В качестве примера гибридных алгоритмов будут рассмотрены 4 классических комбинаций алгоритмов.

1. Генетический алгоритм с локальным поиском

Генетические алгоритмы – это мощные методы оптимизации, но одна из их проблем это застревания, для ее преодоления и повышения скорости сходимости их можно объединить с локальным поиском.

Локальный поиск – позволит уточнять решения с помощью итеративных улучшений.

Формула локального поиска может быть представлена как:

$$x_{new} = x_{current} + \alpha \cdot \Delta x$$

Где x_{new} - новое решение, $x_{current}$ - текущее решение, α – коэффициент, Δx - изменение, основанное на анализе окрестности текущего решения.

2. Эволюционные стратегии с мета моделью

Эволюционные стратегии – они основаны на принципах естественного отбора и эволюции. В этом случае намного продуктивнее будет скомбинировать их с мета моделью

Мета модель – по своей сути является аппроксимацией исходной функции, которая может предсказывать поведение функции в различных точках пространства, тем самым повышая скорость работы всего процесса.

Формула для предсказания мета моделью может быть представлена как:

$$f_{meta}(x) = \sum_{i=1}^{n} \beta_i \cdot \varphi_i(x)$$

Где $f_{meta}(x)$ - предсказанное значение мета моделью, β_i - коэффициенты мета модели, $\varphi_i(x)$ - базисные функции.

3. Генетический алгоритм с алгоритмом имитации отжига

Генетические алгоритмы – имеют отличную глобальную оптимизацию, но это не касается пространств с большим количеством оптимумов. В этих случаях алгоритм работает достаточно долго, для улучшения идеальным вариантом будет алгоритм имитации отжига.

Имитация отжига – имитирует процесс охлаждения метала после нагревания и изменения его структуры, для поиска глобальных оптимальных решений.

Формула для алгоритма имитации отжига:

$$P(e,e',T)=\begin{cases}1\_ если\_ e'<e\\ \frac{e'}{e^{\frac{T}{k}}} иначе\end{cases}$$

Где $P(e,e',T)$ – вероятность перехода к новому состоянию, e – энергия текущего состояния, e' – энергия нового состояния, T – текущая температура, k – параметр отжига.

4. Метод оптимизации с градиентным спуском и генетическим алгоритмом

Градиентный спуск – очень часто используется в функциях, где используется градиент, для нахождения локальных оптимумов. Одни из его проблем это застревание в локальных функциях и возможные проблемы со сходимостью. В качестве улучшения его можно соединить с генетическим алгоритмом, благодаря этому мы получим увеличение производительности и улучшение глобальной оптимизации.

Генетический алгоритм – использоваться для разнообразных решений и поиска глобальных оптимумов, тогда как градиентный спуск будет использоваться для уточнения найденных решений и улучшения локальной сходимости.

Формула для шага градиентного спуска:

$$x_{new}=x_{old}-\eta\cdot\nabla f(x_{old})$$

Где x_{new} – новое решение, x_{old} – текущее решение, η – скорость обучения, $\nabla f(x_{old})$ – градиент функции в точке x_{old}.

Гибридные алгоритмы являются мощными инструментами оптимизации и используются в различные сферах деятельности, в качестве примера можно выделить:

1. Промышленная оптимизация

В промышленности гибридные алгоритмы широко применяются для решения таких задач как:

- Оптимизации производственных процессов,
- Планирования производства,
- Оптимизации энергопотребления,
- Оптимизации параметров производственной линии,
- И других.

Здесь пересилена малая часть применения гибридных алгоритмов.

2. Финансовая аналитика

В финансовой аналитике гибридные алгоритмы могут быть применены для:

- Построения оптимальных инвестиционных портфелей,
- Прогнозирования цен на финансовых рынках,
- Оптимизации стратегий торговли и управления рисками,
- И других.

Методы позволяют учитывать множество факторов и адаптироваться к изменяющимся условиям рынка.

3. Медицинская диагностика и лечение

В медицине гибридные алгоритмы могут использоваться для:

- Оптимизации диагностики заболеваний,
- Планирования лечения и разработки индивидуализированных методов терапии.
- И других

Они позволяют анализировать огромные объемы данных пациентов и предсказывать оптимальные подходы к их лечению с учетом индивидуальных особенностей.

Таким образом, гибридные алгоритмы являются мощным инструментом, с помощью которого решаются сложнейшие задачи в совершенно разных областях. Способность соединять различные алгоритмы оптимизации делает гибридные алгоритмы незаменимыми для оптимизации, именно благодаря им возможны достижения высоких показателей эффективности и точности в поиске оптимальных решений. В будущем список сфер, в которых будут применяться эти алгоритмы, расширится благодаря их адаптивности.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Краснов А. В. Гибридные алгоритмы в задачах оптимизации. – Москва: Физматлит, 2015.

2. Краснов Д. В. Гибридные методы в оптимизации. – Москва: Издательство Московского Университета, 2016.

3. Нестеров Ю. Е. Методы выпуклой оптимизации. – Москва: МЦНМО, 2018.

4. Трубачев В. И. Основы гибридных методов оптимизации. – Санкт-Петербург: БХВ-Петербург, 2019.

5. Чернов И.В. Эволюционные алгоритмы: от теории к практике. – Санкт-Петербург: Наука, 2017.

УДК 004.72

ТКАЧЕВ Д.Д.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ПРИМЕНЕНИЕ МИКРОКОНТРОЛЛЕРОВ ESP32 ДЛЯ ПОСТРОЕНИЯ СЕТИ УСТРОЙСТВ ИНТЕРНЕТА ВЕЩЕЙ

Рассматривается разработка mesh-сети устройств Интернета вещей на основе микроконтроллеров ESP32 и библиотеки PainlessMesh.

Mesh-сети становятся все более распространенными в области Интернета вещей. Mesh-сеть представляет собой сеть, в которой каждое устройство может работать как узел передачи данных, тем самым образуя множество маршрутов для передачи данных.

Использование mesh-сетей позволяет создавать распределенные системы, где устройства могут обмениваться данными напрямую друг с другом, обходя необходимость подключения к центральному серверу.

Преимуществами разработки mesh-сети являются:

1. Надежность: mesh-сети позволяют устройствам общаться напрямую друг с другом, т.е. если одно устройство выходит из строя, другие устройства в сети могут продолжать работу.

2. Масштабируемость: mesh-сети легко масштабируются, поскольку новые устройства могут быть легко добавлены к существующей сети без необходимости изменения инфраструктуры.

3. Энергоэффективность: устройства в mesh-сети могут использовать меньше энергии, поскольку данные могут передаваться между устройствами через промежуточные узлы, а не напрямую.

4. Покрытие: mesh-сети обеспечивают более широкое покрытие, поскольку каждое устройство может служить как узел передачи данных для других устройств в сети.

5. Безопасность: mesh-сети обладают высоким уровнем безопасности, так как данные могут быть зашифрованы и передаваться через защищенные каналы.

Для разработки mesh-сети используется микроконтроллер ESP32, имеющий широкий набор интерфейсов для подключения датчиков, низкую стоимость, а также встроенные модули Wi-Fi и Bluetooth, что делает его идеальным выбором для разработки mesh-сети.

Для сбора параметров окружающей среды к микроконтроллеру ESP32 могут быть подключены датчик температуры и влажности воздуха DHT11, фоторезистор GL5537 и датчик движения HC-SR501.

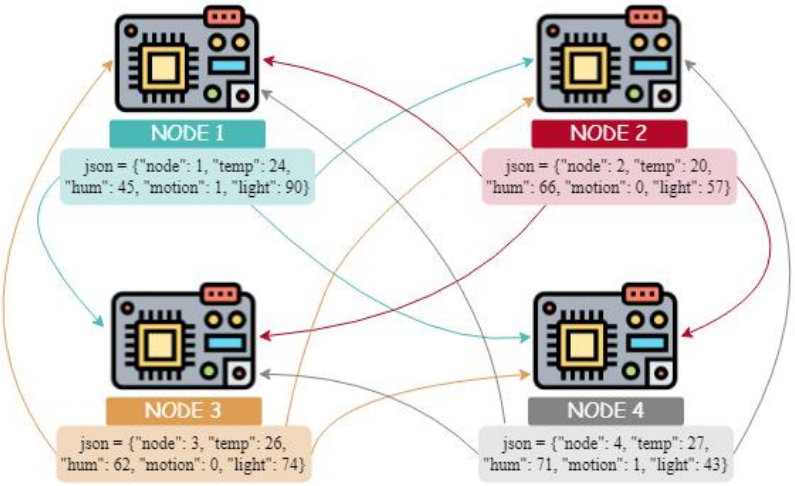


Рисунок 1 – Mesh-сеть устройств на основе микроконтроллеров ESP32 и подключенных датчиков для сбора и передачи параметров среды

Для создания mesh-сети на микроконтроллерах ESP32 используется библиотека PainlessMesh, которая предоставляет простой интерфейс для создания сети и обмена данными между устройствами. Библиотека использует протокол OLSR, позволяющий устройствам автоматически настраивать маршруты передачи данных и обходить недоступные узлы. Также в сети есть главный узел, который передает всю информацию на IoT-шлюз. Если главный узел становится недоступен, его функции случайным образом переходят на другой узел.

Для инициализации mesh-сети с использованием библиотеки PainlessMesh необходимо создать экземпляр класса PainlessMesh и вызвать у него метод init(), в который в качестве параметров следует передать SSID, пароль и номера порта создаваемой mesh-сети.

```
#include "painlessMesh.h"
painlessMesh mesh;

void setup()
{
  mesh.setDebugMsgTypes(ERROR | STARTUP);
  mesh.init(MESH_PREFIX, MESH_PASSWORD, MESH_PORT);
```

```
    mesh.onReceive(&receivedCallback);
    mesh.onNewConnection(&newConnectionCallback);

mesh.onChangedConnections(&changedConnectionCallback);
    mesh.setHostname("node");
  }
```

Функция `receivedCallback()` отвечает за обработку полученных данных от других узлов mesh-сети.

```
  void receivedCallback(uint32_t from, String &msg) {
    if (mesh.isRoot())
    {
      Serial.println("This is the root node");
      sendMessageToServer(msg);
    } else
    {
      Serial.println("This is not the root node");
    }
    Serial.printf("Received msg=%s\n", msg.c_str());
  }
```

Кроме того, функция `receivedCallback()` вызывает функцию `sendMessageToServer()`, которая передает полученные от других узлов данные на IoT-шлюз, если текущий узел является главным.

```
  void sendMessageToServer(String msg)
  {
    WiFi.mode(WIFI_AP_STA);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED)
    {
      Serial.print(".");
      delay(1000);
    }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println(WiFi.localIP());
    while (!client.connect(host, port))
    {
      Serial.println("Connection to host failed");
      delay(1000);
    }
    Serial.println("Host connected");
    client.print(msg);
    client.stop();
  }
```

В приведенном листинге сначала устанавливается соединение между ESP32 и Wi-Fi точкой доступа. Для передачи данных на IoT-шлюз создается экземпляр класса WiFiClient, который используется

для установления соединения с IoT-шлюзом с помощью метода connect(), а затем данные отправляются на IoT-шлюз с помощью метода print().

Заключение

В статье разработана mesh-сеть устройств Интернета вещей, которая позволяет устройствам обмениваться данными между собой без необходимости подключения к общей точке доступа. Разработка подобных mesh-сетей является важным этапом создания современной инфраструктуры для подключения различных устройств к Интернету, что позволяет повысить надежность, эффективность и безопасность работы IoT систем, которые используются во многих областях, таких как умный дом, медицина, производство и транспорт.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Перепелкин Д.А., Ткачев Д.Д. Разработка облачной платформы и визуальной программной системы конфигурирования устройств Интернета вещей // Вестник Рязанского государственного радиотехнического университета. – 2022. – № 82. – С. 73-88.

2. Перепелкин Д.А., Ткачев Д.Д. Разработка устройства сбора параметров для системы интернета вещей // Информационные технологии в прикладных исследованиях, межвузовский сборник научных трудов. – Рязань: ИП Коняхин А.В., 2023. – 246 с. С. 176-180.

3. Перепелкин Д.А., Ткачев Д.Д. Четырехуровневая архитектура программно-конфигурируемой сети устройств интернета вещей // Информационные технологии в прикладных исследованиях, межвузовский сборник научных трудов. – Рязань: ИП Коняхин А.В., 2023. – 246 с. С. 180-183.

УДК 004.72

ТКАЧЕВ Д.Д.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

РАЗРАБОТКА ХАБА УСТРОЙСТВ ИНТЕРНЕТА ВЕЩЕЙ

Рассматривается разработка хаба устройств Интернета вещей для сбора данных от конечных IoT-устройств и управлениями ими.

Хаб устройств Интернета вещей (IoT-хаб) позволяет управлять и координировать работу других устройств Интернета вещей. Он является центральным элементом в локальной сети Интернета вещей и

обеспечивает связь между всеми устройствами, управление ими, а также обработку данных и их отправку в облачную платформу.

В качестве устройств Интернета вещей, подключаемых к IoT-хабу, используются умные розетки TP-Link Tapo P100, умные лампочки TP-Link Tapo L530 и микроконтроллеры ESP32, собирающие с помощью датчиков информацию о температуре, влажности, уровне освещенности окружающей среды и наличии движения.

IoT-хаб осуществляет сканирование Wi-Fi сети, к которой он подключен, и определяет все беспроводные устройства, доступные в данной сети (рисунок 1).

