МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский ядерный университет «МИФИ»
Обнинский институт атомной энергетики —
филиал федерального государственного автономного образовательного
учреждения высшего профессионального образования «Национальный
исследовательский ядерный университет «МИФИ»
(ИАТЭ НИЯУ МИФИ)

Отделение интеллектуальных кибернетических систем

УДК 004.896

ДОПУЩЕНА К ЗАЩИТЕ
И.о. руководителя ОИКС
д.ф.-м.н., профессор
. . . . . С.О. Старков

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К НАУЧНО-КВАЛИФИКАЦИОННОЙ РАБОТЕ АСПИРАНТА

МОДЕЛИРОВАНИЕ И ОПТИМИЗАЦИЯ ПРОЦЕССОВ ПЕРЕЗАГРУЗКИ РЕАКТОРОВ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Аспирант . . . . . Белявцев И.П.

Руководитель
И.о. руководителя ОИКС . . . . . Старков С.О.
д.ф.-м.н., профессор

Обнинск 2018

Реферат

74 стр., 31 рис., 24 ист.

ЯДЕРНЫЙ РЕАКТОР, ПЕРЕГРУЗКА, ТА-МОДЕЛЬ, АВТОМАТ, ГРАФ, АЛГОРИТМЫ ПОИСКА, ИСКУССТВЕННАЯ НЕЙРОННАЯ СЕТЬ, ТОПОЛОГИИ, АЛГОРИТМЫ ОБУЧЕНИЯ, PYTHON, REST, ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ

Объектом исследования являются процессы перегрузки топлива в ядерных энергетических реакторах.

Цель работы — создание метода моделирования и оптимизации процесса перегрузки топлива в ядерных реакторах с использованием методов искусственного интеллекта.

В данной работе исследуется строение ядерного реактора и процесс перегрузки топлива. Предлагается формальная информационная модель реактора на основе представления реактора как композиции автоматов (ТА-модель). Исследуются алгоритмы поиска в графах и искусственные нейронные сети прямого распространения. Проводится работа по созданию программного комплекса для оптимизации процесса перегрузки реактора, который использует представление ТА-модели, поиск по графу переходов ТА-моделей и искусственные нейронные сети для оценок физических параметров состояния ТА-модели.

Программный комплекс написан на языке программирования Python по архитектуре REST сервисов. Возможна установка программного комплекса на облачные решения типа PaaS и IaaS.

Содержание

Введение

На сегодняшний день ядерная энергия является одной из самых перспективных и самых опасных видов энергии. Ядерная энергия, несмотря на огромный объем капитальных затрат на создание и утилизацию атомной электростанции (АЭС), является довольно дешевой энергией, так как из малого объема ядерного топлива получается огромное количество энергии. Для выработки энергии 1 МВт·сут требуется всего лишь $1,2$ г делящегося изотопа (урана-235). При сравнении энергетических эквивалентов органического и ядерного топлива обнаруживается, что несколько граммов делящегося изотопа урана-235 примерно равны 1 т нефти (точнее 4 г $^{235}U \approx 1$ т нефти)!

Другая, не менее важная причина состоит в том, что имеющихся ресурсов органического топлива при нынешних тенденциях энергопотребления хватит, по различным прогнозам, на 50-200 лет, а ресурсов урана (как урана-235, так и урана-238 с учетом построения замкнутого топливного цикла) хватит примерно на 10 000 лет. Кроме запасов урана, на Земле имеются еще запасы тория, объем которых, по оценкам, сопоставим с запасами урана, а возможно, в несколько раз их превосходит. [1]

Несмотря на все эти неоспоримые преимущества атомная энергетика таит в себе ряд опаснейших проблем. Примером, таких проблем может служить опасность теплового взрыва реактора, необходимость захоронения ядерных отходов, усталость капитальных конструкций реактора. При этом если задачу предотвращения опасности теплового взрыва можно считать во многих аспектах решенной, то задача уменьшения усталости капитальных конструкций реактора является актуальной по сей день. На сегодняшний день ядерная энергия является одной из самых перспективных и самых опасных видов энергии. Ядерная энергия, несмотря на огромный объем капитальных затрат на создание и утилизацию атомной электростанции (АЭС), является довольно дешевой энергией, так как из малого объема ядерного топлива получается огромное количество энергии. Для выработки энергии 1 МВт·сут требуется всего лишь $1,2$ г делящегося изотопа (урана-235). При сравнении энергетических эквивалентов органического и ядерного топлива обнаруживается, что несколько граммов делящегося изотопа урана-235 примерно равны 1 т нефти (точнее 4 г $^{235}U \approx 1$ т нефти)!

Другая, не менее важная причина состоит в том, что имеющихся ресурсов органического топлива при нынешних тенденциях энергопотребления хватит, по различным прогнозам, на 50-200 лет, а ресурсов урана (как урана-235, так и урана-238 с учетом построения замкнутого топливного цикла) хватит примерно на 10 000 лет. Кроме запасов урана, на Земле имеются еще запасы тория, объем которых, по оценкам, сопоставим с запасами урана, а возможно, в несколько раз их превосходит. [1]

Несмотря на все эти неоспоримые преимущества атомная энергетика таит в себе ряд опаснейших проблем. Примером, таких проблем может служить опасность теплового взрыва реактора, необходимость захоронения ядерных отходов, усталость капитальных конструкций реактора. При этом если задачу предотвращения опасности теплового взрыва можно считать во многих аспектах решенной, то задача уменьшения усталости капитальных конструкций реактора является актуальной по сей день.

Наибольшие воздействия на капитальные конструкции происходят во время планово-профилактических ремонтов (ППР) реактора. На сегодняшний день отсутствуют методологии по определению наиболее оптимального плана ППР с точки зрения воздействия на капитальные конструкции, а так же по времени проведения ППР.

Целью данной работы является разработка методологии моделирования и построения оптимального порядка планово-профилактического ремонта реакторов с использованием методов искусственного интеллекта.

Для достижения поставленной цели необходимо было решить следующие задачи:

1. Разработать формальную информационную модель, описывающую процесс перезагрузки реактора
2. Исследовать метод построения оптимального порядка процесса перезагрузки реактора
3. Разработать способ адаптивного моделирования физических параметров активной зоны реактора
4. Разработать программный комплекс, реализующий моделирование и построение оптимального порядка ППР реакторов

Основные положения, выносимые на защиту:

1. Разработана формальная информационная модель, описывающая процесс перезагрузки реактора, на основе аппарата теории автоматов
2. Предложен метод оптимизации порядка ППР с использованием графа переходов и алгоритма A*
3. Разработан способ адаптивного моделирования физических параметров активной зоны реактора с использованием искусственных нейронных сетей прямого распространения.
4. Реализован программный комплекс, реализующий моделирование и построение оптимального порядка ППР реакторов

Научная новизна:

1. Было выполнено оригинальное исследование процесса ППР различных реакторов
2. Впервые была предложена и опробована формальная информационная модель ППР на основе аппарата теории графов
3. Впервые был предложен и опробован адаптивный способ моделирования физических параметров активной зоны реактора с использованием искусственных нейронных сетей

Научная и практическая значимость. Результаты настоящей работы могут быть применены для моделирования и построения оптимального порядка планово-профилактического ремонта реакторов.

Апробация работы. Основные результаты работы докладывались на конференции «Применение кибернетических методов в решении проблем общества XXI века» (Обнинск, 2014).

Публикации. Основные результаты по теме диссертации изложены в тезисах докладов [2].

Объем и структура работы. Диссертация состоит из введения, пяти глав, заключения и двух приложений. Полный объем диссертации составляет 74 страницы с 31 рисунком и 13 таблицами. Список литературы содержит 24 наименования.

Глава 1

Информационная модель реактора

В данной главе будут рассмотрены устройство и особенности осуществления процесса перегрузки реактора на примере реактора БН-350. Будет исследованы общие положения теории автоматов, а также будет предложена информационная модель реактора в терминах теории автоматов.

1.1 Описательная информационная модель реактора

1.1.1 Общие сведения об устройстве реактора класса БН-350

Рассмотрим устройство энергетического реактора на быстрых нейтронах на примере установки БН-350. Реактор БН-350 — первый реактор на быстрых нейтронах энергетического назначения (рис. 1.1). В его конструкцию были заложены многие технические решения, выработанные на реакторе БОР-60, и впоследствии реализованные в конструкциях реакторов БН-600 и БН-800. Внутри герметичного корпуса реактора, выполненного в виде бака переменного диаметра, расположены тепловыделяющие сборки (ТВС) активной зоны и боковой зоны воспроизводства, хранилище отработавших ТВС, отражатель нейтронов, механизмы системы перегрузки, стержни системы управления и защиты (СУЗ). Натрий под давлением 1 МПа подводится от главных циркуляционных насосов (ГЦН) первого контура по шести напорным трубопроводам диаметром 500 мм в нижнюю часть корпуса реактора, образующую напорную камеру. Сверху на камере закреплен напорный коллектор с установленными в нем ТВС. В коллекторе поток теплоносителя распределяется по ТВС в соответствии с уровнем энерговыделения в них. Он представляет собой прочную силовую конструкцию, воспринимающую весовую нагрузку от ТВС и других выемных частей реактора, а также внутренние давление теплоносителя. Крепление коллектора к напорной камере допускает его извлечение и замену в случае необходимости.

Пройдя ТВС, нагретый натрий попадает в выходную смесительную камеру, откуда по шести сливным трубопроводам диаметром 600 мм отводится к промежуточным теплообменникам.