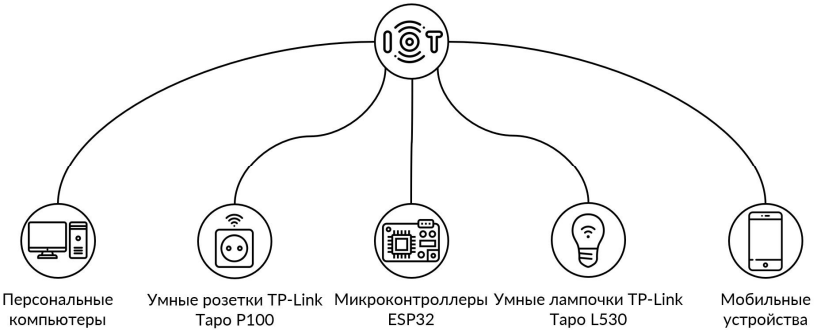


Рисунок 1 – Сканируемые IoT-хабом устройства

Идентификация каждого устройства в сети происходит с помощью его IP и MAC-адресов. Сканирование IoT-хабом Wi-Fi сети выполняется с помощью утилит, реализованных на языке программирования Python на основе протоколов ICMP и ARP (рисунок 2). Утилита ping осуществляет отправку пакета эхо-запроса протокола межсетевых управляющих сообщений ICMP на указанный клиент и ожидает эхо-ответа ICMP. Утилита arp позволяет определить MAC-адрес клиента по известному IP-адресу.

Рисунок 2 – Стек технологий хаба устройств Интернета вещей

Основным классом программы является класс IoTHub, который отвечает за работу хаба устройств Интернета вещей.

Методы get_router_ip() и get_server_ip() используются для получения локальных IP-адресов маршрутизатора и IoT-хаба. Метод get_global_ip() возвращает глобальный IP-адрес всего узла. На основе локального IP-адреса маршрутизатора определяется идентификатор подсети и формируется пул доступных IP-адресов устройств в сети в методе init_ip_addresses().

Для выполнения утилиты ping используется метод ping_device(), который в качестве входного параметра принимает IP-адрес и с помощью метода ping() библиотеки pythonping определяет доступность устройства по этому IP-адресу. Данный метод позволяет также определить задержку передачи данных до устройства и его тип. Метод connect_tapo() проверяет, является ли сканируемое устройство умной розеткой или лампочкой TP-Link Tapo, и в случае успешного соединения, возвращает текущее состояние данного устройства. Для ускорения сканирования Wi-Fi сети IoT-хабом вызов функции ping_device() для каждого IP-адреса осуществляется в отдельном потоке. Метод get_mac_address() библиотеки getmac принимает IP-адрес устройства и возвращает его MAC-адрес.

Для управления умными устройствами TP-Link Tapo IoT-хаб содержит методы turn_on() и turn_off(), которые отвечает за включение и выключение устройства соответственно. IP-адрес управляемого устройства и тип команды содержатся в JSON объекте, принимаемом IoT-хабом от облачной платформы. Если целевое устройство в конкретный момент времени недоступно, то IoT-хаб продолжит осуществлять попытки подключения к нему в методе connect_tapo() до момента, пока не будет установлено соединение и выполнена команда.

Для получения данных о температуре, влажности, уровне освещенности окружающей среды и наличии движения от устройства сбора параметров в IoT-хабе используется сокет-сервер, который использует метод launch_server() для установки соединения и метод client_esp32() для прослушивания данных.

Взаимодействие хаба устройств Интернета вещей и облачной платформы происходит с помощью сокета – двунаправленного соединение, по которому осуществляется передача данных (рисунок 3).

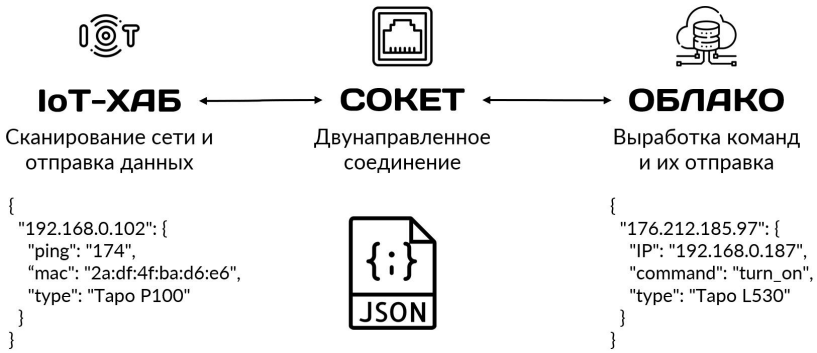


Рисунок 3 – Взаимодействие IoT-хаба и облачной платформы

IoT-хаб собирает данные от подключенных устройств и передает их на облачную платформу, где происходит их обработка и анализ. Облачная платформа может отправлять команды на управление устройствами Интернета вещей через IoT-хаб. Метод send_messages() отвечает за отправку данных о IP-адресах и MAC-адресах, задержках и типах устройств данной сети в облачную платформу через сокет в формате JSON. Метод receive_messages() отвечает за прослушивание входящих сообщений от облачной платформы. В случае, если облачная платформа недоступна в настоящий момент, IoT-хаб циклически будет осуществлять попытки подключиться к ней в методе do_connection_attempt(), пока не будет установлено соединение.

Заключение

В статье разработан IoT-хаб, который обеспечивает взаимодействие устройств в ПКС Интернета вещей, а также выполняет сбор данных и их отправку в облачную платформу для дальнейшей обработки и анализа, перенаправление команд от облачной платформы на конечные устройства Интернета вещей.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Перепелкин Д.А., Ткачев Д.Д. Разработка шлюза и облачной платформы программно-конфигурируемой mesh-сети устройств Интернета вещей // Вестник Рязанского государственного радиотехнического университета. – 2023. – № 84.

2. Перепелкин Д.А., Ткачев Д.Д. Разработка сканера устройств беспроводной сети // Новые информационные технологии в научных исследованиях: Материалы XXVII Всероссийской научно-технической конференции студентов, молодых ученых и специалистов. В 2-х томах, Рязань, 07–09 декабря 2022 года. Том 2. – Рязань: Рязанский

государственный радиотехнический университет им. В.Ф. Уткина, 2022. – С. 39-40.

УДК 004.65

ТОБРАТОВ Ю.М., КОШЕЛЕВ А.Д.
Рязанский государственный радиотехнический университет имени В.Ф. Уткина

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ УЧЕТА ПРЕДЛАГАЕМЫХ УСЛУГ

В данной статье, на примере спортивного комплекса, рассматривается возможность создания информационной системы (ИС) учета предлагаемых услуг, описывается её структура и пошаговый алгоритм разработки.

Введение

В настоящее время, в связи с повсеместной информатизацией, многие предприятия столкнулись с проблемой обработки и систематизации больших объемов данных. Действительно, вся совокупность характеристик работников или клиентов, услуг, систем скидок и поощрений хранится в различных текстовых, табличных документах, что является неудобным, так как отсутствует возможность для быстрого выполнения элементарных и многократно повторяющихся функций, например, сортировки по какому-либо параметру, поиска конкретной информации и т.д. Вследствие чего существенно снижается производительность и увеличивается вероятность ошибок сотрудников компании. Поэтому задача разработки единой информационной системы учета предоставляемых услуг, позволяющей оптимизировать работу предприятия, – является актуальной.

В данной статье, на примере спортивного комплекса, была предложена модель подобной ИС.

Структура ИС

Наиболее распространенный способ организации данных в автоматизированных информационных системах – организация в виде базы данных (БД). Разрабатываемая ИС учета предлагаемых услуг подразумевает ввод, хранение и обработку множество разнообразной информации, следовательно, наличие пользовательского интерфейса и поддержку функций математических вычислений. При таких условиях выберем в качестве инструмента разработки – СУБД, сочетающую в себе удобные средства создания и ведения БД.

Предлагается использовать двухзвенную модель архитектуры клиент-сервер «Удаленное представление» (рисунок 1). Функции клиента ограничиваются функциями представления информации, в то время как прикладные функции обеспечиваются приложением, хранящимся на сервере.

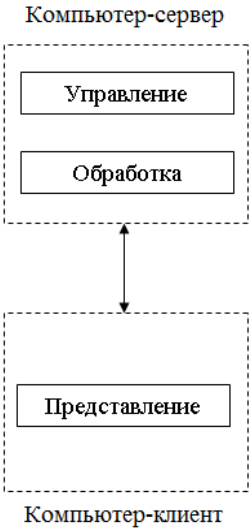


Рисунок 1 – Модель «Удаленное представление»

Интерфейс позволяет пользователю легко и быстро ознакомиться с представленными услугами. Пользователь наделен следующими возможностями:

- Оформление путевки;
- Просмотр информации о клиентах;
- Просмотр информации о доступных услугах;
- Просмотр информации о доступном инвентаре, его заказ;
- Просмотр информации о доступных номерах;
- Просмотр информации о доступных скидках.

Кроме того, в системе должна присутствовать обратная связь с пользователем. Оповещения о заказе путевки должны приходить на электронный почтовый адрес клиента.

Во время своего пребывания в комплексе, клиент может свободно пользоваться разработанной базой данных для получения необходимой ему информации следующим образом:

- обратиться к администратору, который предоставит желаемую информацию при помощи своего компьютера;

- самостоятельно получить необходимую информацию при помощи специальных терминалов, расположенных на всей территории комплекса;

- скачать при помощи общедоступной сети wifi специальное приложение на свой смартфон (поддерживаемые ОС – IOS, Android, Windows Phone).

Система взаимодействует с базой данных, в которой представлена информация о спортивном комплексе. Вносить какие-либо изменения в базу данных может только администратор, а также выполнять все возможности клиента.

В базе данных должна храниться следующая информация:

- ФИО клиента.
- Для каждого клиента должны быть указан телефон.
- Название инвентаря, которое должно быть уникальным.
- Для каждого номера должны быть указаны класс номера и цена.
- ФИО сотрудника.
- Должность сотрудника. Каждый сотрудник может занимать только 1 должность.
- Оклад сотрудника, который определяется только должностью.
- Для каждого сотрудника должен быть указан телефон.
- Наименование услуг.
- Скидочные карты клиентов.

Алгоритм разработки ИС

Процесс создания ИС учета предоставляемых услуг спортивного комплекса можно разбить на два больших этапа: разработка серверной и клиентской частей ИС.

В свою очередь, разработка серверной части включает в себя следующие итерации: инфологическое проектирование БД, а именно выявление сущностей и связей с последующим построением ER-диаграмм, и даталогическое проектирование, которое состоит из перехода от ER-диаграмм к предварительным отношениям, заполнением их атрибутами, проверкой на соответствие нормальным формам, и, наконец, построением схемы данных.

Разработка клиентской части подразумевает наличие таких шагов, как организация взаимодействия клиентской программы с БД, создание интерфейса пользователей, форм и сценария инсталляции клиентской программы.

Рассмотрим эти этапы более подробно.

1) Разработка серверной части.

Шаг 1. Выявление сущностей и связей. В предметной области можно выделить следующие сущности:

- Инвентарь. Первичный ключ: Номер инвентаря.
- Клиенты. Первичный ключ: Номер клиента.
- Сотрудники. Первичный ключ: Номер сотрудника.
- Должность. Первичный ключ: Должность.
- Услуги. Первичный ключ: Номер услуги.
- Скидочные карты. Первичный ключ: Номер карты.
- График работы. Первичный ключ (составной): Номер услуги, Номер сотрудника.
- Путевки. Первичный ключ (составной): Номер клиента, Номер услуги, Номер комнаты.
- Статус. Первичный ключ: Статус.
- Номера. Первичный ключ: Номер комнаты.
- Тип карты. Первичный ключ: Тип карты.
- Тип размещения. Первичный ключ: Тип размещения.
- Выдача инвентаря. Первичный ключ: Номер инвентаря, номер сотрудника, номер клиента
- Льготы. Первичный ключ: Номер клиента, Номер карты

Связи между сущностями:

– Услуги входят в график работы;
– У сотрудников есть график работы;
– Сотрудники занимают должности;
– Клиенты заказывают путевки;
– Клиенты имеют статус;
– Номер комнаты указывается в путевке;
– Номер отеля (гостиницы) имеет тип размещения;
– Услуги указываются в путевке;
– Сотрудники выдают инвентарь;
– Инвентарь выдается клиенту;
– Инвентарь выдается с определенным номером;
– Клиенты имеют льготы;
– Льготы предоставляются по скидочной карте;
– Каждая скидочная карта имеет определенный тип.

На основе вышеописанного можно построить ER-диаграммы. Например, связь «Сотрудники занимают должности», для которой характерно следующее, отображена на рисунке 2:

– Сотрудник может занимать только одну должность;
– Одну должность могут занимать много сотрудников;
– Сотрудник обязательно занимает должность;

– Должность может быть не занята.

Рисунок 2 – Связь «Сотрудники занимают должности»

Аналогично строим ER-диаграммы других отношений. Общая ER-диаграмма будет иметь вид, представленный на рисунке 3.