Активная зона набирается из шестигранных ТВС размером «под ключ» 96 мм. Каждая сборка содержит пучок гладкостержневых тепловыделяющих элементов (ТВЭЛов) диаметром 6,9 мм, расположенных с шагом 8 мм. Топливо — обогащенная двуокись урана. Дистанцирование ТВЭЛов в пучке осуществляется проволокой, спирально навитой на оболочку ТВЭЛа. Верхняя торцевая зона воспроизводства набирается из элементов увеличенного диаметра (12 мм) с двуокисью обедненного урана, установленных внутри сборки над пучком ТВЭЛов. Гексагональная чехловая труба сборки несет давление теплоносителя, которое в нижней части значительно превышает давление в межкассетном пространстве реактора. К чехлу сборки с обоих торцов приварены концевые детали ци-

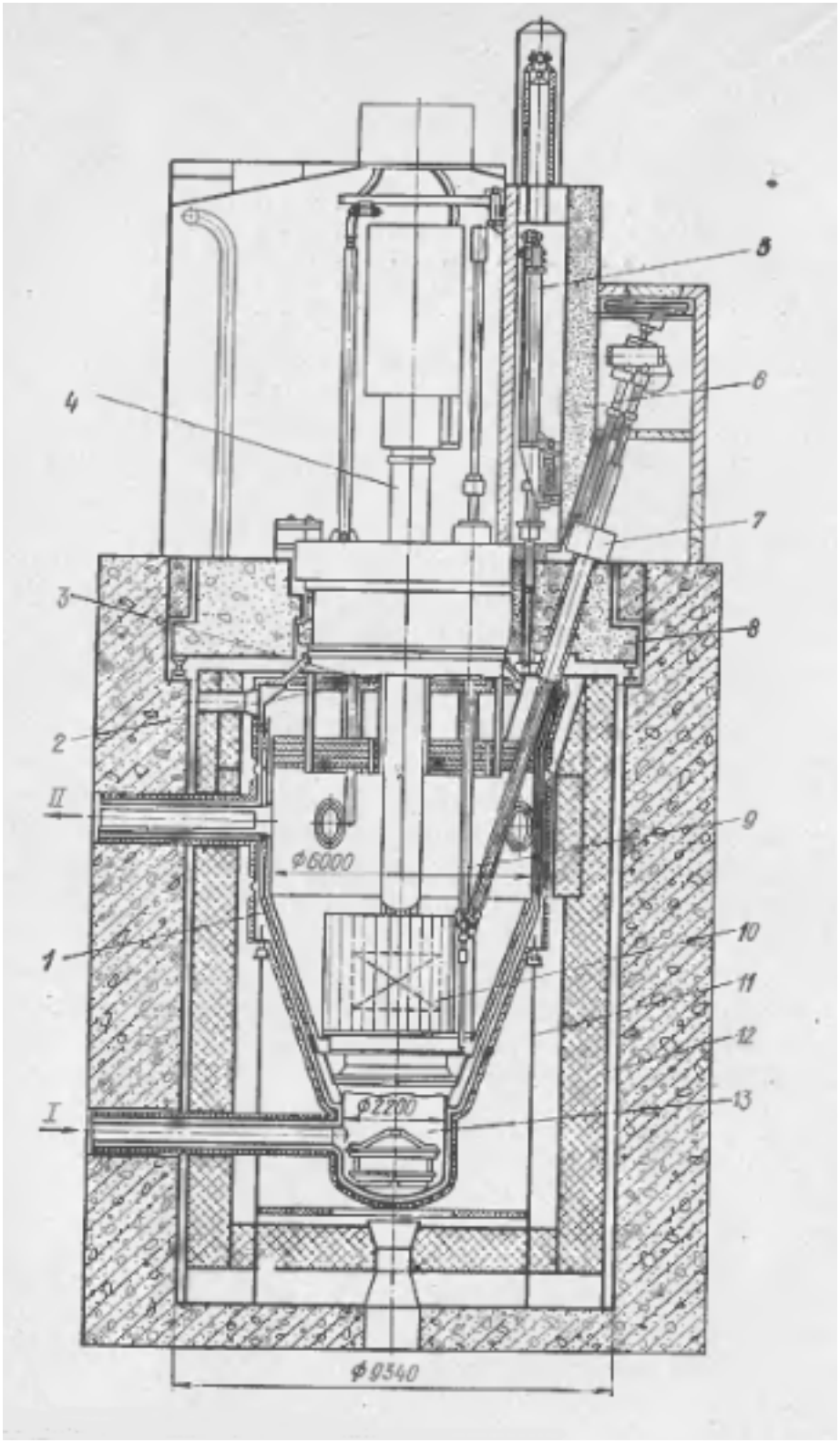

Рисунок 1.1: Общий вид реактора БН-350: 1 – корпус реактора; 2 – большая поворотная пробка; 3 – малая поворотная пробка; 4 – центральная поворотная колонка с механизмом СУЗ; 5 – механизм передачи сборок; 6 – передаточный бокс; 7 – элеватор загрузки-выгрузки; 8 – верхняя неподвижная защита; 9 – механизм перегрузки; 10 – активная зона; 11 – опора реактора; 12 – боковая защита; 13 – напорная камера; I – вход натрия; II – выход натрия

линдрической формы: нижний хвостовик с боковыми отверстиями, через которые в ТВС поступает теплоноситель, и фигурная головка с выходными окнами, предназначенная для захвата сборки механизмом перезагрузки. Для уменьшения паразитных протечек натрия из напорного коллектора вдоль хвостовика сборки на нем предусмотрены лабиринтные уплотнения. Зазор между ТВС принят минимальным (2 мм) исходя из условий нормального извлечения их при перезагрузке с учетом возможных при работе формоизменений. Для пространственной фиксации сборок они дистанцируются с помощью выступов («платиков») в верхней части чехловых труб.

Часть гнезд в центральной области активной зоны занята органами СУЗ (рис. 1.2) Бо-

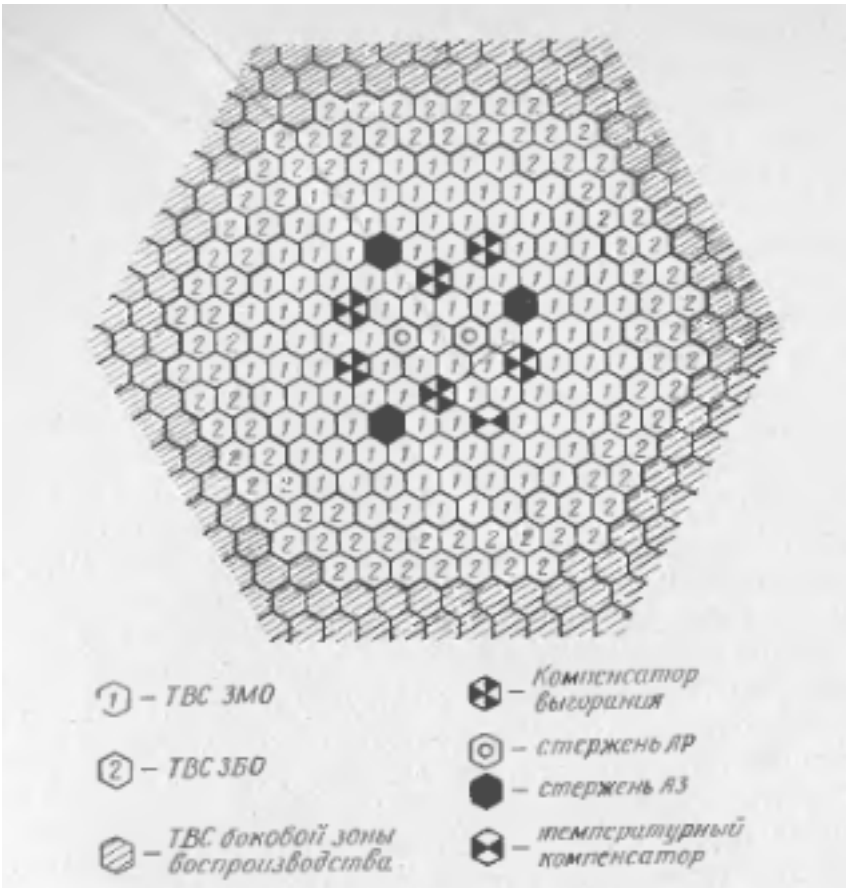

Рисунок 1.2: Сетка СУЗ в реакторе БН-350: ЗМО – зона малого обогащения, ЗБО – зона большого обогащения

ковая зона воспроизводства общей толщиной 600 мм набрана из нескольких рядов ТВС, каждая из которых состоит из элементов диаметром 14,2 мм с двуокисью обедненного урана. За внешним рядом сборок боковой зоны воспроизводства размещаются гнезда внутриреакторного хранилища, где отработавшие ТВС выдерживаются в течение интервала между перезагрузками реактора (2 месяца). Необходимость выдержки сборок активной зоны перед выгрузкой из реактора обусловлена высоким уровнем остаточных тепловыделений в них. В хранилище ТВС расхолаживаются теплоносителем, поступающим из напорного коллектора реактора. После выдержки сборок значительно упрощается обращение с ними в тракте перегрузки. Сборки боковой зоны воспроизводства имеют относительно низкое остаточное тепловыделение, поэтому предварительного охлаждения их перед выгрузкой не требуется. Непосредственно за хранилищем устанавливается отражатель нейтронов из нескольких рядов стальных болванок с внешней конфигурацией ТВС (общая толщина 200 мм).