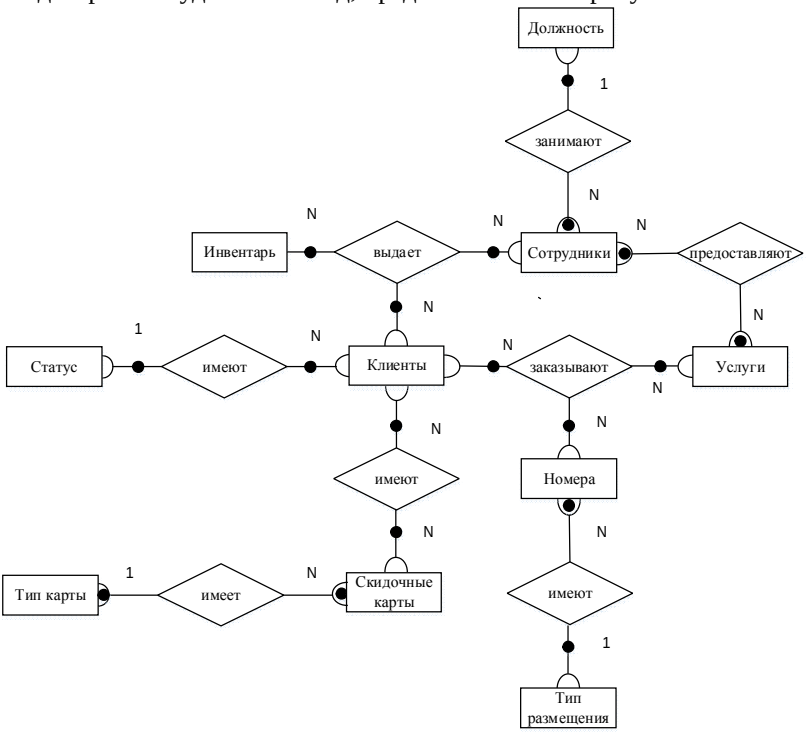


Рисунок 3 – Общая ER-диаграмма

Шаг 2. Переход от ER-диаграмм к предварительным отношениям. Например, для связи «Сотрудники занимают должности»: так как степень бинарной связи 1: N, а класс принадлежности N-связной сущности обязателен, то формируется 2 отношения [1]. По одному для каждой сущности. При этом ключом в каждом отношении является ключ соответствующей сущности, а

также в отношение для N-связной сущности добавляется в качестве не ключевого атрибута ключ односвязной сущности – *Сотрудники (Номер сотрудника, должность); Должности (Должность).*

Аналогичный переход прописывается для всех связей. После чего происходит распределение атрибутов по отношениям.

Шаг 3. Проверка предварительных отношений на соответствие нормальным формам. Для отношений строятся диаграммы функциональных зависимостей, как на рисунке 4.

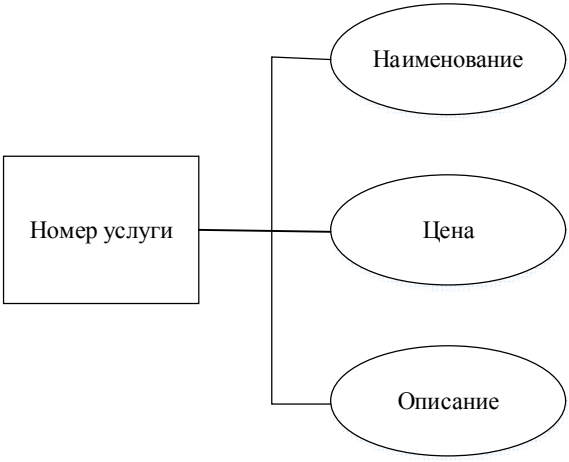


Рисунок 4 – диаграмма функциональных зависимостей для отношений «Услуги»

Например, для отношений «Услуги» проверка на соответствие имеет вид: отношение находится в 2НФ, так как находится в 1НФ, и каждый не ключевой атрибут функционально полно зависит от первичного ключа. Отношение находится в 3НФ, так как находится в 2 НФ, и в нем нет транзитивных зависимостей не ключевых атрибутов от первичного ключа. Отношение находится в БКНФ, так как оно находится в 3НФ, и в детерминанты всех функциональных зависимостей являются потенциальными ключами [2].

Аналогично для других отношений.

Шаг 4. Построение схемы данных. После проделанных итераций получаем общую схему БД, которая будет иметь вид, отображенный на рисунке 5.

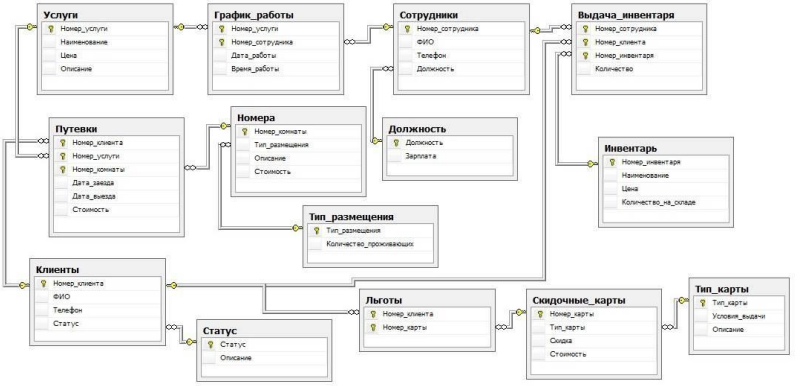


Рисунок 5 – Схема базы данных

2) Разработка клиентской части. Для доступа к базе данных предлагается использовать технологию ADO.NET., а именно следующие экземпляры компонентов доступа к БД:

SqlCommand – представляет инструкцию Transact-SQL или хранимую процедуру, выполняемую над базой данных SQL Server, используется для выполнения запросов к БД или вызова хранимых процедур;

SqlDataAdapter – представляет набор выполняемых над данными команд и подключения базы данных, которые используется для заполнения и обновления базы данных SQL Server, используется для извлечения и сохранения данных в БД;

DataSet – расположенный в памяти кэш данных, загружаемых из источника данных.

DataGridView – настраиваемая таблица для отображения данных.

Разработка форм подразумевает следующее: программа «Спортивный комплекс» содержит 7 форм – главное меню, информация о клиентах, скидки, оформить путевку, предоставляемые услуги (услуги), предоставляемые услуги (инвентарь), проживание.

Например, при запуске программы на экран выводится главная форма «Главное меню» (рис. 6), которая позволяет просматривать и редактировать информацию из всех таблиц.

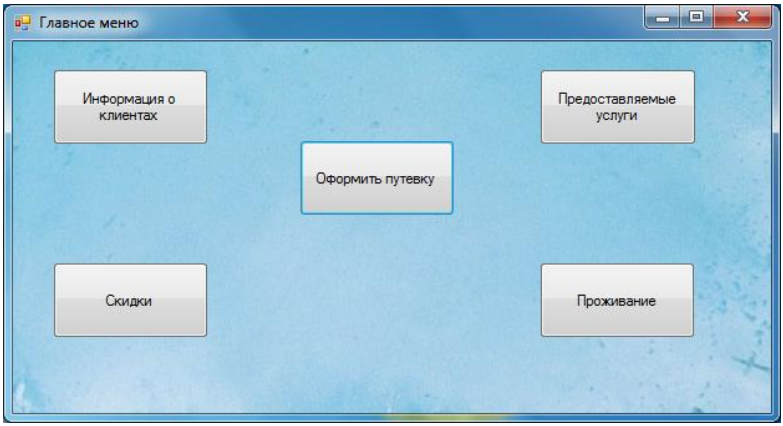


Рисунок 6 – Форма «Главное меню»

Список компонентов данной формы и их назначение представлены следующим образом:

button1 – кнопка оформления путевки;

button2 – кнопка для просмотра предоставляемых услуг;

button3 – кнопка для просмотра информации о проживании;

button4 – кнопка для просмотра информации о существующих скидках;

button5 – кнопка для просмотра информации о клиентах.

Затем производится разработка сценария инсталляции клиентской программы, руководств программиста и пользователя.

Заключение

В статье была рассмотрена возможная модель автоматизированной информационной системы учета предлагаемых услуг спортивного комплекса. Были описаны структура и пошаговый алгоритм разработки ИС.

В качестве преимуществ данной системы можно выделить следующее: наглядность пользовательского интерфейса, возможность доработки и адаптации системы в соответствии с возрастающими требованиями заказчика, что обеспечивает гибкость и поддержание актуальности данных на протяжении всего срока эксплуатации ИС.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Васюткина, И.А. Разработка приложений на C# с использованием СУБД PostgreSQL: учебное пособие / И.А. Васюткина, Г.В. Трошина, М.И. Бычков, С.А. Менжулин. – Новосибирск: НГТУ, 2015. – 143 с.

2. Волк, В. К. Базы данных. Проектирование, программирование, управление и администрирование / В. К. Волк. – 4-е изд., стер. – Санкт-Петербург: Лань, 2023. – 244 с.

УДК 658.56

ТОБРАТОВ Ю.М., КОШЕЛЕВА М.С.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

СИСТЕМА МОНИТОРИНГА МОБИЛЬНОЙ МЕДИЦИНСКОЙ ЛАБОРАТОРИИ

В данной статье была предложена модель системы удаленного мониторинга мобильной медицинской лаборатории, описаны структурная и функциональная схемы устройства, а также его программная реализация.

В связи со стремительным развитием и повсеместным распространением информационных технологий, ведущие предприятия автомобильной, медицинской, нефтегазовой и других отраслей промышленности всё чаще задумываются о внедрении систем удаленного контроля за разрабатываемыми устройствами. Действительно, данное решение дает ряд преимуществ, в том числе – своевременная передача информации об аварии на контролируемом объекте, предварительные сведения о конкретной неисправности, а также осуществление базовых шагов по устранению поломки на программном уровне.

В настоящее время медицинские учреждения Российской Федерации оснащены огромным количеством подвижных лабораторных комплексов, не подразумевающих наличие:

– технических специалистов, способных дать первичное экспертное заключение о возможных причинах выхода оборудования из строя,

– автоматизированных систем контроля параметров эксплуатации оборудования.

Следовательно, функцию наблюдения за аппаратной составляющей лаборатории берет на себя обслуживающий данный комплекс медицинский персонал. Поэтому задача разработки системы удаленного мониторинга мобильного медицинского комплекса, способной повысить уровень безопасности, надежности комфорта

сотрудников и сократить эксплуатационные расходы, является актуальной.

Система удаленного мониторинга мобильной медицинской лаборатории

В общем виде система удаленного мониторинга подвижной медицинской лаборатории включает в себя:

– датчики – аварийные извещатели, реагирующие на тревожное событие (пожар, попытка проникновения в кабинет), характеристики датчиков определяют основные параметры всей системы;

– приемно-контрольные приборы (ПКП) – устройства, которые получают сигнал тревоги от датчиков и осуществляют управление по заданному алгоритму исполнительными устройствами (в простейшем случае контроль за работой охранно-пожарной сигнализации состоит из включения и выключения датчиков, фиксации сигналов тревоги);

– исполнительные устройства обеспечивают выполнение заданного алгоритма действий системы в ответ на то или иное тревожное событие (подача сигнала оповещения, включение механизмов пожаротушения, автодозвон по заданным номерам телефонов и т. п.);

– канал связи для оповещения пользователя или передачи сообщения в отдел наблюдения. Передача данных может осуществляется посредством: GSM-канала; подключения к локальной сети (стек протоколов Ethernet); подключения к сети Internet. Выбор варианта передачи данных зависит от наличия технической возможности подключения на контролируемом объекте.

Общая структурная схема системы наблюдения представлена на рисунке 1.

В состав терминальных устройств, устанавливаемых на подвижных объектах, входит приемник GPS, с помощью которого определяется его местоположение. Информация о местоположении объекта, в виде коротких сообщений, поступает в центр с помощью GSM модема. Терминальное устройство обрабатывает сигналы от охранной системы объекта по 8-ми или 16 входам и в случае возникновения угрозы передает запрограммированное заранее короткое сообщение в центр, заданный телефонный номер. По двум входам терминальное устройство может обрабатывать аналоговую величину (напряжение), поступающую от какого-либо датчика, и сообщать информацию о достижении заранее установленной пороговой величины или о выходе этой величины за пределы заданного интервала.

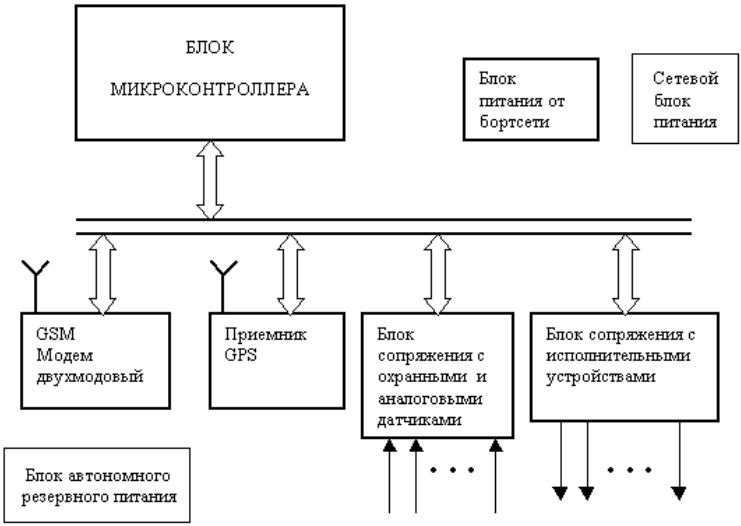


Рисунок 1 – Общая структурная схема системы мониторинга

Терминальные устройства для стационарных объектов отличаются от терминальных устройств для подвижных объектов, в основном, только отсутствием в их составе GPS приемников и выполняют все остальные перечисленные выше функции.

Центр принимает информацию от множества датчиков, расположенных на подвижных объектах, и накапливает в соответствующих базах данных из состава своего математико-программного обеспечения. Данные о местоположение подвижных объектов отображаются на электронной карте в виде точки или траектории маршрута движения. Остальные данные архивируются и подвергаются статистической обработке. Маршруты движения подвижных объектов также накапливаются в соответствующих базах данных центра.

Центр может передавать команды на терминальное устройство. В соответствии с принимаемыми командами терминальное устройство управляет исполнительными устройствами систем автомобиля или инженерными системами объекта. Например, включает автономный обогрев в подвижном кабинете заранее, т.е. до начала работы кабинета на месте прибытия.