Натрий, заполняющий корпус реактора, образует свободный уровень, над которым находится аргоновая подушка. Газовый объем служит для компенсации температурных расширений теплоносителя. Вместе с тем он изолирует верхнюю часть реактора от горячего теплоносителя, выходящего из активной зоны, и защищает натрий от контакта с воздухом при случайной разгерметизации реактора. Высота уровня над головками ТВС выбрана таким образом, чтобы транспортировка выгружаемых сборок проходила под слоем натрия, чем обеспечивается их надежное и эффективное охлаждение. [3]

1.1.2 Описание процесса планово-профилактического ремонта

Рассмотрим процесс планово-профилактического ремонта для реактора БН-350.

Характерной особенностью конструкции реактора является то, что он не имеет съемной герметизирующей крышки, а перегрузка ТВС осуществляется без общего вскрытия корпуса герметичными дистанционно управляемыми механизмами под защитой инертного газа по закрытому тракту от реактора до внешнего хранилища. Такое решение продиктовано специфическим требованием натриевой технологии — недопустимостью контакта натрия с воздухом. Сверху корпус реактора закрыт двумя многослойными плитами («пробками») верхней радиационной защиты. Защитные пробки являются одновременно частью системы перегрузки реактора. С их помощью осуществляются наведение внутриреакторного механизма перегрузки (ВМП) на ТВС, подлежащие перегрузке и перенос сборок внутри реактора. Эти операции выполняются совместным вращением обоих пробок — большой, перекрывающей горловину реактора, и расположенной в ней эксцентрично малой пробки, в которую вмонтирован ВМП. Обе пробки установлены на шаровых опорах, имеют по периферии зубчатый венец и электромеханические приводы, обеспечивающие их вращение по командам автоматизированной системы управления. Поворотные пробки имеют значительный диаметр (большая пробка – 4300 мм), поэтому создание надежного механического уплотнения по всему периметру, исключающего выход из реактора радиоактивного защитного газа, является весьма сложной технической задачей. Благодаря небольшой разнице давлений между газовой полостью реактора и окружающей средой оказалось возможным выполнить уплотнение пробок в виде гидрозатвора: каждая пробка имеет цилиндрическую юбку, которая опущена в кольцевую ванну, заполненную тяжелой уплотняющей жидкостью, герметично связанную с корпусом реактора. Уплотняющей средой в гидрозатворе служит сплав олова и висмута (57% Sn и 43% Bi), имеющий плотность $8,3 \cdot 10^3$ кг/м^3 и температуру плавления 138°C. При работе реактора сплав находится в твердом состоянии и вращение пробок невозможно. Перед перегрузкой его расплавляют с помощью электронагревателей, вмонтированных в корпус гидрозатвора.

Перегрузка ТВС осуществляется на остановленном реакторе комплексом взаимодействующих механизмов в режиме автоматического управления. Операции с ТВС внутри реактора (извлечение, перенос, установка) выполняются ВМП. Он представляет собой прямую телескопическую штангу с захватным устройством и несколькими электромеханическими приводами. После сцепления захвата с головкой перегружаемой сборки последняя втягивается внутрь направляющей трубы ВМП и в таком положении переносится вращением пробок в заданное место. Перенос ТВС в направляющей трубе предохраняет сборку от раскачивания и случайных механических повреждений. Выгрузка отработавших ТВС из реактора и загрузка свежих в реактор осуществляются через специальный перегрузочный патрубок небольшого диаметра в верхней части корпуса реактора с помощью двух механизмов: элеватора и механизма передачи сборок (МПС). Элеватор представляет собой наклонный подъемник с цепным приводом и подвижной кареткой, в которую ВМП устанавливает перегружаемую сборку. Движением каретки по наклонной направляющей сборка перемещается вверх под перегрузочный патрубок, в котором установлен МПС. В

точке встречи с механизмом передачи головка сборки выходит из-под уровня натрия в газовую полость реактора. Это исключает погружение в натрий захвата МПС и повышает его надежность. МПС втягивает сборку в герметичный передаточный бокс, установленный над перегрузочным патрубком, и транспортирует ее во внешние хранилище отработавших сборок. Двигаясь в обратном направлении, МПС захватывает свежую ТВС, переносит ее к патрубку и устанавливает в каретку элеватора, которая опускает сборку в нижнее положение. Отсюда она переносится с помощью ВМП в свободное гнездо активной зоны или боковой зоны воспроизводства. С учетом совмещения во времени операций, выполняемых различными механизмами перегрузочного комплекса, полный цикл перегрузки одного гнезда активной зоны (от извлечения отработавшей сборки до установки свежей) занимает около 50 минут. [3]

1.2 Формальное информационная модель реактора

1.2.1 Общие положения теории автоматов

Приведем общие понятия и положения теории автоматов, необходимые для построения формальной информационной модели реактора.

Определение 1.1. Алфавитом Σ называют некоторое непустое конечное множество символов.

Определение 1.2. Словом называют некоторую конечную (возможно, пустую) последовательность символов алфавита: $\omega = \sigma_1\sigma_2\ldots\sigma_l$. Количество символов в слове (число l) называют длиной слова. Пустое слово принято обозначать ε.

Определение 1.3. Детерминированным конечным автоматом (ДКА) называется пятерка (кортеж) $A = \langle Q, \Sigma, \delta, q_0, F\rangle$, где:

- Q — конечное множество состояний;
- Σ — алфавит;
- $\delta : Q \times \Sigma \to Q$ — функция переходов;
- $q_0 \in Q$ — начальное состояние;
- $F \subseteq Q$ — множество терминальных (конечных) состояний.

Обработка слова $\omega = \sigma_1\sigma_2\ldots\sigma_l$ ДКА A происходит следующим образом. Сначала автомат A находится в стартовом состоянии q_0. Затем на каждом шаге обработки автомат считывает очередной символ $sigma_i$ слова ω и переходит из своего текущего состояния q в состояние $\delta(q;\omega_i)$. К моменту, когда все символы входного слова ω обработаны, автомат находится в некотором состоянии p. Говорят, что автомат A принимает слово ω, если $p \in F$, и не принимает в обратном случае. [4]

Определение ДКА как пятерки объектов с детальным описанием функций переходов слишком сухое и неудобочитаемое. Существует два более удобных способа описания автоматов.

1. Диаграмма переходов, которая представляет собой граф.
2. Таблица переходов, дающая табличное представление функции δ. Из нее очевидны состояния и входной алфавит.

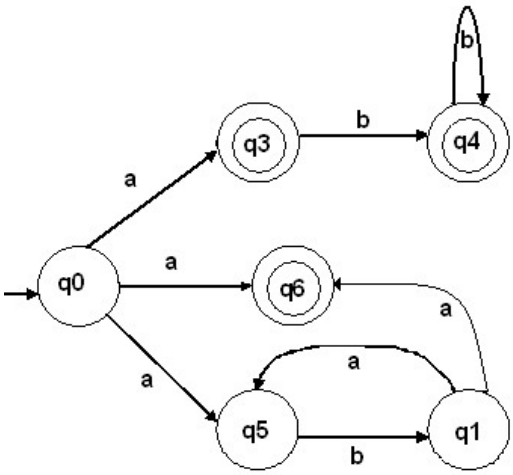

Рисунок 1.3: Пример диаграммы перехода конечного автомата

Диаграмма переходов (см. рис. 1.3) для ДКА вида $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ есть граф, определяемый следующим образом:

а) всякому состоянию из Q соответствует некоторая вершина;

б) пусть $\delta(q, a) = p$ для некоторого состояния q из Q и входного символа a из Σ. Тогда диаграмма переходов должна содержать дугу из вершины q в вершину p, отмеченную a. Если существует несколько входных символов, переводящих автомат A из состояния q в состояние p, то диаграмма переходов может содержать одну дугу, отмеченную списком этих символов;

в) диаграмма содержит стрелку в начальное состояние, отмеченную как Начало. Эта стрелка не выходит не из какого состояния;

г) вершины, соответствующие допускающим состояниям (состояниям из F), отмечаются двойным кружком. Состояния, не принадлежащие F, изображаются простым (одинарным) кружком.

Таблица переходов представляет собой обычное табличное представление функции, подобной δ, которая двум значениям ставит в соответствие одно значение. Строки таблицы соответствуют состояниям, а столбцы — входным символам. На пересечении строки, соответствующей состоянию q, и столбца, соответствующего символу a, находится состояние $\delta(q, a)$. [5]

Определение 1.4. Автомат называется полностью определенным или полным, если $D_\delta = Q \times \Sigma$. Иными словами, для любого начального состояния и любой входной последовательности (в пределах алфавита Σ) однозначно определена выходная последовательность.

Определение 1.5. Автомат называется частично определенным или частичным, если функция δ определена не для всех пар $(q, a) \in Q \times \Sigma$.

В табличном задании таких автоматов на месте неопределенных состояний и выходных сигналов ставится прочерк. На графе автомата неопределенному состоянию можно поставить в соответствие дополнительную вершину.

Неопределенное состояние автомата называется граничным. Дальнейшее поведение автомата, попавшего в граничное состояние, не определяется. Выходная последовательность, содержащая неопределенные сигналы, называется частичной. Заметим, что модель частичного автомата является более абстрактной, чем модель полного автомата: она задает целое множество дискретных устройств. [6]

Под композицией элементарных автоматов в общем случае понимается следующее. Пусть заданы элементарные автоматы $A_1, A_2, \ldots, A_k$. Произведем объединение элементарных автоматов в систему совместно работающих автоматов. Введем в рассмотрение некоторое конечное множество узлов, называемых внешними выходными узлами. Эти узлы отличаются от узлов рассматриваемых элементарных автоматов, которые носят название внутренних. Композиция автомата состоит в том, что в полученной системе элементарных автоматов $A_1, A_2, \ldots, A_k$ и внешних узлов производится отождествление некоторых узлов (как внешних так и внутренних).

С точки зрения совместной работы системы автоматов смысл операции отождествления узлов состоит в том, что элементарный сигнал, попадающий на один из узлов, входящих в множество отождествляемых между собой узлов, попадает тем самым на все узлы этого множества. После проведенных отождествлений узлов система автоматов превращается в так называемую схему (сеть) автоматов. Будем считать, что автоматы, входящие в систему автоматов, работают совместно, если в каждый момент автоматного времени на все внешние входные узлы подается набор входных сигналов (структурный входной сигнал схемы) и со всех внешних выходных узлов снимается набор выходных сигналов (структурный выходной сигнал). Следует заметить, что при построении схемы автоматов должны выполняться условия корректности. [7]

1.2.2 Описание реактора в терминах теории автоматов

Исходя из положений, представленных в разделах 1.1 и 1.2.1, представим реактор как композицию конечных детерминированных автоматов.

Пусть реактор состоит из k структурных элементов, каждый из которых можно представить как конечный детерминированный автомат. Таким образом $i-$й структурный элемент реактора описывается кортежем $A_i = \langle Q^i, \Sigma^i, \delta^i, q_0^i, F^i \rangle$.

Тогда реактор можно представить следующим кортежем $A = \langle Q, \Sigma, \delta, q_0, F \rangle$, где

- $Q = \cup_{i=0}^{n} Q^i$;
- $\Sigma = \cup_{i=0}^{n} \Sigma^i$;
- $\delta = f(q, a)$;
- $q_0 = \cup_{i=0}^{n} q_0^i$;
- $F = \cap_{i=0}^{n} F^i$.

Рассмотрим данную модель на примере фиктивного реактора. Пусть реактор имеет следующую конфигурацию активной зоны (см. рис. 1.4). Ячейки А1, Б1, В1, Г1, Д1, Е1 предназначены для организации внутриреакторного хранилища отработавших ТВС, ячейки А2, Б2, В2, Г2, Д2, Е2 содержат сборки для организации боковой зоны воспроизводства, ячейки А3, Б3, В3, Г3, Д3, Е3 содержат сборки активной зоны реактора и ячейка А4 содержит механизм автоматического регулирования реактора.

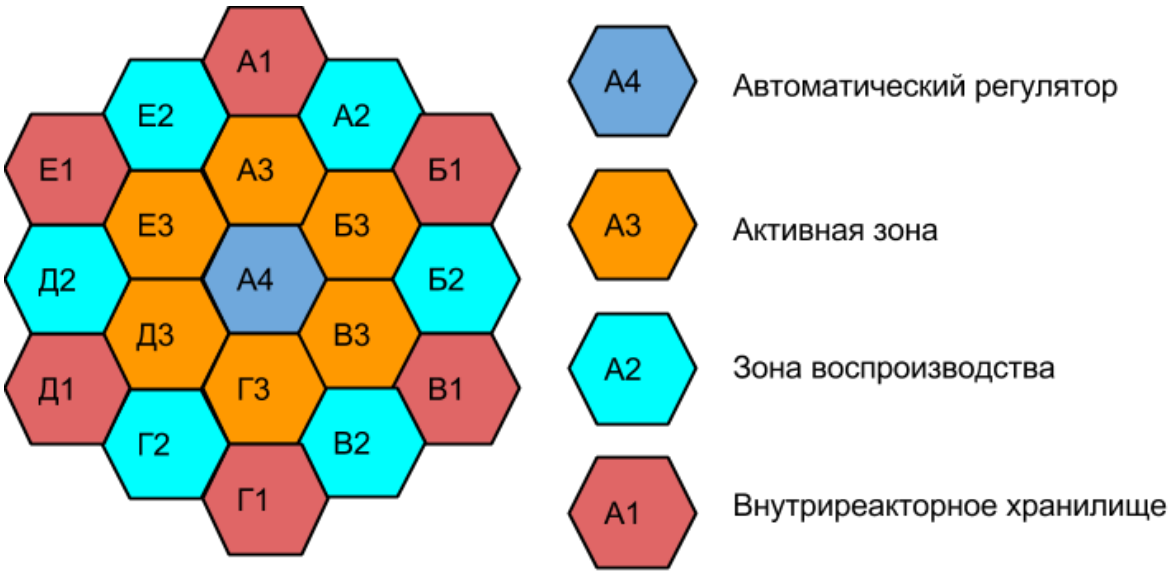

Рисунок 1.4: Конфигурация активной зоны фиктивного реактора: А1 – внутриреакторное хранилище, А2 – зона воспроизводства, А3 — активная зона, А4 — автоматический регулятор

Фиктивный реактор имеет механизм перегрузки, аналогичный описанному в 1.1.2. Большая поворотная пробка имеет 6 положений, которые соответствуют букве в названии ячейки, малая поворотная пробка - 4 положения, которые соответствуют цифре в названии ячейки.

Перед началом планово-профилактического ремонта поворотные пробки находятся в положении А1, активная зона и зоны воспроизводства полностью заполнены, внутриреакторное хранилище пусто, частично или полностью заполнено. Некоторые ТВС активной зоны и зоны воспроизводства, а также все ТВС из внутриреакторного хранилищ помечены как требующие перезагрузки. После планово-профилактического ремонта поворотные пробки находятся в положении А1, активная зона и зоны воспроизводства полностью заполнены, внутриреакторное хранилище пусто, частично или полностью заполнено. Все ТВС активной зоны и зоны воспроизводства, а также все ТВС из внутриреакторного хранилищ помечены как не требующие перезагрузки.

Разделим реактор на следующие структурные компоненты: тепловыделяющие сборки с обогащенным ураном, тепловыделяющие сборки с обедненным ураном, гнезда внутриреакторного хранилища, гнезда зоны воспроизводства, гнезда активной зоны, большая поворотная пробка, малая поворотная пробка, перегрузочный механизм. Рассмотрим отдельные структурные компоненты реактора как автоматы.

Тепловыделяющие сборки с обогащенным ураном:

- $Q = \{$В активную зону, Во внутриреакторное хранилище, Во внешнее хранилище, Обслуживание завершено$\}$;
- $\Sigma = \{$Обслужить$\}$;
- $\delta = f(q, a)$, где f соответствует таблице 1.1;
- $q_0 \in \{$В активную зону, Во внутриреакторное хранилище, Во внешнее хранилище, Обслуживание завершено$\}$;
- $F = \{$Обслуживание завершено$\}$.

Таблица 1.1: Таблица переходов автомата «Тепловыделяющая сборка с обогащенным ураном»

$q_i \diagdown \sigma_j$	Обслужить
В активную зону	Обслуживание завершено
Во внутриреакторное хранилище	Обслуживание завершено
Во внешнее хранилище	Обслуживание завершено
Обслуживание завершено	—

Тепловыделяющие сборки с обедненным ураном:

- $Q = \{$В зону воспроизводства, Во внешнее хранилище, Обслуживание завершено$\}$;
- $\Sigma = \{$Обслужить$\}$;
- $\delta = f(q, a)$, где f соответствует таблице 1.2;
- $q_0 \in \{$В зону воспроизводства, Во внешнее хранилище, Обслуживание завершено$\}$;
- $F = \{$Обслуживание завершено$\}$.

Таблица 1.2: Таблица переходов автомата «Тепловыделяющая сборка с обедненным ураном»

$q_i \diagdown \sigma_j$	Обслужить
В зону воспроизводства	Обслуживание завершено
Во внешнее хранилище	Обслуживание завершено
Обслуживание завершено	—

Гнезда внутриреакторного хранилища:

- $Q = A_{ef} \cup \varnothing$, где A_{ef} – множество автоматов «Тепловыделяющая сборка с обогащенным ураном»;
- $\Sigma = A_{ef} \cup \varnothing$, где A_{ef} – множество автоматов «Тепловыделяющая сборка с обогащенным ураном»;
- $\delta = f(q, a)$, где f соответствует таблице 1.3;
- $q_0 \in Q$;
- $F \in Q$.

Гнезда зоны воспроизводства:

- $Q = A_{uf} \cup \varnothing$, где A_{uf} – множество автоматов «Тепловыделяющая сборка с обедненным ураном»;
- $\Sigma = A_{uf} \cup \varnothing$, где A_{uf} – множество автоматов «Тепловыделяющая сборка с обедненным ураном»;
- $\delta = f(q, a)$, где f соответствует таблице 1.4;

Таблица 1.3: Таблица переходов автомата «Гнездо внутриреакторного хранилища»

$q_i \diagdown \sigma_j$	a_{ef}	$\varnothing$
a_{ef}	—	$\varnothing$
$\varnothing$	a_{ef}	—

- $q_0 \in A_{uf}$;
- $F \in A_{uf}$.

Таблица 1.4: Таблица переходов автомата «Гнездо зоны воспроизводства»

$q_i \diagdown \sigma_j$	a_{uf}	$\varnothing$
a_{uf}	—	$\varnothing$
$\varnothing$	a_{uf}	—

Гнезда активной зоны:

- $Q = A_{ef} \cup \varnothing$, где A_{ef} – множество автоматов «Тепловыделяющая сборка с обогащенным ураном»;
- $\Sigma = A_{ef} \cup \varnothing$, где A_{ef} – множество автоматов «Тепловыделяющая сборка с обогащенным ураном»;
- $\delta = f(q, a)$, где f соответствует таблице 1.5;
- $q_0 \in A_{ef}$;
- $F \in A_{ef}$.