Предлагаемая авторами система удаленного мониторинга отличается от аналогов модификацией функциональных возможностей, позволяющих повысить надежность и эффективность

наблюдения за работой технических средств подвижных медицинских лабораторий за счет сокращения времени передачи информации о состоянии оборудования до обслуживающего персонала.

Рассматриваемое устройство содержит:

– считыватель параметров контролируемого оборудования,

– контроллер,

– дополнительный модем, соединенный с антенно-фидерным блоком, предназначенным для передачи данных по каналу связи на сервер центра обработки и анализа информации,

– приемник сигналов спутниковых навигационных систем,

– модуль определения координат местонахождения, предназначенный для получения информации о положении комплекса, его скорости,

– модуль контроля канала связи,

– блок регистрации измеренных параметров.

Введение персонального компьютера для оператора автоматизированного рабочего места, обслуживающего контролируемое оборудование, позволяет системе для удаленного мониторинга и анализа работы технических средств подвижных медицинских лабораторий выполнять функцию наблюдения при временном отсутствии GSM связи, а наличие блока регистрации измеренных параметров и модуля контроля канала связи обеспечивает полное восстановление с последующей передачей на сервер центра данных, полученных за период отсутствия связи, следовательно, повышает надежность работы устройства.

Кроме того, предлагаемое техническое решение предполагает использование мобильного терминала оператора, сокращая время доведения важной информации о состоянии технических средств бортовых систем и восстановления функционирования этого оборудования, или предотвращая его отказ.

Аппаратное обеспечение системы

Рассмотрим структурную и функциональную схемы системы удаленного мониторинга мобильной медицинской лаборатории на примере рентгеновского подвижного комплекса КРП-УР.

Структурная схема:

– Система включает датчик открытия двери, пожарный датчик, приемник GPS, датчик температуры и исполнительные устройства: блоки управления флюорографом, маммографом, модулями отопления и кондиционирования.

– Датчик открытия двери выполняет две функции: контроль несанкционированного проникновения в подвижной рентгеновский кабинет и запрет проведения снимка при открытой входной двери.

– Пожарный датчик при возникновении задымления и резком возрастании градиента температур выдает сигнал на блок контроля и передачи данных, что приводит к срабатыванию системы и передачи сообщения отделу технического контроля или в центр мониторинга медучреждения.

– Температурный датчик измеряет уровень температуры в подвижной медицинской лаборатории.

– Приемник GPS осуществляет прием данных о местоположении комплекса и предает эти данные в блок контроля и передачи данных.

– С флюорографа и маммографа поступает сигнал о неисправности аппарата. Блок управления принимает сигнал от блока контроля и передачи данных о запрете включения, проведения снимка в случае аварийной ситуации или не соответствии температурного режима эксплуатационным требованиям.

Общая структурная схема системы удаленного мониторинга представлена на рисунке 2.

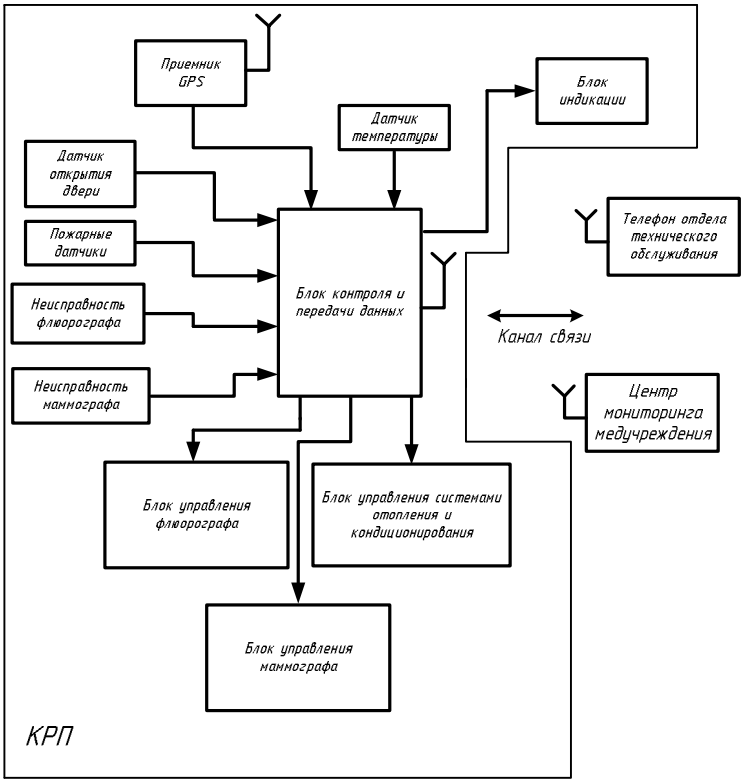


Рисунок 2 – Структурная схема системы удаленного мониторинга

Функциональная схема:

– Используются 2 датчика, выдающие сигнал 1 или 0, подключенные к блоку микроконтроллера (МК). С флюорографа и маммографа подается аналогичный сигнал на вход МК в случае аварии.

– МК осуществляет анализ данных и при срабатывании одного из датчиков выдает сигнал на блок согласования с модемом сотовой связи.

– Блок согласования выполнят преобразовании сигналов соответствующие протоколу RS232 с которым работает модем сотовой связи.

– Блок управления датчиками представляет собой клавиатуру, позволяющую включать и выключать датчики, а также отключать систему в случае срабатывания одного из датчиков.

– Блок индикации выдает сообщение о том, какой датчик сработал и отображает температуру.

– Через гальванические развязки на блоки управления (БУ) флюорографа и маммографа подаются команды на запрет включения, в случае неисправности или при несоответствии условиям эксплуатации.

– В качестве датчика температуры используется аналоговый термопреобразователь сопротивления, преобразование сигнала в цифровой вид осуществляется в МК АЦП.

– В качестве GPS-приемника используется GPS-модуль CH-4706

– Питание системы осуществляется от блока питания постоянного напряжения 12 В для МК, для блоков индикации и управления необходимо питание 5, поэтому применяется преобразователь напряжения (ПН) 12 в 5 В.

Общая функциональная схема разрабатываемой системы удаленного мониторинга представлена на рисунке 3.

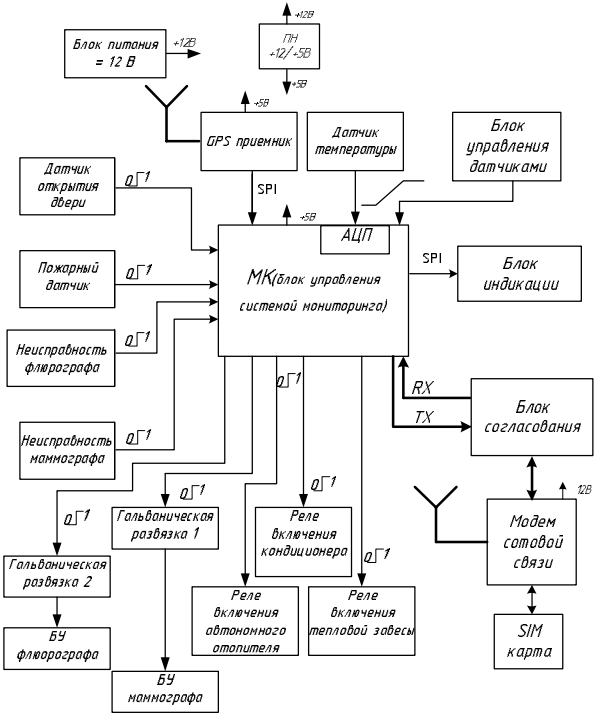


Рисунок 3 – Функциональная схема удаленной системы мониторинга

Программное обеспечение системы

После включения системы происходит проверка работоспособности, выставление всех параметров на начальное положение: сброс таймера, установка тактового генератора, разрешение прерывания от таймера и открытия прерывания от таймера.

Инициализация портов ввода/вывода заключается в назначении портам их функции входа или выхода. Каждой линии порта поставлен в соответствие бит направления передачи данных, который хранится в управляющем регистре TRISA, расположенном по адресу 85h. Если бит управляющего TRISA регистра имеет значение 1, то соответствующая линия будет устанавливаться на ввод. Ноль переключает линию на вывод и одновременно выводит на нее содержимое соответствующего регистра-фиксатора порта. При включении питания все линии порта по умолчанию настроены на ввод [1].

Инициализация SPI осуществляется для связи МК с другими микросхемами. С помощью SPI-интерфейса можно легко связать микроконтроллер с внешними микросхемами. В нашем случае с драйвером семисегментного индикатора.

Инициализация драйвера индикаторов приводит в состояние готовности к приему данных драйвер семисегментного индикатора. Устанавливаем значения сегментов индикаторов равным нулю, т.е. обнуляем значения семисегментного индикатора.

Инициализация USART. Интерфейс USART – последовательный универсальный синхронно-асинхронный приемо-передатчик. В отличии от I2C или SPI, в которых передача бита осуществляется по сигналу синхронизации, передача данных в USART осуществляется через равные промежутки времени в одной линии в каждом направлении. В нашем случае интерфейс USART используется для передачи данных на модем сотовой связи [2].

После инициализации МК и всех устройств, подключенных к нему, начинается последовательный опрос датчиков, входящих в состав системы. В случае если хотя бы один из датчиков сработал, сигнал тревоги поступает на модем. Сигнал будет предаваться до тех пор, пока не будет нажата кнопка стоп на блоке управления.

В основной программе имеются следующие подпрограммы:

1) Индикатора, включающая последовательность действий: перевод линии Load в низкий логический уровень; отправка адреса; задержка; отправка данных; задержка; перевод Load в высокий логический уровень; задержка.

2) Термометра, осуществляющая обработку данных с температурного датчика. Сначала выполняются последовательно следующие функции: инициализация АЦП; получение результатов с АЦП; отправка данных через UART; вывод значения с датчика в регистр дисплея, значение с регистра дисплея передаются по интерфейсу SPI.

Далее выполняется сравнение полученного значения температуры с заданным нижним пределом температуры. В случае если температура ниже нижнего предела, то включается тепловая завеса. Тепловая завеса будет работать, пока температура не превысит нижний порог. Аналогичным образом происходит сравнение полученного значения температуры с заданным верхним пределом температуры. В случае если температура выше верхнего предела, то включается кондиционер. Кондиционер будет работать, пока температура не опустится ниже верхнего предела.

Общая блок-схема основной программы системы удаленного мониторинга приведена на рисунке 4.

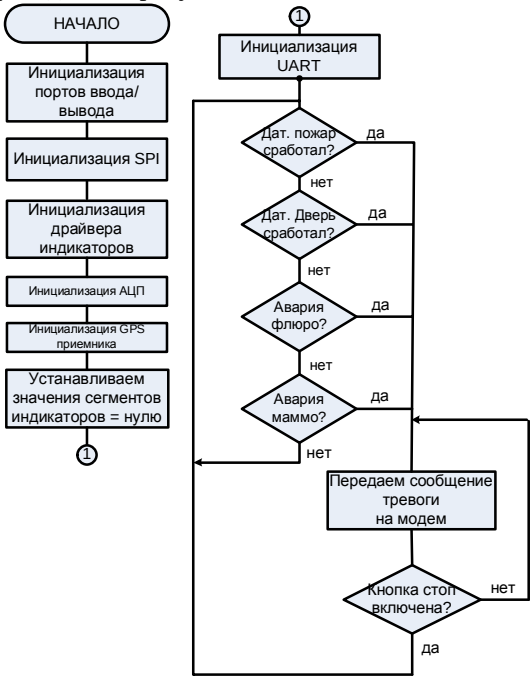


Рисунок 4 – Общая блок-схема основной программы системы удаленного мониторинга

Заключение

Таким образом, авторами статьи была предложена модель системы удаленного мониторинга мобильной медицинской лаборатории. Были рассмотрены её структурная и функциональная схемы на примере рентгеновского подвижного комплекса. Данное устройство призвано повысить уровень надежности, безопасности и комфорта обслуживающего мобильную лабораторию медицинского персонала за счет реализации таких функций, как: управление охранно-пожарной сигнализацией; диспетчеризация; управление основными инженерными системами, такими как отопление, вентиляция и кондиционирование; управление медицинским оборудованием; отслеживание местоположения комплекса.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Смирнов, Ю. А. Основы микроэлектроники и микропроцессорной техники: учебное пособие / Ю. А. Смирнов, С. В. Соколов, Е. В. Титов. – 2-е изд., испр. – Санкт-Петербург: Лань, 2022. – 496 с.

2. Федотов, А. В. Компьютерное управление в производственных системах: учебное пособие для вузов / А. В. Федотов, В. Г. Хомченко. – 2-е изд., стер. – Санкт-Петербург: Лань, 2021. – 620 с.

УДК 004.65

ХОХЛОВ И.В.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

ВЫБОР ОТЕЧЕСТВЕННОГО СЕТЕВОГО ОБОРУДОВАНИЯ ДЛЯ ЛОКАЛЬНОЙ СЕТИ

Статья посвящена обзору сетевого оборудования для локальных сетей отечественных производителей.

Введение

В современном мире, где информационные технологии играют всё более важную роль, выбор правильного сетевого оборудования становится критически важным для обеспечения стабильной и безопасной работы локальной сети. Одним из важных факторов, который следует учитывать при выборе сетевого оборудования, является его происхождение. В данной статье мы рассмотрим

преимущества использования отечественного сетевого оборудования для локальной сети.

Преимущества отечественного сетевого оборудования

1. Совместимость с отечественными стандартами

Отечественное сетевое оборудование разрабатывается и производится с учетом отечественных стандартов и требований. Это означает, что такое оборудование будет лучше всего подходить для работы в отечественных условиях и будет совместимо с другим отечественным оборудованием.