Таблица 1.5: Таблица переходов автомата «Гнездо активной зоны»

$q_i \diagdown \sigma_j$	a_{ef}	$\varnothing$
a_{ef}	—	$\varnothing$
$\varnothing$	a_{ef}	—

Большая поворотная пробка:

- $Q = \{\text{А}, \text{Б}, \text{В}, \text{Г}, \text{Д}, \text{Е}\}$;
- $\Sigma = \{\text{влево}, \text{вправо}\}$;
- $\delta = f(q, a)$, где f соответствует таблице 1.6;
- $q_0 = \text{А}$;
- $F = \text{А}$.

Таблица 1.6: Таблица переходов автомата «Большая поворотная пробка»

$q_i \diagdown \sigma_j$	влево	вправо
А	Е	Б
Б	А	В
В	Б	Г
Г	В	Д
Д	Г	Е
Е	Д	А

Малая поворотная пробка:

- $Q = \{1, 2, 3, 4\}$;
- $\Sigma = \{\langle$влево$, q_{bp}\rangle, \langle$вправо$, q_{bp}\rangle\}$, где q_{bp} – текущее состояние автомата «Большая поворотная пробка»;
- $\delta = f(q, a)$, где f соответствует таблице 1.7;
- $q_0 = 1$;
- $F = 1$.

Таблица 1.7: Таблица переходов автомата «Малая поворотная пробка»

$q_i \diagdown \sigma_j$	$\langle$влево, А$\rangle$	$\langle$вправо, А$\rangle$	$\langle$влево, $\overline{\text{А}}\rangle$	$\langle$вправо, $\overline{\text{А}}\rangle$
1	4	2	3	2
2	1	3	1	3
3	2	4	2	1
4	3	1	-	-

Перегрузочный механизм: Рассмотрим перегрузочный механизм как совокупность следующих автоматов: большая поворотная пробка, малая поворотная пробка, определитель типа гнезда, селектор гнезда, перегрузочный механизм.

Определитель типа гнезда:

- $Q = \{$Внутриреакторное хранилище, Зона воспроизводства, Активная зона, Автоматический регулятор$\}$;
- $\Sigma = \{1, 2, 3, 4\}$;
- $\delta = f(q, a)$, где f соответствует таблице 1.8;
- $q_0 =$ Внутриреакторное хранилище;
- $F =$ Внутриреакторное хранилище.

Селектор гнезда:

- $Q = Q_{\text{гнезда}}$;

Таблица 1.8: Таблица переходов автомата «Определитель типа гнезда»: ВХ – Внутриреакторное хранилище; ЗВ – Зона воспроизводства; АЗ – Активная зона; АР – Автоматический регулятор.

$q_i \diagdown \sigma_j$	1	2	3	4
ВХ	ВХ	ЗВ	АЗ	АР
ЗВ	ВХ	ЗВ	АЗ	АР
АЗ	ВХ	ЗВ	АЗ	АР
АР	ВХ	ЗВ	АЗ	АР

- $\Sigma = \langle Q_{\text{А1}}, \ldots, Q_{\text{Е3}}, Q_{bp}, Q_{lp} \rangle$;
- $\delta = f(q, a)$, где f соответствует таблице 1.9;
- $q_0 = Q_{\text{А1}}$;
- $F = Q_{\text{А1}}$.

Таблица 1.9: Таблица переходов автомата «Селектор гнезда».

$q_i \diagdown \sigma_j$	$\langle q_{\text{А1}}, \ldots, q_{\text{Е3}}, \text{А}, 1 \rangle$	$\cdots$	$\langle q_{\text{А1}}, \ldots, q_{\text{Е3}}, \text{Е}, 3 \rangle$
$\forall q \in Q$	$q_{\text{А1}}$	$\cdots$	$q_{\text{Е3}}$

Перегрузочный механизм:

- $Q = A_{\text{ТВС}} \cup \varnothing$;
- $\Sigma = \langle Q_{\text{Селектора}}, \ldots, Q_{\text{Определителя}}, \{\text{извлечь}, \text{установить}, \text{во внешнее хранилище}, \text{из внешнего хранилища}\} \rangle$;
- $\delta = f(q, a)$, где f соответствует таблице 1.10;
- $q_0 = \varnothing$;
- $F = \varnothing$.

Таблица 1.10: Таблица переходов автомата «Перегрузочный механизм»: ВХ – Внутриреакторное хранилище; ВнешХ – Внешнее хранилище; ЗВ – Зона воспроизводства; АЗ – Активная зона; АР – Автоматический регулятор.

$\sigma_j \diagdown q_i$	$\varnothing$	В АЗ	Во ВХ	Во внешХ	В ЗВ
$\langle \varnothing, \text{АЗ}, \text{извлечь} \rangle$	—	—	—	—	—
$\langle \varnothing, \text{АЗ}, \text{установить} \rangle$	—	$\varnothing$	—	—	—
$\langle \varnothing, \text{АЗ}, \text{во внешнее хранилище} \rangle$	—	—	—	—	—
$\langle \varnothing, \text{АЗ}, \text{из внешнего хранилища} \rangle$	$A_{\text{ТВС}}$	—	—	—	—
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\langle \text{В ЗВ}, \text{ВХ}, \text{из внешнего хранилища} \rangle$	—	—	—	—	—

Рассмотрев отдельные автоматы, моделирующие поведение структурных элементов реактора, рассмотрим композицию этих автоматов (ТА-модель). Согласно описанию фиктивного реактора, приведенного выше (см. рис. 1.4), ТА-модель реактора состоит из $6-12$ ТА-моделей тепловыделяющих сборок с обогащенным ураном, 6 ТА-моделей тепловыделяющих сборок с обедненным ураном, 6 ТА-моделей гнезд активной зоны, 6 ТА-моделей гнезд зоны воспроизводства, 6 ТА-моделей гнезд внутриреакторного хранилища, ТА-модели большой поворотной пробки, ТА-модели малой поворотной пробки, ТА-модели селектора гнезд, ТА-модели определителя типа гнезда и ТА-модели перегрузочного механизма.

Выходы ТА-моделей ТВС с обогащенным ураном являются входными символами для ТА-моделей гнезд активной зоны и ТА-моделей гнезд внутриреакторного хранилища. Выходы ТА-моделей ТВС с обедненным ураном являются входными символами для ТА-модели гнезд зоны воспроизводства. Выходы ТА-моделей гнезд реактора являются входными символами ТА-модели селектора гнезд. Выход ТА-модели большой поворотной пробки является входным символом для ТА-моделей малой поворотной пробки, селектора гнезд и определителя типа гнезд. Выход ТА-модели малой поворотной пробки является входным символом для ТА-модели селектора гнезд и определителя типа гнезд. Выходы ТА-моделей селектора гнезд и определителя типа гнезд являются входными символами для ТА-модели перегрузочного механизма. Композиционная схема ТА-модели фиктивного реактора представлена на рисунке 1.5.

Исходя из описанного выше представим модель фиктивного реактора в общем виде следующим образом:

- $Q = \langle Q_{\text{ТВС 1}}, \ldots, Q_{\text{ТВС 18}}, Q_{\text{А1}}, \ldots, Q_{\text{Е3}}, Q_{bp}, Q_{lp}, Q_{\text{перегрузочного механизма}} \rangle$;
- $\Sigma = \{$извлечь, установить, во внешнее хранилище, из внешнего хранилища, БПП влево, БПП вправо, МПП влево, МПП вправо$\}$;
- $\delta = \delta_{\text{ТВС 1}} \cdot \cdots\cdot \delta_{\text{ТВС 18}} \cdot \delta_{\text{А1}} \cdot \cdots\cdot \delta_{\text{Е3}} \cdot \delta_{bp} \cdot \delta_{lp} \cdot \delta_{\text{перегрузочного механизма}}$;
- $q_0 = \langle q_{\text{ТВС 1}}, \ldots, q_{\text{ТВС 18}}, q_{\text{А1}}, \ldots, q_{\text{Е3}}, q_{bp}, q_{lp}, q_{\text{перегрузочного механизма}} \rangle$;
- $F = \langle F_{\text{ТВС 1}}, \ldots, F_{\text{ТВС 18}}, F_{\text{А1}}, \ldots, F_{\text{Е3}}, F_{bp}, F_{lp}, F_{\text{перегрузочного механизма}} \rangle$.

Таким образом, были рассмотрены устройство и особенности осуществления процесса перегрузки реактора на примере реактора БН-350. Будет исследованы общие положения теории автоматов, а также будет предложена информационная модель реактора в терминах теории автоматов (ТА-модель) для фиктивного реактора.

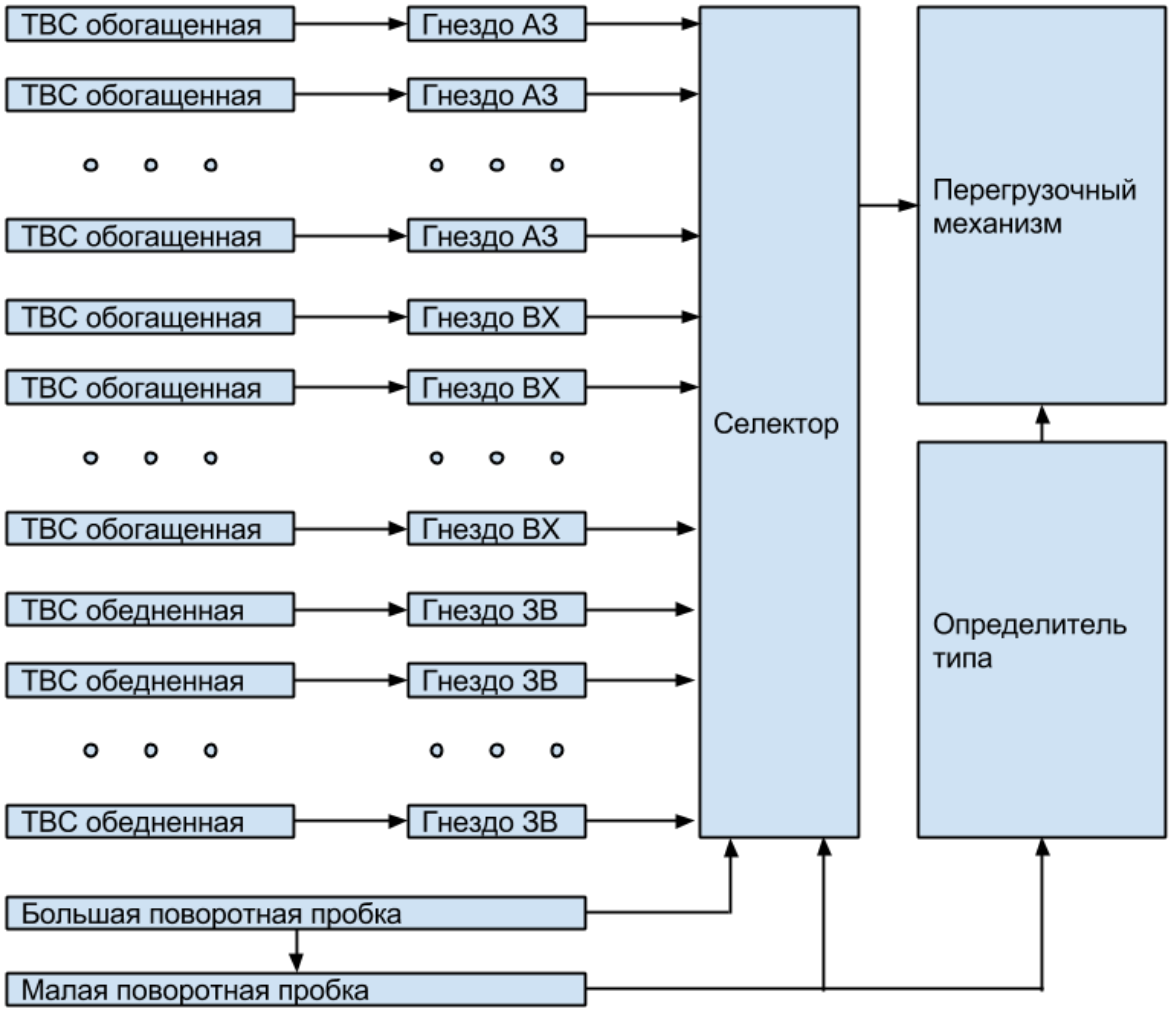

Рисунок 1.5: Композиционная схема автоматов модели фиктивного реактора: ТВС – тепловыделющая сборка, ВХ – Внутриреакторное хранилище; ЗВ – Зона воспроизводства; АЗ – Активная зона.

Глава 2

Метод построения оптимального порядка перезагрузки

В данной главе будут рассмотрены методы построения оптимального порядка перезагрузки реактора. Будет описан метод поиска порядка в графовом представлении ТА-модели реактора, а также исследованы алгоритмы поиска кратчайшего пути в графе.

2.1 Поиск оптимального порядка в ТА-модели реактора

Описанная в разделе 1.2.2 модель реактора обладает следующими свойствами:

- состояние ТА-модели q_0 является моделью состояния реактора к началу планово-профилактического ремонта;
- состояния ТА-модели из множества F являются искомыми состояниями, в которых планово-профилактический ремонт будет завершен;
- входное слово для ТА-модели является последовательно выполняемыми действиями над реактором, которые приводят его в новое состояние.

Таким образом, задача построения порядка проведения планово-профилактического ремонта реактора сводится к поиску входных слов конечного автомата, которые приводя его в конечное состояние.

Описанный в разделе 1.2.1 метод описания детерминированных конечных автоматов в виде диаграммы переходов представляет собой ориентированный граф. Вершинами графа являются состояния автомата, а ребрами графа — символы входного алфавита автомата. Тогда становится возможным произвести поиск по графу из вершины q_0 в вершины из множества F, используя алгоритмы поиска пути в графе (будут рассмотрены далее).

Каждое действие, производимое над реактором имеет различные трудо- и энергозатраты, следовательно для построения оптимального порядка перезагрузки требуется учитывать эти затраты в модели. Установим в соответствие каждому входному символу КДА коэффициент затрат и введем его в ТА-модель. Для этого каждому ребру графа поставим в соответствие коэффициент затрат, т.е. сделаем граф взвешенным. Тогда оптимальным порядком проведения планово-профилактического ремонта будет являться такая последовательность операций, которое приводит ТА-модель реактора из начального в конечное состояние с минимальной суммой энергозатрат.

2.2 Алгоритмы поиска пути в графе

Задача о кратчайшем пути — задача поиска самого короткого пути (цепи) между двумя точками (вершинами) на графе, в которой минимизируется сумма весов ребер, составляющих путь. Кратчайшая (простая) цепь часто называется геодезической. [8] Задача о кратчайшем пути является одной из важнейших классических задач теории графов. Сегодня известно множество алгоритмов для ее решения. У данной задачи существуют и другие названия: задача о минимальном пути или, в устаревшем варианте, задача о дилижансе.

Задача поиска кратчайшего пути на графе может быть определена для неориентированного, ориентированного или смешанного графа. Далее будет рассмотрена постановка задачи в самом простом виде для неориентированного графа. Для смешанного и ориентированного графа дополнительно должны учитываться направления ребер.

Граф представляет собой совокупность непустого множества вершин и ребер (наборов пар вершин). Две вершины на графе смежны, если они соединяются общим ребром. Путь в неориентированном графе представляет собой последовательность вершин $P = (v_1, v_2, \ldots, v_n) \in V \times V \times \cdots \times V$, таких что, v_i смежна с v_{i+1} для $1 \leqslant i < n$. Такой путь P называется путем длиной n из вершины v_1 в v_n (i указывает на номер вершины пути и не имеет никакого отношения к нумерации вершин на графе).

Пусть $e_{i,j}$ — ребро соединяющее две вершины: v_i и v_j. Дана весовая функция $f : E \to \mathbb{R}$, которая отображает ребра на их веса, значения которых выражаются действительными числами, и неориентированный граф G. Тогда кратчайшим путем из вершины v в вершину v' будет называться путь $P = (v_1, v_2, \ldots, v_n)$ (где $v_1 = v$ и $v_n = v'$), который имеет минимальное значение суммы $\sum_{i=1}^{n-1} f(e_{i,i+1})$. Если все ребра в графе имеют единичный вес, то задача сводится к определению наименьшего количества обходимых ребер. [9]

Существуют различные постановки задачи о кратчайшем пути:

- Задача о кратчайшем пути в заданный пункт назначения. Требуется найти кратчайший путь в заданную вершину назначения t, который начинается в каждой из вершин графа (кроме t). Поменяв направление каждого принадлежащего графу ребра, эту задачу можно свести к задаче о единой исходной вершине (в которой осуществляется поиск кратчайшего пути из заданной вершины во все остальные).

- Задача о кратчайшем пути между заданной парой вершин. Требуется найти кратчайший путь из заданной вершины u в заданную вершину v.

- Задача о кратчайшем пути между всеми парами вершин. Требуется найти кратчайший путь из каждой вершины u в каждую вершину v. Эту задачу тоже можно решить с помощью алгоритма, предназначенного для решения задачи об одной исходной вершине, однако обычно она решается быстрее.

В различных постановках задачи, роль длины ребра могут играть не только сами длины, но и время, стоимость, расходы, объем затрачиваемых ресурсов (материальных, финансовых, топливно-энергетических и т. п.) или другие характеристики, связанные с прохождением каждого ребра. Таким образом, задача находит практическое применение в большом количестве областей (информатика, экономика, география и др.).

В связи с тем, что существует множество различных постановок данной задачи, есть наиболее популярные алгоритмы для решения задачи поиска кратчайшего пути на графе:

- Алгоритм Дейкстры находит кратчайший путь от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

- Алгоритм Беллмана – Форда находит кратчайшие пути от одной вершины графа до всех остальных во взвешенном графе. Вес ребер может быть отрицательным.

- Алгоритм поиска A* находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной), используя алгоритм поиска по первому наилучшему совпадению на графе.

- Алгоритм Флойда — Уоршелла находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа.

- Алгоритм Джонсона находит кратчайшие пути между всеми парами вершин взвешенного ориентированного графа.

- Алгоритм Ли (волновой алгоритм) основан на методе поиска в ширину. Находит путь между вершинами s и t графа (s не совпадает с t), содержащий минимальное количество промежуточных вершин (ребер). Основное применение — трассировки электрических соединений на кристаллах микросхем и на печатных платах. Так же используется для поиска кратчайшего расстояния на карте в стратегических играх.

- Поиск кратчайшего пути на основе алгоритма Килдала.

Далее рассмотрим подробно алгоритм Дейкстры, алгоритм A* и волновой алгоритм.

2.2.1 Алгоритм Дейкстры

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским ученым Эдсгером Дейкстрой в 1959 году. Алгоритм Дейкстры решает задачу о кратчайших путях из одной вершины для взвешенного ориентированного графа $G = (V, E)$ с исходной вершиной s, в котором веса всех рёбер неотрицательны ($\omega(u, v) \geqslant 0$ для всех $(u, v) \in E$).

Формальное объяснение. В процессе работы алгоритма Дейкстры поддерживается множество $S \subseteq V$, состоящее из вершин v, для которых $\delta(s, v)$ уже найдено. Алгоритм выбирает вершину $u \in V$
S с наименьшим $d[u]$, добавляет u к множеству S и производит релаксацию всех рёбер, выходящих из u, после чего цикл повторяется. Вершины, не лежащие в S , хранятся в очереди Q с приоритетами, определяемыми значениями функции d. Предполагается, что граф задан с помощью списков смежных вершин.