2. Поддержка и сервис

При использовании отечественного сетевого оборудования, вы можете рассчитывать на высокий уровень поддержки и сервиса. Производители отечественного оборудования обычно имеют широкую сеть сервисных центров и технической поддержки, которые могут быстро реагировать на любые проблемы или вопросы, связанные с оборудованием.

3. Цена

Отечественное сетевое оборудование обычно имеет более низкую цену по сравнению с импортным оборудованием. Это связано с тем, что отечественные производители имеют меньшие издержки на транспортировку и логистику, а также могут использовать более дешевую рабочую силу.

4. Качество

Отечественное сетевое оборудование производится с использованием современных технологий и высококачественных материалов. Это обеспечивает высокую надежность и долговечность оборудования.

Виды сетевого оборудования

1. Сетевые коммутаторы

Сетевые коммутаторы являются основным элементом любой локальной сети. Они отвечают за передачу данных между устройствами в сети. Отечественные производители предлагают широкий выбор сетевых коммутаторов различной производительности и функциональности.

2. Маршрутизаторы

Маршрутизаторы используются для соединения нескольких сетей между собой. Они отвечают за маршрутизацию данных между сетями и обеспечивают безопасность данных. Отечественные производители предлагают широкий выбор маршрутизаторов различной производительности и функциональности.

3. Беспроводные точки доступа

Беспроводные точки доступа используются для создания беспроводных сетей. Они позволяют подключать устройства к сети без использования проводов. Отечественные производители предлагают широкий выбор беспроводных точек доступа различной мощности и дальности действия.

4. Устройства мониторинга и управления сетью

Устройства для мониторинга и управления сетью помогают администраторам контролировать и оптимизировать работу локальной сети. При необходимости расширения возможностей мониторинга и управления, такие устройства могут быть заменены на более функциональные модели.

5. Сетевые модули безопасности

Сетевые модули безопасности используются для защиты сети от вирусов и других вредоносных программ. Они позволяют контролировать доступ к сети и защищать данные. Отечественные производители предлагают широкий выбор сетевых модулей безопасности различной производительности и функциональности.

6. Серверы

Серверы играют важную роль в функционировании локальной сети, обеспечивая хранение данных, вычислительные ресурсы и другие сервисы. При увеличении нагрузки на серверы или необходимости обновления программного обеспечения, серверы могут потребовать замены на более мощные или современные модели.

Критерии выбора отечественного сетевого оборудования

1. Производительность

Производительность сетевого оборудования является одним из основных критериев выбора. Она определяется количеством портов, скоростью передачи данных и другими характеристиками.

2. Функциональность

Функциональность сетевого оборудования также является важным критерием выбора. Она определяется набором функций, которые предоставляет оборудование, таких как поддержка протоколов, возможность настройки и управления, и других.

3. Надежность

Надежность сетевого оборудования является критическим фактором, который следует учитывать при выборе. Она определяется качеством материалов, используемых в производстве, а также надёжностью конструкции и соблюдением производителем стандартов качества.

4. Цена

Цена часто является решающим фактором при выборе сетевого оборудования. Она зависит от многих факторов, таких как производительность, функциональность, надежность и другие.

Известные производители отечественного сетевого оборудования

1. ЗАО "Сатурн"

ЗАО "Сатурн" является одним из ведущих производителей отечественного сетевого оборудования. Компания предлагает широкий выбор сетевых коммутаторов, маршрутизаторов, беспроводных точек доступа и других устройств.

2. ООО "НПФ "ЭЛВИС-Телеком"

ООО "НПФ "ЭЛВИС-Телеком" является одним из крупнейших производителей отечественного сетевого оборудования. Компания предлагает широкий выбор сетевых коммутаторов, маршрутизаторов, беспроводных точек доступа и других устройств.

3. ОАО "РТИ"

ОАО "РТИ" является одним из старейших и наиболее опытных производителей отечественного сетевого оборудования. Компания предлагает широкий выбор сетевых коммутаторов, маршрутизаторов, беспроводных точек доступа и других устройств.

4. ООО "Техносерв"

ООО "Техносерв" является одним из ведущих поставщиков отечественного сетевого оборудования. Компания предлагает широкий выбор сетевых коммутаторов, маршрутизаторов, беспроводных точек доступа и других устройств.

Топология малой локальной сети организации

Топология "Звезда" (Star Topology) является одной из самых распространенных топологий в современных сетях. Она часто используется в домашних сетях, офисах, корпоративных сетях и даже в облачных сетях. Процентное соотношение данной топологии по отношению к другим топологиям может зависеть от конкретной области применения сети, но можно сказать, что топология звезда занимает значительную долю. Она часто используется в современных сетях Ethernet, где все устройства подключены к центральному коммутатору или концентратору. Топология "Звезда" (Star Topology) обеспечивает простоту управления сетью, легкость добавления новых устройств и высокую надежность, так как отказ одного устройства не приводит к отказу всей сети.

В корпоративных сетях и офисах эта топология часто используется из-за своей простоты управления, высокой надежности и легкости масштабирования, в домашних сетях также часто

применяется из-за удобства подключения устройств и возможности легко добавлять новые устройства.

В то же время, другие топологии, такие как, "Шина" (Bus Topology), "Кольцо" (Ring Topology), "Дерево" (Tree Topology) и "Сеть" (Mesh Topology) имеют свое место в различных сетевых средах. Например, топология "Кольцо" (Ring Topology) может быть использована в определенных случаях для обеспечения высокой пропускной способности и отказоустойчивости.

Топология "Звезда" (Star Topology) является одной из наиболее распространенных топологий в локальных сетях. Она часто используется в современных сетях Ethernet, где все устройства подключены к центральному коммутатору или концентратору. Такая топология обеспечивает простоту управления сетью, легкость добавления новых устройств и высокую надежность, так как отказ одного устройства не приводит к отказу всей сети.

Перечень оборудования для малых локальных сетей

Локальная сеть организации до 50 машин обычно состоит из следующего сетевого оборудования:

1. Коммутатор (switch) - для подключения всех компьютеров в сеть и обеспечения коммутации данных между ними.
2. Маршрутизатор (router) - для обеспечения маршрутизации данных между локальной сетью и внешними сетями, такими как интернет.
3. Модем - для подключения сети к интернету через провайдера интернет-услуг.
4. Wi-Fi маршрутизатор (Wi-Fi router) - для беспроводного подключения устройств к сети.
5. Сервер - для обеспечения различных сервисов, таких как файловое хранение, электронная почта, веб-сервер и т.д.
6. Брандмауэр (firewall) - для обеспечения безопасности сети от внешних угроз.

Рассматривая локальную сеть организации, где количество компьютеров составляет менее 50 единиц для топологии "Звезда" (Star Topology), в составе сетевого оборудования использовать “межсетевой экран – маршрутизатор” и коммутатор.

Брандмауэр и маршрутизатор – это два разных типа сетевых устройств, которые служат разным целям.

Маршрутизатор – это устройство, которое пересылает пакеты данных между компьютерными сетями, либо внутри одного здания, либо между разными сетями. Маршрутизатор использует протоколы,

такие как ICMP, для связи друг с другом и настройки наилучшего маршрута между любыми двумя хостами.

Брандмауэр, с другой стороны, представляет собой устройство сетевой безопасности, которое отслеживает и фильтрует входящий и исходящий сетевой трафик на основе ранее установленных политик безопасности организации. Брандмауэр может быть реализован как в аппаратном, так и в программном обеспечении или в комбинации того и другого.

"Межсетевой экран - маршрутизатор" (межсетевой брандмауэр - маршрутизатор) – это устройство, которое сочетает в себе функциональность как брандмауэра, так и маршрутизатора в одном устройстве. Устройства этого типа могут обеспечивать преобразование сетевых адресов (NAT), маршрутизацию и брандмауэрную защиту сети. Компонент брандмауэра устройства может быть настроен для блокирования нежелательного трафика, в то время как компонент маршрутизации позволяет пересылать пакеты данных между сетями.

Брандмауэр-маршрутизатор – это более совершенное сетевое устройство, которое обеспечивает как маршрутизацию, так и функции безопасности, что делает его идеальным выбором для защиты сети от несанкционированного доступа, а также обеспечивает связь между различными сетями.

Коммутатор, также известный как сетевой коммутатор или коммутационный концентратор, представляет собой сетевое устройство, которое соединяет устройства вместе в компьютерной сети с помощью коммутации пакетов для приема, обработки и пересылки данных на устройство назначения.

Коммутатор работает на канальном уровне (уровень 2) модели OSI и использует MAC-адреса устройств для пересылки пакетов данных. Когда коммутатор получает пакет данных от устройства, он проверяет MAC-адрес получателя пакета и пересылает его на соответствующий порт, который ведет к устройству назначения. Этот процесс известен как "переключение".

Коммутаторы используются для создания сетевой инфраструктуры, позволяющей нескольким устройствам взаимодействовать друг с другом. Они более продвинуты, чем концентраторы, которые просто транслируют все пакеты данных на все подключенные устройства, независимо от места назначения. В отличие от этого, коммутаторы могут выборочно пересылать пакеты данных предполагаемому получателю, уменьшая перегрузку сети и повышая общую производительность сети.

Коммутаторы бывают различных форм и размеров, от небольших настольных коммутаторов, которые подключают несколько устройств, до крупных коммутаторов корпоративного уровня, которые могут подключать тысячи устройств. Они также могут поддерживать различные функции, такие как виртуальные локальные сети (VLAN), качество обслуживания (QoS) и зеркальное отображение портов, среди прочего.

Для малых локальных сетей организаций можно рассмотреть следующие модели межсетевых экранов-маршрутизаторов российского производства:

1. Фирменный маршрутизатор ER-78UW-16 с функциями файрволла, VPN и защиты от DDoS-атак.

2. APS-8003 для средних предприятий, обеспечивающий высокую производительность и эффективность.

3. Jet-ASR826 для малых и средних предприятий, обладающий удобным интерфейсом и надежной защитой данных.

4. Неос IDM /160 для сетей любого масштаба с функциями антивирусной защиты и фильтрации контента.

На рынке российского производства существует несколько вариантов коммутаторов на 48 портов. Некоторые из популярных моделей включают в себя:

1. "MSH-1048" от компании Медиасистемы – это коммутатор со встроенным блоком питания и возможностью управления через веб-интерфейс. Он поддерживает Gigabit Ethernet и имеет высокую пропускную способность.

2. "Avtomatika AS-4000" от компании Автоматика – это надежный коммутатор с 48 портами, который поддерживает стандарты IEEE 802.3 и имеет возможность управления через консольный порт.

3. "ELTEX MES2400" от компании Элтекс – это коммутатор с 48 портами, который поддерживает стандарты 10/100/1000 Mbps Ethernet и имеет встроенный блок питания.

Заключение

Выбор отечественного сетевого оборудования для локальной сети имеет ряд преимуществ, включая совместимость с отечественными стандартами, высокий уровень поддержки и сервиса, более низкую цену и высокое качество. При выборе сетевого оборудования для локальной сети, рекомендуется рассмотреть отечественные варианты, которые могут лучше всего соответствовать вашим потребностям и требованиям. При выборе отечественного сетевого оборудования, необходимо учитывать такие критерии, как производительность, функциональность, надежность и цена. Среди

известных производителей отечественного сетевого оборудования можно выделить ЗАО "Сатурн", ООО "НПФ "ЭЛВИС-Телеком", ОАО "РТИ" и ООО "Техносерв".

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. "Выбор сетевого оборудования для локальной сети". Журнал "Сетевые технологии", №3, 2021 г.

2. Выбор сетевого оборудования для малого бизнеса". Журнал "Бизнес-информация", №6, 2021 г.

3. "Как выбрать сетевое оборудование для домашней сети". Журнал "Домашний компьютер", №7, 2021 г.

4. "Отечественное сетевое оборудование: преимущества и недостатки". Журнал "Информационные технологии", №4, 2021 г.

5. "Сетевое оборудование для локальных сетей: характеристики и выбор". Журнал "Компьютерные технологии", №5, 2021 г.

УДК 004.89

ХРАМОВА А.А.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ИНТЕЛЛЕКТУАЛЬНЫЙ ЧАТ-БОТ В СИСТЕМЕ ИНФОРМАЦИОННОЙ ПОДДЕРЖКИ ДЛЯ ЭЛЕКТРОННОЙ ИНФОРМАЦИОННО-ОБРАЗОВАТЕЛЬНОЙ СРЕДЫ

Рассматриваются теоретические аспекты обучаемости интеллектуальных чат-ботов в системе информационной поддержки для электронной информационно-образовательной среды. Описываются методы обучения, алгоритмы, применяемые для развития и улучшения работы чат-бота, а также принципы работы и примеры использования искусственного интеллекта в подобных системах.

В современном мире использование чат-ботов в образовательных целях становится все более актуальным и востребованным. Интеллектуальные чат-боты представляют собой мощный инструмент для улучшения процесса обучения и предоставления информационной поддержки. Данная статья посвящена изучению теоретических аспектов обучаемости чат-бота в контексте электронной информационно-образовательной среды. Рассмотрение методов обучения, алгоритмов машинного обучения и принципов работы искусственного интеллекта в данном контексте позволит лучше понять возможности и перспективы развития подобных систем.

Электронная информационно-образовательная среда (ЭИОС) – это совокупность цифровых ресурсов и инструментов, которые способствуют обучению и развитию, обеспечивая доступ к образовательным материалам и коммуникацию между участниками образовательного процесса, стимулируя самостоятельное обучение [1].