Не формальное объяснение. Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Инициализация. Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещенные.

Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае из еще не посещенных вершин выбирается вершина u, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, соединенные с вершиной u ребрами, назовем соседями этой вершины. Для каждого соседа рассмотрим новую длину пути, равную сумме текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученная длина меньше метки соседа, заменим метку этой длиной. Рассмотрев всех соседей, пометим вершину u как посещенную и повторим шаг.

Время работы алгоритма Дейкстры. Сложность алгоритма Дейкстры зависит от способа нахождения вершины v, а также способа хранения множества непосещенных вершин и способа обновления меток. Обозначим через n количество вершин, а через m — количество ребер в графе G.

В простейшем случае, когда для поиска вершины с минимальным $d[v]$ просматривается все множество вершин, а для хранения величин d — массив, время работы алгоритма есть $O(n^2 + m)$. Основной цикл выполняется порядка n раз, в каждом из них на нахождение минимума тратится порядка n операций, плюс количество релаксаций (смен меток), которое не превосходит количества ребер в исходном графе.

Для разреженных графов (то есть таких, для которых m много меньше n^2) непосещенные вершины можно хранить в двоичной куче, а в качестве ключа использовать значения $d[i]$, тогда время извлечения вершины из U станет $\log n$, при том, что время модификации $d[i]$ возрастет до $\log n$. Так как цикл выполняется порядка n раз, а количество релаксаций не больше m, скорость работы такой реализации $O(n \log n + m \log n)$.

Если для хранения непосещенных вершин использовать фибоначчиеву кучу, для которой удаление происходит в среднем за $O(\log n)$, а уменьшение значения в среднем за $O(1)$, то время работы алгоритма составит $O(n \log n + m)$.

Пример работы алгоритма Дейкстры

Для примера возьмем такой ориентированный граф G (см. рис. 2.1):

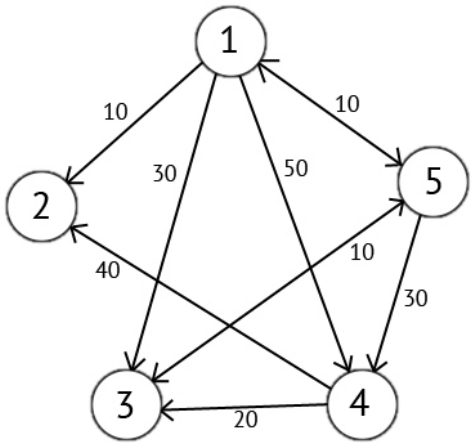

Рисунок 2.1: Пример алгоритма Дейкстры. Исходный граф

Возьмем в качестве источника вершину 1. Это значит что мы будем искать кратчайшие маршруты из вершины 1 в вершины 2, 3, 4 и 5. Данный алгоритм пошагово перебирает все вершины графа и назначает им метки, которые являются известным минимальным расстоянием от вершины источника до конкретной вершины. Рассмотрим этот алгоритм на примере.

Присвоим 1-й вершине метку равную 0, потому как эта вершина — источник. Остальным вершинам присвоим метки равные бесконечности (см. рис. 2.2).

Далее выберем такую вершину W, которая имеет минимальную метку (сейчас это вершина 1) и рассмотрим все вершины в которые из вершины W есть путь, не содержащий вершин посредников. Каждой из рассмотренных вершин назначим метку равную сумме метки W и длинны пути из W в рассматриваемую вершину, но только в том случае, если полученная сумма будет меньше предыдущего значения метки. Если же сумма не будет меньше, то оставляем предыдущую метку без изменений (см. рис. 2.3).

После того как мы рассмотрели все вершины, в которые есть прямой путь из W, вершину W мы отмечаем как посещённую, и выбираем из ещё не посещенных такую, которая имеет минимальное значение метки, она и будет следующей вершиной W. В данном слу-

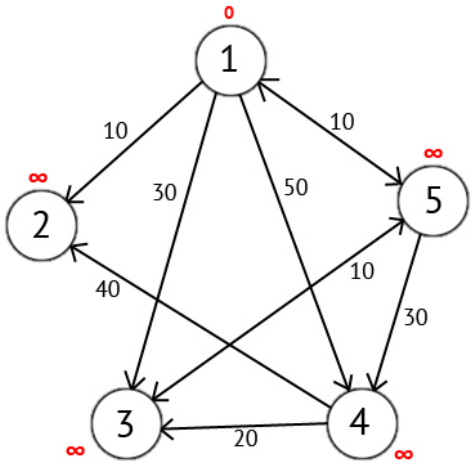

Рисунок 2.2: Пример алгоритма Дейкстры. Инициализация

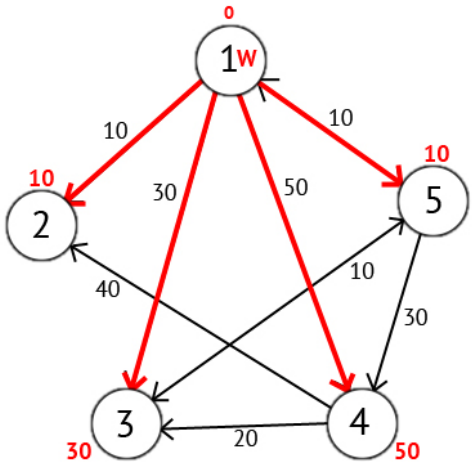

Рисунок 2.3: Пример алгоритма Дейкстры. Первый шаг алгоритма

чае это вершина 2 или 5. Если есть несколько вершин с одинаковыми метками, то не имеет значения какую из них мы выберем как W.

Мы выберем вершину 2. Но из нее нет ни одного исходящего пути, поэтому мы сразу отмечаем эту вершину как посещенную и переходим к следующей вершине с минимальной меткой. На этот раз только вершина 5 имеет минимальную метку. Рассмотрим все вершины в которые есть прямые пути из 5, но которые ещё не помечены как посещенные. Снова находим сумму метки вершины W и веса ребра из W в текущую вершину, и если эта сумма будет меньше предыдущей метки, то заменяем значение метки на полученную сумму (см. рис. 2.4).

Исходя из картинки мы можем увидеть, что метки 3-ей и 4-ой вершин стали меньше, то есть был найден более короткий маршрут в эти вершины из вершины источника. Далее отмечаем 5-ю вершину как посещенную и выбираем следующую вершину, которая имеет минимальную метку. Повторяем все перечисленные выше действия до тех пор, пока есть непосещенные вершины.

Выполнив все действия получим такой результат (см. рис. 2.5)

Также есть вектор P, исходя из которого можно построить кратчайшие маршруты. По количеству элементов этот вектор равен количеству вершин в графе. Каждый элемент содержит последнюю промежуточную вершину на кратчайшем пути между вершиной-

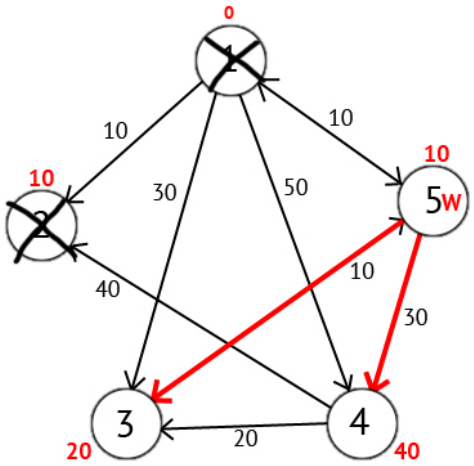

Рисунок 2.4: Пример алгоритма Дейкстры. Второй шаг алгоритма

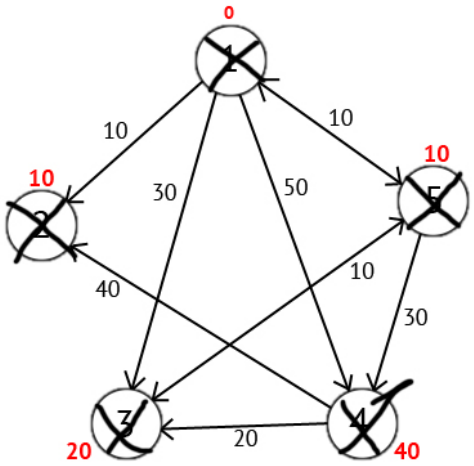

Рисунок 2.5: Пример алгоритма Дейкстры. Конец работы алгоритма

источником и конечной вершиной. В начале алгоритма все элементы вектора P равны вершине источнику (в нашем случае $P = \{1, 1, 1, 1, 1\}$). Далее на этапе пересчета значения метки для рассматриваемой вершины, в случае если метка рассматриваемой вершины меняется на меньшую, в массив P мы записываем значение текущей вершины W. Например: у 3-ей вершины была метка со значением «30», при $W = 1$. Далее при $W = 5$, метка 3-ей вершины изменилась на «20», следовательно, мы запишем значение в вектор P — $P[3] = 5$. Также при $W = 5$ изменилось значение метки у 4-й вершины (было «50», стало «40»), значит нужно присвоить 4-му элементу вектора P значение W — $P[4] = 5$. В результате получим вектор $P = \{1, 1, 5, 5, 1\}$.