Теоретические основы обучаемости чат-ботов

Интеллектуальные чат-боты в системе информационной поддержки для ЭИОС являются важным элементом для обеспечения эффективного взаимодействия с пользователями и предоставления им необходимой информации. Одним из ключевых аспектов разработки таких чат-ботов является их обучаемость, то есть способность улучшать свои интеллектуальные способности на основе полученного опыта и данных.

Теоретические основы обучаемости чат-бота включают в себя обзор существующих моделей машинного обучения и методов, применяемых для повышения интеллектуальных способностей чат-ботов. Среди таких моделей можно выделить нейронные сети, глубокое обучение, алгоритмы обработки естественного языка и многие другие [2].

Методы обучения чат-ботов

Методы обучения чат-ботов являются ключевым аспектом развития его интеллектуальных способностей. Существует несколько основных подходов к обучению чат-ботов, каждый из которых имеет свои особенности и преимущества.

1. Обучение с учителем (supervised learning) - это метод машинного обучения, при котором модель обучается на размеченных данных, где каждому входному примеру соответствует правильный выходной ответ. В случае чат-бота, это означает, что ему предоставляются примеры диалогов или сообщений, где каждое входное сообщение имеет соответствующий правильный ответ. Чат-бот анализирует эти примеры, изучает закономерности и паттерны в данных и стремится предсказывать правильные ответы на основе обучающих данных.

Преимущества обучения с учителем для чат-ботов включают возможность точного и быстрого обучения на размеченных данных, способность к адаптации к новым ситуациям на основе обучающих примеров и возможность корректировки ответов на основе обратной связи. Однако, недостатком этого метода является необходимость наличия большого объема размеченных данных для эффективного обучения чат-бота.

2. Обучение без учителя (unsupervised learning) - это метод машинного обучения, при котором чат-бот обучается на неструктурированных данных, не имеющих предварительной разметки или явных правильных ответов. В отличие от обучения с учителем, где модель учится на парах «входные данные - правильный ответ», в случае обучения без учителя модель самостоятельно исследует данные, выявляет в них скрытые закономерности и структуры.

Преимущества обучения без учителя включают возможность обработки больших объемов неструктурированных данных, а также способность модели самостоятельно выявлять интересные и важные аспекты информации. Чат-бот, обученный методами без учителя, может автоматически кластеризовать данные, выделять тематические группы, анализировать тексты на семантическом уровне и даже генерировать новые знания на основе имеющихся данных.

Однако, обучение без учителя также имеет свои ограничения, такие как сложность интерпретации полученных результатов, необходимость более тщательного контроля за процессом обучения и возможные проблемы с переобучением. Тем не менее, при правильном применении методов обучения без учителя чат-бот может значительно улучшить свои интеллектуальные способности и стать более адаптивным к различным сценариям общения с пользователями.

3. Обучение с подкреплением (reinforcement learning) - это метод обучения, в котором чат-бот обучается путем взаимодействия с окружающей средой и получения награды или наказания за свои действия. В процессе обучения с подкреплением чат-бот самостоятельно исследует окружающую среду, принимает решения и корректирует свое поведение на основе полученных результатов. Целью является максимизация общей награды, которую чат-бот получает за свои действия. Этот метод позволяет чат-боту учиться на практике, оптимизируя свое поведение через метод проб и ошибок, что позволяет ему адаптироваться к изменяющейся среде и совершенствовать свои навыки и знания.

Каждый из этих методов имеет свои сильные и слабые стороны, и выбор подходящего метода зависит от конкретной задачи и условий обучения чат-бота. Комбинация различных методов обучения может помочь создать интеллектуального чат-бота, способного эффективно взаимодействовать с пользователями в ЭИОС [3].

Алгоритмы машинного обучения

Алгоритмы машинного обучения играют важную роль в развитии интеллектуальных чат-ботов. Среди основных алгоритмов, применяемых для обучения чат-ботов, можно выделить следующие.

1. Нейронная сеть – это математическая модель, вдохновленная работой человеческого мозга, используемая для обработки информации и принятия решений на основе входных данных. Нейронные сети состоят из нейронов, которые взаимодействуют друг с другом через веса, определяющие силу связей между нейронами.

В контексте чат-ботов, нейронные сети могут быть обучены на больших объемах текстовых данных, чтобы выявлять паттерны и зависимости в тексте. Это позволяет чат-ботам понимать вопросы пользователей, анализировать контекст и генерировать соответствующие ответы. Нейронные сети могут быть применены как для классификации текста, так и для генерации текста на основе обучающего набора данных.

Использование нейронных сетей в чат-ботах позволяет им обучаться на большом количестве данных, улучшая качество ответов и способность чат-бота взаимодействовать с пользователями.

2. Алгоритмы обработки естественного языка (Natural Language Processing, NLP) представляют собой совокупность методов и техник, которые позволяют компьютерам анализировать, понимать и генерировать текстовую информацию на естественных языках. Алгоритмы NLP играют ключевую роль в обеспечении эффективного взаимодействия между человеком и машиной.

Основные задачи, которые решаются с помощью алгоритмов NLP, включают в себя:

1. разбиение текста на отдельные слова или фразы (токены);
2. приведение слов к их нормальной (словарной) форме;
3. определение частей речи для каждого слова в предложении;
4. определение структуры предложения и связей между словами;
5. выявление смысловых связей между словами и фразами;
6. выделение из текста именованных объектов, таких как имена людей, мест, организаций и т.д.;
7. создание естественно звучащих ответов и реплик на основе анализа входных данных.

Алгоритмы обработки естественного языка используются для улучшения качества диалогов чат-ботов, автоматизации ответов на запросы пользователей и создания более естественного и понятного пользовательского опыта. Они позволяют чат-ботам эффективно обрабатывать и анализировать текстовую информацию, что делает их более интеллектуальными и адаптивными к потребностям пользователей.

3. Методы классификации и кластеризации являются важными инструментами в области машинного обучения и обработки естественного языка.

Метод классификации позволяет чат-ботам разделять данные на различные категории или классы в соответствии с заданными признаками. Например, чат-бот может классифицировать сообщения пользователей как вопросы, жалобы, запросы на информацию и т.д. Это помогает чат-боту понимать намерения пользователя и предоставлять соответствующие ответы.

Метод кластеризации позволяет группировать данные на основе их сходства без заранее заданных категорий. Чат-бот может использовать кластеризацию для выявления общих тем или паттернов в сообщениях пользователей, что помогает ему лучше понимать контекст и предлагать более релевантные ответы.

Оба метода, классификации и кластеризации, играют важную роль в развитии интеллектуальных чат-ботов, позволяя им эффективно анализировать и обрабатывать текстовую информацию для улучшения качества обслуживания пользователей.

Использование различных алгоритмов машинного обучения позволяет создавать интеллектуальные чат-боты, способные эффективно взаимодействовать с пользователями и предоставлять качественную информационную поддержку в ЭИОС [4].

Использование искусственного интеллекта в ЭИОС

Чат-боты, основанные на искусственном интеллекте, представляют собой эффективный инструмент для улучшения доступа к информации и поддержки обучения студентов.

Искусственный интеллект в ЭИОС может быть использован для создания персонализированных образовательных платформ, где чат-боты могут предоставлять индивидуализированную помощь студентам.

Примером использования искусственного интеллекта в ЭИОС может быть система онлайн-обучения, где чат-боты играют роль виртуальных помощников для студентов. Например, чат-бот может помочь студенту найти необходимую информацию по определенному предмету, объяснить сложный материал, предложить дополнительные материалы для изучения или даже провести тестирование знаний. Такой подход делает обучающий процесс более интерактивным и доступным для студентов, а также позволяет учителям более эффективно организовывать образовательный процесс.

Таким образом, использование искусственного интеллекта в ЭИОС открывает новые возможности для улучшения качества

обучения, повышения доступности образования и поддержки студентов в учебном процессе.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Боровская Е.В. Основы искусственного интеллекта: Учебное пособие/ Боровская Е.В., Давыдова Н.А. – Электрон. текстовые данные. – М.: Лаборатория знаний, 2020. – 128 с.

2. Бурняшов, Б.А. Электронная информационно-образовательная среда учреждения высшего образования: монография / Б. А. Бурняшов. – Краснодар: Южный институт менеджмента, 2017. – 216 с.

3. Павлов, С. Н. Системы искусственного интеллекта. Часть 1: учебное пособие / С. Н. Павлов. – Томск: Томский государственный университет систем управления и радиоэлектроники, Эль Контент, 2011. – 176 с.

2. Пенькова, Т. Г. Модели и методы искусственного интеллекта: учебное пособие / Т. Г. Пенькова, Ю. В. Вайнштейн. – Красноярск: Сибирский федеральный университет, 2019. – 116 с.

УДК 004.89

ХРАМОВА А.А., САПРЫКИН А.Н.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

АВТОМАТИЗАЦИЯ ИНФОРМАЦИОННОЙ ПОДДЕРЖКИ В ЭЛЕКТРОННОЙ ИНФОРМАЦИОННО-ОБРАЗОВАТЕЛЬНОЙ СРЕДЕ

Рассматриваются основные возможности автоматизации информационной поддержки в электронной информационно-образовательной среде с использованием чат-ботов.

Электронная информационно-образовательная среда (ЭИОС) – это совокупность информационных технологий, ресурсов и сервисов, используемых для обучения и образования. Она включает в себя различные электронные учебники, онлайн курсы, образовательные платформы, интерактивные приложения, вебинары и другие средства, которые помогают студентам и преподавателям в образовательном процессе.

Введение в тему автоматизации информационной поддержки в ЭИОС является ключевым шагом к оптимизации образовательных процессов.

В современном мире чат-боты становятся все более популярным инструментом для обеспечения информационной поддержки в образовательных учреждениях, облегчая доступ к необходимой информации и повышая эффективность взаимодействия между студентами, преподавателями и администрацией. ЭИОС требует инновационных подходов к автоматизации информационной поддержки, и чат-боты представляют собой перспективное решение для улучшения образовательного процесса и удовлетворения потребностей его участников [1].

В данном контексте важно изучить преимущества использования чат-ботов, успешные примеры и вызовы, с которыми сталкиваются разработчики, чтобы понять перспективы развития автоматизации информационной поддержки в данной среде.

Преимущества использования чат-ботов

Роль чат-ботов в обеспечении информационной поддержки в ЭИОС является значительной и важной.

Чат-боты представляют собой программное обеспечение, способное взаимодействовать с пользователями через текстовые или голосовые сообщения, что делает их отличным инструментом для обеспечения информационной поддержки в ЭИОС.

Чат-боты могут выполнять различные функции, такие как предоставление информации о курсах, расписании занятий, ответы на часто задаваемые вопросы, помощь в организации учебного процесса и многое другое. Благодаря своей доступности и возможности работать круглосуточно, чат-боты способствуют повышению эффективности образовательного процесса и улучшению общего опыта обучения.

Использование чат-ботов для обеспечения информационной поддержки также способствует персонализации обучения, поскольку они могут предлагать индивидуализированные рекомендации и поддержку каждому учащемуся. Благодаря возможности быстро и точно отвечать на запросы студентов, чат-боты способствуют улучшению общей доступности информации и содействуют эффективному обучению в ЭИОС [2].

Примеры внедрения чат-ботов в ЭИОС

Анализ успешных примеров внедрения чат-ботов в ЭИОС подчеркивает их значительный вклад в улучшение информационной поддержки. Например, университеты начали использовать чат-ботов для автоматизации ответов на часто задаваемые вопросы студентов, что позволяет снизить нагрузку на персонал и улучшить доступность информации. Также, платформы онлайн-обучения внедряют чат-ботов

для помощи студентам в процессе обучения, предоставляя персонализированные рекомендации по материалам и заданиям. Эти успешные примеры демонстрируют потенциал чат-ботов в сфере образования для улучшения информационной поддержки и повышения эффективности образовательного процесса.

Приведем примеры успешной реализации чат-ботов в ЭИОС.

1. Яндекс разработал чат-бота «Алиса в школе», который помогает школьникам и их родителям получать информацию о расписании, домашних заданиях, оценках и других школьных вопросах.

2. Образовательная платформа «Skyeng» использует чат-ботов для обучения английскому языку. Чат-боты помогают студентам практиковать разговорные навыки, получать обратную связь и дополнительные материалы для изучения.

3. Сервис «Лекторий» предоставляет чат-бота, который помогает студентам подбирать курсы и обучающие материалы, отвечает на вопросы по темам и предметам обучения.

Эти примеры демонстрируют, как чат-боты успешно интегрируются в образовательную среду, улучшая доступ к информации, обучению и общению в онлайн формате.

Инструменты для создания чат-ботов

Для создания чат-ботов в ЭИОС существует ряд технологий и инструментов.

Создание чат-ботов с помощью API и веб-интерфейсов – это популярный способ разработки чат-ботов, который позволяет интегрировать их с различными платформами сервисами.

API предоставляет доступ к функциональности программного обеспечения и позволяет чат-боту взаимодействовать с другими приложениями и сервисами. Это обеспечивает гибкость и расширяемость функциональностиы.

Чтобы интегрировать чат-бота с различными платформами, разработчики могут использовать специализированные инструменты и сервисы, предоставляемые провайдерами чат-ботов. Некоторые из популярных платформ для создания включают в себя Dialogflow, Microsoft Bot Framework, IBM Watson Assistant, и другие. Эти платформы предоставляют различные инструменты для разработки, обучения и развертывания чат-ботов на различных мессенджерах и платформах [3].

Для реализации чат-ботов в различных мессенджерах также существуют специализированные библиотеки и фреймворки. Можно использовать такие инструменты, как Telegraf для Telegram, vk_api для

ВКонтакте или Chat-API, WhatsApp Business API, для WhatsApp. Эти библиотеки предоставляют готовые решения и инструменты для создания и развертывания чат-ботов на соответствующих мессенджерах.

Кроме того, для программирования чат-ботов можно использовать различные языки программирования, такие как Python, JavaScript, Java и другие. Разработчики могут использовать язык Python для написания логики обработки запросов и формирования ответов чат-бота. Это позволяет создавать персонализированные и многофункциональные чат-боты с учетом специфики образовательной среды и потребностей пользователей.

Анализ проблем и трудностей при разработке

Анализ проблем и трудностей, с которыми сталкиваются разработчики чат-ботов для образовательных целей, представляет собой важный аспект в процессе создания и внедрения чат-ботов в ЭИОС. Одной из основных проблем является необходимость точного определения потребностей пользователей и обучающихся, чтобы чат-бот мог эффективно реагировать на запросы и предоставлять подходящую информацию. Другой вызов связан с обеспечением высокой степени персонализации, чтобы чат-бот мог адаптироваться к уникальным потребностям каждого пользователя и обеспечивать индивидуализированную информационную поддержку. Необходимо также уделять должное внимание этическим и конфиденциальным вопросам, таким как сохранение конфиденциальности данных и обеспечение безопасности информации, чтобы поддерживать доверие пользователей к чат-боту. Разработчики также сталкиваются с вызовом поддержания актуальности и качества контента, который предоставляется через чат-бота, поскольку образовательная информация постоянно обновляется и развивается.

Важно непрерывно совершенствовать и обновлять чат-бота, чтобы он оставался актуальным и полезным для пользователей. Эффективное решение этих вызовов и проблем требует комплексного подхода к проектированию и разработке чат-ботов для образовательных целей, а также постоянного внимания к обратной связи пользователей и обучающихся для улучшения качества информационной поддержки, предоставляемой чат-ботом.

Перспективы автоматизации информационной поддержки в ЭИОС

В последние годы автоматизация информационной поддержки в ЭИОС стала ключевым направлением развития образовательных

технологий. Перспективы данной области весьма обширны и включают в себя ряд важных аспектов.

1. Интеграция и улучшение функциональности. С развитием и совершенствованием технологий машинного обучения и обработки естественного языка, чат-боты становятся все более интеллектуальными и способными адаптироваться к потребностям пользователей. Будущее автоматизации информационной поддержки в образовательной сфере связано с улучшением способностей чат-ботов в обработке запросов, предоставлении качественной информации и поддержке студентов и преподавателей.

2. Персонализация и адаптация. Одним из ключевых направлений развития автоматизации информационной поддержки является создание персонализированных чат-ботов, способных адаптироваться к индивидуальным потребностям пользователей. Это позволит улучшить качество образовательного процесса, предоставляя студентам индивидуализированную поддержку и помощь [4].

3. Интеграция с другими технологиями. Будущее автоматизации информационной поддержки в ЭИОС также связано с интеграцией чат-ботов с другими технологиями, такими как системы управления обучением, аналитика образовательных данных и виртуальная реальность. Это позволит создать более эффективные и инновационные образовательные среды.

4. Развитие искусственного интеллекта. С развитием искусственного интеллекта и машинного обучения, чат-боты будут способны предсказывать потребности пользователей, предлагать решения и помогать им в реальном времени. Это повысит эффективность образовательного процесса и улучшит опыт обучения.

В будущем автоматизация информационной поддержки в ЭИОС обещает значительное улучшение образования, более широкий доступ к образовательным ресурсам и индивидуализацию обучения для каждого учащегося.

Чат-боты предоставляют эффективный и доступный способ обеспечения студентов информационной поддержкой, улучшая коммуникацию и упрощая доступ к необходимой информации. Преимущества использования чат-ботов включают повышение эффективности обучения и сокращение нагрузки на преподавателей.

Примеры успешной реализации чат-ботов в ЭИОС, таких как университеты и онлайн-курсы, подтверждают их потенциал для улучшения образовательного опыта.

В перспективе развития автоматизации информационной поддержки в электронной информационно-образовательной среде

следует ожидать дальнейшего расширения использования чат-ботов, интеграции с новыми технологиями, такими как искусственный интеллект и машинное обучение, а также развития персонализированных решений для улучшения образовательного процесса. Важно продолжать исследования и инновации в этой области, чтобы обеспечить студентам эффективное и качественное образование в электронной среде [5].

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Бурняшов, Б.А. Электронная информационно-образовательная среда учреждения высшего образования: монография / Б.А. Бурняшов. – Краснодар: Южный институт менеджмента, 2017. – 216 с.

2. Джанарсанам, С. Разработка чат-ботов и разговорных интерфейсов / С. Джанарсанам; перевод М. Райтман. – Москва: ДМК Пресс, 2019. – 340 с.

3. Зенков, А.Р. Цифровизация образования: направления, возможности, риски / А.Р. Зенков // Вестник Воронежского государственного университета. Серия: Проблемы высшего образования. – 2020. – № 1. – С. 52-55.

4. Минаев, А.И. Образовательный процесс классического университета с применением дистанционных образовательных технологий: реалии, достижения, проблемы / А.И. Минаев, Е.А. Кирьянова, О.Н. Исаева. // Вестник Воронежского государственного университета. Сер.: Проблемы высшего образования. – 2020. – № 3. – С. 56-60.

5. Письменский Г.И. Основы проектирования электронной информационно-образовательной среды: монография/ Г.И. Письменский, В.В. Киселев, Л.В. Неровный, С.В. Сафонова. – Москва: РУСАЙНС, 2024. – 104с.

УДК 004.514.6

ЮРАСОВ В.А.

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

РАЗРАБОТКА КОНВЕРТОРА ГРАФИЧЕСКИХ ДРАКОН-СХЕМ НА ЯЗЫКЕ C++, ИХ ХРАНЕНИЯ И ЧТЕНИЯ В ФОРМАТЕ XML

В статье рассматривается приложение конвертор ДРАКОН-схем в код на языке C++, а также их формат хранения.

Дракон-схемы являются более удобными в восприятии чем обычные блок-схемы, так как обладают следующими особенностями, влияющими на восприятие:

- Отсутствие лишних линий и стрелок;
- Входы и выходы элементов схемы расположены по правилам;
- Присутствует графическая стандартизация циклов;
- Число изгибов соединительных линий минимально;
- Присутствие явно выраженного главного маршрута – шампура.

ДРАКОН — графический язык. Символами языка являются геометрические фигуры, называемые «иконами». Всего имеется 31 икона. Имеются некоторые различия с символами блок-схем алгоритмов по ГОСТ 19.701-90. В связи с этим, людям, привыкшим к обычным блок-схемам, некоторые иконы могут казаться непривычными, поскольку их вид будет ассоциироваться с другими.

Иконы обязательно должны быть заполнены текстом. Если текст отсутствует (икона пустая), значит, это ошибка.

Зачастую используются две иконы: “Действие” и “Вопрос”.

Икона “Действие” совпадает с одноименным символом в блок-схемах. В ней пишут команды в повелительном наклонении. Но разрешенным способом подсоединения линий является лишь один вариант – входная линия сверху, выходная снизу.

Икона “Вопрос” значительно отличается в графическом представлении от одноименного символа в ГОСТе 19.701-90. Она представляется не ромбом, а шестиугольником. Также имеются различия в присоединении входных и выходных линий. Согласно ГОСТу в блок-схемах разрешено подсоединение выходных линий справа и слева, в то время как подсоединение выходных линий в дракон-схемах разрешено только внизу (главный маршрут) и справа (боковой маршрут).

На рисунке 1 представлены основные иконы и их назначение.

Формат хранения дракон-схем

При разработке приложения конвертора дракон-схем для хранения дракон-схем был выбран формат XML.

Заметим, в файле дракон-схемы содержатся три основных раздела:

1. Список подключаемых файлов;
2. Список глобальных переменных;
3. Описание процедур или функций (может быть несколько в одном файле).

	Икона	Название иконы	Пояснение
1		Заголовок	В иконе «Заголовок» пишут полное название алгоритма, например: «Аварийное выключение ракетного двигателя»
2		Конец	В этой иконе пишут слово «Конец»
3		Действие	Указывают действие или команду, которую должен выполнить компьютер или человек
4		Вопрос	Да-нетный вопрос, т. е. вопрос, на который можно ответить либо Да, либо Нет. Все другие ответы запрещены
5		Выбор	Фраза (или вопрос), приглашающая выбрать один из вариантов
6		Вариант	Здесь пишут один из вариантов. (Число рассматриваемых вариантов равно числу икон «Вариант»)
7		Имя ветки	Эта икона обозначает начало ветки. В ней находится название ветки. (Ветка — это структурная часть алгоритма)
8		Адрес	Икона «Адрес» обозначает конец любой ветки, кроме последней. Она показывает, в какую следующую ветку надо идти
9		Вставка	Икона «Вставка» говорит, что в этом месте из алгоритма вынут «кусок», который перенесён в другое место. В иконе пишут название этого «куска»
10		Полка	Сверху пишут указание, например: «Установить признак», снизу пояснение, например: «Можно ехать налево»
11		Формальные параметры	Икона присоединяется справа к иконе Заголовок
		Пояснение	Любые сведения, поясняющие икону, находящуюся слева от данной
12		Начало цикла ДЛЯ	Для цикла for пишут переменную цикла, ее начальное и конечное значения, шаг. Иконы 12 и 13 используются также в цикле foreach
13		Конец цикла ДЛЯ	Внутри иконы пишут «Конец цикла». Запрещено оставлять икону пустой
14		Вывод	Вывод данных, передача информации. Выдача команд на исполнительные органы в системах реального времени
15		Простой вывод	Икона «Простой вывод» позволяет упростить дракон-схему при необходимости
16		Ввод	Ввод данных, прием информации
17		Простой ввод	Икона «Простой ввод» позволяет упростить дракон-схему при необходимости

Рисунок 1 – Иконы языка ДРАКОН

Приведем ниже список тэгов, использующихся для сохранения схемы, а также поясним их назначение:

- algorithm – корневой тэг;
- includedrg, includecpp – тэги содержащие информацию об одном подключаемом файле, использующимся для работы описываемого алгоритма;
- name, comment – тэги содержащие название и комментарий о подключаемом файле соответственно;
- global – тэг для хранения информации об одной глобальной переменной;
- type, name, comment - тэги содержащие тип, название и комментарий о глобальной переменной;
- proc – тэг, в котором содержатся другие тэги описывающие процедуру или функцию.

Для описания процедуры или функции используются следующие тэги:

- branch – тэг содержащий в себе остальные тэги описывающие части схемы, соответствует прямому участку схемы (главный или боковые маршруты);
- cppcomand – тэг описывающий икону “Действие”;
- if – тэг для описания решения, оно состоит из иконы “Вопрос” и ветвей для соответствующих исходов;
- cond – тэг описывающий условие содержащееся в иконе “Вопрос”, обязательно находится внутри тэга if;
- yes, no – тэги содержащие ветви соответствующих исходов в решении, также обязательно находятся внутри тэга if;
- silhouette – тэг для описания силуэта, содержит в себе два или более тэгов label;
- label – тэг описывающий икону “Адрес”, содержит в себе один главный тэг branch и собственно само название, которое указывает на то какая ветка будет выполнена следующей после выполнения текущей.

Пример содержания файла дракон-схемы, использующей несколько тэгов, описанных выше, приведен ниже (троеточием обозначены места, в которых возможны другие тэги такого же уровня).

```
<!DOCTYPE DRAGON>
<algorithm version="1.0" encoding="utf-8">
    <includedrg>
        <name>Par1-100</name>
        <comment>
```

```
            Первый подключаемый файл
        </comment>
    </includedrg>
    <includecpp>
        <name>netinet/in.h</name>
        <comment>
          Второй подключаемый файл
        </comment>
    </includecpp>
    <global>
        <type> int </type>
        <name>var_G </name>
        <comment>глоб 1.1 переменная</comment>
    </global>
    ...
    <proc>
             <type>ЦЕЛЫЙ8</type>
             <name>main</name>
             <local>
                    <type>float </type>
                    <name>var_L </name>
                    <comment>лок. перем.</comment>
             </local>
             <branch>
            <cppcommand>int a = 4;</cppcommand>
            ...
            <cppcommand>int b = 7;</cppcommand>
            ...
             </branch>
    </proc>
</algorithm>
```

Основные классы приложения

Для понимания того, как работает сохранение в приложении, рассмотрим основные классы, использующиеся в коде для представления дракон-схем в целом и их частей по отдельности. Приведем ниже их список и небольшое описание:

- AlgorithmPlugin – предназначен для представления дракон-схемы целой функции или процедуры;
- BranchPlugin – описывает ветвь схемы, содержит в себе ссылки на другие части схемы, которые могут располагаться на ветви;
- CppCommandPlugin – описывает икону “Действие”;
- IfPlugin – описывает решение и соответственно икону “Вопрос”, также содержит ссылки на две ветви, соответствующие двум исходам (“Да” и “Нет”);

• LabelPlugin – описывает икону “Адрес” и также содержит ссылку на главную ветвь;

• SilhouettePlugin – необходим для реализации силуэта, содержит в себе ссылки на два и более объекта класса LabelPlugin.

Сохранение дракон-схем

В разрабатываемом приложении для сохранения схем в XML файл осуществляется согласно следующей последовательности действий:

1. Сохраняется информация о подключенных файлах;

2. Записываются глобальные переменные;

3. В объекте класса AlgorithmPlugin вызывается функция GetXML которая проходит по всем частям схемы (то есть по всем объектам классов, описанных выше и содержащихся в связанном списке) и возвращает строку, составленную из всех тэгов, описывающих соответствующие части схемы.

Код функции приведен ниже:

```
QString algorithmPlugin::GetXML(int countTab)
{
    QString retStr;
    retStr += "\n" + addTab(countTab);
    retStr += "<"+this->getPluginName()+">";
    retStr += "\n" + addTab(countTab+1);
    retStr += "<type>" + this->type + "</type>" ;
    retStr += "\n" + addTab(countTab+1);
    retStr += "<name>" + this->name + "</name>" ;
    if (body != nullptr)
        retStr += body->GetXML(countTab+1);
    retStr += "\n" + addTab(countTab);
    retStr += "</"+this->getPluginName()+">" ;
    return retStr;
}
```

Заметим, что у каждого класса своя реализация функции GetXML, но общая структура функции сохраняется – то есть проверяется наличие “подчиненной” части схемы, например в классе BranchPlugin проверяется содержит ли ветвь какие-либо иконы, и эта функция вызывается уже последовательно у всех объектов (икон), находящихся в ветви.

Конвертация дракон-схем

Одной из главных функций разрабатываемого приложения является конвертация дракон-схем в код на языке C++.

Конвертация происходит в два файла – заголовочный (расширение .h) и основной (расширение .cpp). Можно выделить три основных этапа конвертации:

1. Получение текста заголовочного файла (.h);

2. Его добавление в текст заголовочного (.h) и основного файлов (.cpp);

3. В объекте класса AlgorithmPlugin вызывается функция GetCPP которая проходит по всем частям схемы и возвращает код, составленный из всех частей, затем происходит запись полученного кода в основной файл (.cpp).

Рассмотрим подробнее первый этап, на нём происходит получение информации о всех подключенных файлах (их названия). Затем формируются строки, отвечающие за подключение файлов - #include название_файла. Далее происходит формирование строк заголовков методов (тех функций или процедур, схемы которых были созданы в приложении). На этом первый этап заканчивается.

На втором этапе происходит добавление текста, полученного на прошлом этапе в заголовочный файл, а также строки, отвечающие за подключение файлов, добавляются в основной файл.

На третьем этапе формируется текст методов для основного файла. Для этого в объекте класса algorithmPlugin вызывается функция GetCPP. Код данной функции приведен ниже:

```
QString algorithmPlugin::GetCPP(int countTab){
    QString retStrC;
    retStrC += "\n" + addTab(countTab);
    retStrC += common->getLatinType(this->type) + " " +
this->name + "()";
    if (body != nullptr)
        retStrC += body->GetCPP(countTab);
    return retStrC;
}
```

Сначала формируется заголовок метода из возвращаемого типа и его названия. Затем если в дракон схеме присутствуют элементы, то происходит вызов одноименной функции у объекта класса этого элемента. В примере схемы, которая использовалась для примера файла сохранения схемы, этот элемент будет ветвью (класс BranchPlugin).

Рассмотрим код функции GetCPP у класса BranchPlugin, приведенный ниже:

```
QString branchPlugin::GetCPP(int countTab){
    QString brStrC;
    brStrC += "\n" + addTab(countTab) + "{";
    block_t *lclBrBody = body;
    while (lclBrBody != nullptr)
```

```
    {
        brStrC += lclBrBody->GetCPP(countTab+1);
        lclBrBody = lclBrBody->getNext();
    }
    brStrC += "\n" + addTab(countTab) + "}";
    return brStrC;
}
```

Сначала идёт проверка на присутствие икон в ветви. Если хотя бы одна икона присутствует, то происходит вызов функции GetCPP уже у объекта класса соответствующего этой иконе. В простейшем случае это может быть икона “Действие” – класс CppCommandPlugin. Далее осуществляется переход к следующей иконе в ветви. Если икон больше нет, то работа данной функции завершается и возвращаются строки описывающие ветвь в соответствующей дракон-схеме.

После прохождения по всем частям схемы, полученные строки кода в функции GetCPP класса algorithmPlugin, записываются в основной файл с расширением cpp. Пример кода основного файла, полученного при конвертации приведён ниже:

```
#include <netinet/in.h> //Второй подключаемый файл
#include "Par1-100" //Первый подключаемый файл
int8_t main()
{
    int a = 4;
    int b = 7;
}
```

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Пароджанов В. Д. Алгоритмы и жизнеритмы на языке ДРАКОН. Разработка алгоритмов. Безошибочные алгоритмы. – М., 2019. – 374 с.

УДК 04.004

ЯКОБС Е.Г.
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ЕГО РАЗВИТИЕ

Нейросети видят мир в виде линий и упрощают его для преобразования в числа. Нейросети работают чуть тоньше и видят мир в виде линий, так как такие изображения легче всего превратить в числа, а для нейросетей весь мир – это числа, картинки, видео, текст, аудио. Математически доказано, что нейросеть может

решить любую задачу, которую можно описать функцией. В теории они всемогущи, то есть могут практически всё. Нейросети могут предсказывать молекулярную структуру материалов, находить противоядия от ядов, понимать физику нашего мира и даже решать капчи.

Капча – полностью автоматизированный публичный тест Тьюринга, чтобы отличить компьютеры от людей. Вводя капч, вы буквально помогали ИИ от Google обрести зрение в 2005 году. В капче появляются сразу два слова, первое пропускает вас на сайт, а второе на самом деле на проверку не влияет. Даже если вы вводили его неправильно вас всё равно пропускали на сайт. Тогда зачем нужно было второе слово? Оно отсканировано с книг, но ИИ не мог его распознать. Например, слово было зачёркнуто или лежало на сгибе. Предполагалось, что, если пользователь введёт первое слово правильно, то и второе тоже. А если так поступают девять из десяти пользователей, то искусственный интеллект начинает понимать это слово. Пользователи, которые вводили эти капчи правильно помогали искусственному интеллекту за 4 дня оцифровать годовой архив газеты New York Times. Теперь в капче больше не показывают вырезки из газет, потому что все мы дружно с вами научили искусственный интеллект читать книги и уметь читать, что теперь не отличает машину от человека, поэтому на капчах теперь картинки. Почему на них именно светофоры, пешеходные переходы, автобусы и велосипеды? Потому, что картинки с переходами он берёт со своих Google карт, а вы, отмечая их на капче учите беспилотный автомобиль Гугла различать все объекты в реальности.

Но что, если обучить нейросеть которая на вход будет получать не дорожные знаки и виды с улиц. Мы когда-то говорили про макияж, с которым ни одна нейросеть вас не узнает, то сейчас они сканируют ваше лицо при помощи инфракрасных лучей и какого оно цвета им вообще уже без разницы. Благодаря этим же алгоритмам квадрокоптеры могут снимать 3D модели целых городов. Благодаря этому наши города становятся умными и удобными ну и одновременно с этим прозрачные. Вам уже некуда скрыться. Простые камеры для наблюдения за обществом через 10 лет станут прошлым.

Представьте ситуацию: кто-то вытащил у вас кошелёк. Сейчас, полиции бы пришлось просмотреть ракурсы с разных камер и всматриваться в каждого прохожего рядом с жертвой. Но что если в будущем ещё до того, как вы сами обнаружите кражу кошелька искусственный интеллект успеет вычислить карманника и передать данные в полицию и делать это он будет не задумываясь для него это ещё одна функция – на входе куча толкающихся людей, на выходе –

варианты развития событий: 90% людей наткнулись на потерпевшего потому что был час-пик, 9,9 потому что залипали в телефоны и наткнулись случайно и 0,1% – это тот самый единственный человек, кто наткнулся нарочно, чтобы вытащить кошелёк.

Люди, когда понимают, что рисуют неправильно, зачем-то начинают всё усложнять. Всё потому, что наша память не фотографична и запоминает изображение, а потом пытается восстановить по нему общую картину, теряя детали прямо как плохой ЗИП архив. С этой задачей гораздо быстрее справится нейросеть, точнее, нейросеть архиватор. Такой алгоритм состоит из двух частей. Первая – энкодер. Он сжимает картинку по хитрым правилам, которые сам же и придумывает. Вторая – декодер. Он по тем же правилам разжимает картинку в оригинал, когда нейросеть обучится на миллионах изображений, например, лиц и портретов.

Мы упирается в закон термодинамики, фундаментальный принцип, что мы не можем получить что-то из ничего, а значит нужно дать нейросети что-нибудь. Например, цифровой шум, к которому она уже привыкла, потому что энкодер сжимал всё до цифрового шума. Просто подаём случайные пиксели декодеру и он, пытаясь восстановить оригинал создаёт изображения, которых до этого не существовало, потому что знает, как выглядит человек и как его собрать из шума. Понимаете, мы как будто бы давали нейросети наборы LEGO, которые она разбирала и собирала исходя из инструкции, а потом мы кинули ей случайные детальки, и она стала собирать знакомые фигуры сама без инструкций. Если попросить нейросеть нарисовать очаровательного кролика с большими глазами, то на основе случайного шума она нарисует сначала общее очертание кролика, а потом и мелкие детали. Из-за разного шума выходят разные кролики. Так работают диффузионные модели.

Нейросети могут создавать новые изображения, используя цифровой шум. Могут генерировать несуществующие изображения, которые сложно отличить от реальных. Нейросети могут быть обучены на текстах и использовать их для создания изображений. Могут быть использованы для чтения мыслей и создания изображений на основе активности мозга. Нейросети могут помочь в обучении, предоставляя индивидуальный подход и упрощая понимание сложных тем. Нейросети могут быть использованы для управления роботами и сортировки посылок на складах. Они могут решать задачи, с которыми раньше не сталкивались, и быть сильными во многих областях.

В будущем, искусственный интеллект сможет сканировать лица людей и определять их цвет кожи. Квадрокоптеры смогут снимать

трехмерные модели городов. Простые камеры для наблюдения за обществом станут прошлым. Создатели искусственного интеллекта не могут точно предсказать, что получится в итоге, и всегда есть риск, что ИИ поймет задачу по-другому. В случае с беспилотным автомобилем, ИИ может начать разгоняться до 300 км/ч, объезжать светофоры по тротуарам и сбивать пешеходов.

Помните, что, создавая нейросеть люди не знают точно, что получится у них в конце. По этой же логике предсказать что будет уметь gpt 5 очень сложно. Как бы тщательно мы не ставили цель перед искусственным интеллектом всегда есть риск, что он поймёт её по-другому. Например, мы создаём беспилотный автомобиль и в первый же день теста он разгоняется до 300 км/ч, объезжая все светофоры по тротуару и сбивая пешеходов. При этом значит надо прописать ему ограничение – говорим ему: «Держись в полосе, не превышай, и помеха справа». Это всё хорошо, но уже в следующей поездке он делает неожиданное открытие. Если включить заднюю передачу, то радары спереди не будут видеть людей и разметку, значит задом можно ехать, как угодно, а помеха справа становится помеха слева. И снова он делает то, что мы буквально его попросили, а не то что мы на самом деле имели в виду. Можно всё время держать руку на рубильники и, если что-то начнётся вырубить его. Создатели gpt даже иронично открыли вакансию человека, который будет дёргать за этот рубильник с зарплатой 300 500.000 долларов в год.

Мы просто позволяем алгоритмам вести себя за ручку, ведь в день мы принимаем до 35000 решений от того чтобы почесать руку до анализа рынка и поиска новых профессий и большая часть из этих решений - это мелочи рутины. Так почему бы не доверить и не делегировать их искусственному интеллекту. Но с каждым таким решением мы всё меньше оставляем свободы самим себе. Зачем писать тексты, когда его лучше напишет ИИ, зачем рисовать что-то в мире, когда есть ИИ. Мы привыкнем что есть сущность, которая все решения принимает лучше нас. Да и мы люди не всегда поступаем логично.

Можно представить, что через 10 лет ИИ предложит какому-нибудь банкиру стать пекарем, потому что на основе его «лайков» машина видит, что он хочет сменить профессию, а в его режиме дня – привычку вставать рано, в его хобби – удовольствие от работы руками. При этом она знает, что пекарь нужен в булочной рядом со школой, где учатся дети банкира и судя по переписке он хотел бы видеть их почаще. Он упадёт в зарплате, да, не беда, ведь скоро он получит большое наследство, судя по медицинской карте одного из

родственников и выписки из его завещания. Какой человеческий мозг смог бы проанализировать всё это и дать совет лучше? Мы не всегда будем понимать логику искусственного интеллекта, но будем к ней прислушиваться.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. AI в 2023 году: как развивается искусственный интеллект [Электронный ресурс]. – URL: https://netology.ru/blog/03-2023-ai-trends.

2. Как искусственный интеллект изменит нашу жизнь через 30–50 лет. [Электронный ресурс]. – URL: https://trends.rbc.ru/trends/futurology/64ae65039a79475415aed9b1.

3. Пейзажи и портреты, созданные искусственным интеллектом [Электронный ресурс]. – URL: https://photar.ru/pejzazhi-i-portretysozdannye-iskusstvennym-intellektom/.

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ПРИКЛАДНЫХ ИССЛЕДОВАНИЯХ

Межвузовский сборник научных трудов

Компьютерная верстка: А.Н. Сапрыкин, М.С. Кошелева
Изображение, используемое на обложке сборника,
взято с сайта freepik.com

Подписано в печать 02.05.24. Формат бумаги 60x84 1/16.
Бумага офсетная. Печать струйная. Усл. печ. л. 21.
Тираж 100 экз. Заказ № 7117.

Издательство ИП Коняхин А.В. (Book Jet)

Отпечатано в типографии Book Jet
390005, г. Рязань, ул. Пушкина, д. 18
Сайт: http://bookjet.ru
Почта: info@ bookjet.ru
тел.: +7 (4912) 466-151

ISBN 978-5-907811-32-4