Зная что в каждом элементе вектора P записана последняя промежуточная вершина на пути между источником и конечной вершиной, мы можем получить и сам кратчайший маршрут. [10]

2.2.2 Алгоритм A*

Алгоритм поиска A* — алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость» (обычно обозначаемой как $f(x)$). Эта функция — сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из начальной (обычно обозначается как $g(x)$ и может быть как эвристической, так и нет) и эвристической оценкой расстояния от рассматриваемой вершины к конечной (обозначается как $h(x)$). Функция $h(x)$ должна быть допустимой эвристической оценкой, то есть не должна переоценивать расстояния к целевой вершине. Например, для задачи маршрутизации $h(x)$ может представлять собой расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками.

A* пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдёт минимальный. Как и все информированные алгоритмы поиска, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. От жадного алгоритма (который тоже является алгоритмом поиска по первому лучшему совпадению) его отличает то, что при выборе вершины он учитывает, помимо прочего, весь пройденный до неё путь (составляющая $g(x)$ — это стоимость пути от начальной вершины, а не от предыдущей, как в жадном алгоритме). В начале работы просматриваются узлы, смежные с начальным; выбирается тот из них, который имеет минимальное значение $f(x)$, после чего этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа («множеством частных решений»), которое размещается в очереди с приоритетом. Приоритет пути определяется по значению $f(x) = g(x) + h(x)$. Алгоритм продолжает свою работу до тех пор, пока значение $f(x)$ целевой вершины не окажется меньшим, чем любое значение в очереди (либо пока всё дерево не будет просмотрено). Из множественных решений выбирается решение с наименьшей стоимостью.

Как и алгоритм поиска в ширину, A* является полным в том смысле, что он всегда находит решение, если таковое существует. Если эвристическая функция h допустима, то есть никогда не переоценивает действительную минимальную стоимость достижения цели, то A* сам является допустимым (или оптимальным), также при условии, что мы не отсекаем пройденные вершины. Если же мы это делаем, то для оптимальности алгоритма требуется, чтобы $h(x)$ была ещё и монотонной, или преемственной эвристикой. Свойство монотонности означает, что если существуют пути $A - B - C$ и $A - C$ (не обязательно через B), то оценка стоимости пути от A до C должна быть меньше либо равна сумме оценок путей $A - B$ и $B - C$. (Монотонность также известна как неравенство треугольника: одна сторона треугольника не может быть длиннее, чем сумма двух других сторон.) Математически, для всех путей x, y (где y – потомок x) выполняется:

$$g(x) + h(x) \leq g(y) + h(y). \tag{2.1}$$

A* также оптимально эффективен для заданной эвристики h. Это значит, что любой другой алгоритм исследует не меньше узлов, чем A* (за исключением случаев, когда существует несколько частных решений с одинаковой эвристикой, точно соответствующей стоимости оптимального пути). В то время как A* оптимален для «случайно» заданных графов, нет гарантии, что он сделает свою работу лучше, чем более простые, но и более информированные относительно проблемной области алгоритмы. Например, в неком лабиринте может потребоваться сначала идти по направлению от выхода, и только потом повернуть назад. В этом случае обследование вначале тех вершин, которые расположены ближе к выходу (по прямой дистанции), будет потерей времени. [11]

Пример работы алгоритма A*

Представим, что у нас есть поле, разделенное на сетку с ячейками, на котором отмечены начальная (зеленая) и конечная (красная) ячейки. Каждая ячейка имеет состояние

(открыта и закрыта), характеризующее её проходимость. В примере — открытые ячейки обведены точками, а закрытые пунктиром. Принцип алгоритма состоит в том, что он постепенно просматривает все окружающие текущую позицию ячейки и выбирает ячейку с наименьшим «весом». Стартовая ячейка по умолчанию является первой и единственной «открытой» ячейкой, с которой мы и начинаем поиск пути (см. рис. 2.6).

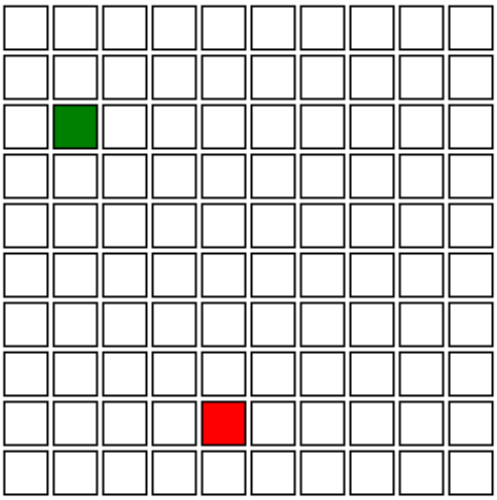

Рисунок 2.6: Пример алгоритма A*. Исходное поле

Теперь мы можем начать поиск пути. Ищем у текущей ячейки все «открытые» соседние ячейки и добавляем их в «открытый» список, а текущую ячейку в «закрытый» (см. рис. 2.7).

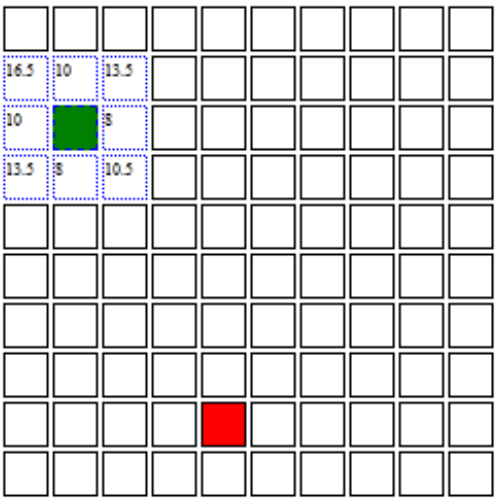

Рисунок 2.7: Пример алгоритма A*. Первый шаг алгоритма

Нужно переместиться в «открытую» соседнюю ячейку, выбрав ту, у которой стоимость (F) минимальна. В нашем примере стоимость (F) рассчитывается, как «прямое» расстояние до конечно точки, с учетом того, что ячейки по диагонали имеют вес в 1,5 раза больше, чем ортогональные (не по диагонали). Стоимость ячеек для наглядности указана внутри (см. рис. 2.8).

Продолжаем повторение действий:

- Выставляем текущей ячейке статус «закрытой»;
- Ищем «открытые» соседние ячейки;

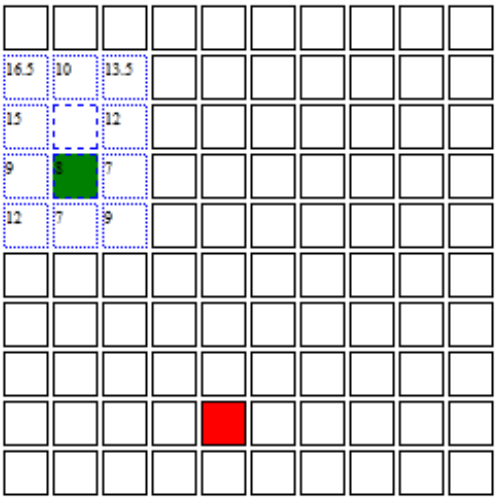

Рисунок 2.8: Пример алгоритма A*. Второй шаг алгоритма

- Рассчитываем стоимость (F);
- Перемещаемся в ячейку с минимальной стоимостью.

В результате мы дойдем до конечной ячейки самым коротким путем (см. рис. 2.9). [12]

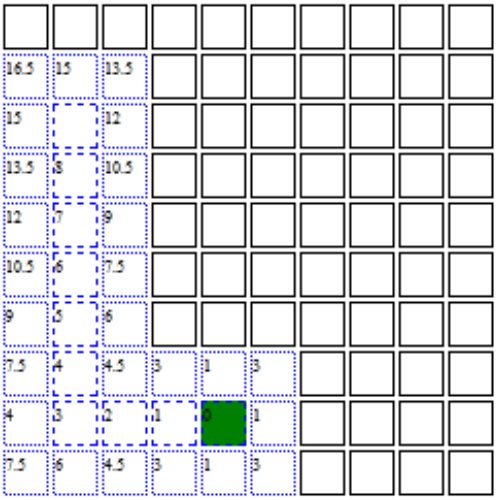

Рисунок 2.9: Пример алгоритма A*. Конец работы алгоритма

2.2.3 Волновой алгоритм

Алгоритм волновой трассировки (волновой алгоритм, алгоритм Ли) — алгоритм поиска кратчайшего пути на планарном графе. Принадлежит к алгоритмам, основанным на методах поиска в ширину.

В основном используется при компьютерной трассировке (разводке) печатных плат, соединительных проводников на поверхности микросхем. Другое применение волнового алгоритма — поиск кратчайшего расстояния на карте в компьютерных стратегических играх.

Волновой алгоритм в контексте поиска пути в лабиринте был предложен Э.Ф. Муром. Ли независимо открыл этот же алгоритм при формализации алгоритмов трассировки печатных плат в 1961 году.

Описание алгоритма. Алгоритм работает на «дискретном рабочем поле» (ДРП), представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные ячейки, в частном случае — квадратные. Множество всех ячеек ДРП разбивается на подмножества: «проходимые» (свободные), т. е при поиске пути их можно проходить, «непроходимы» (препятствия), путь через эту ячейку запрещен, стартовая ячейка (источник) и финишная (приемник). Назначение стартовой и финишной ячеек условно, достаточно — указание пары ячеек, между которыми нужно найти кратчайший путь.

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути выдать сообщение о непроходимости.

Работа алгоритма включает в себя три этапа: «инициализацию», «распространение волны» и «восстановление пути».

Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки.

Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке.

Соседние ячейки принято классифицировать двояко: в смысле окрестности Мура и окрестности фон Неймана, отличающийся тем, что в окрестности фон Неймана соседними ячейками считаются только 4 ячейки по вертикали и горизонтали, в окрестности Мура — все 8 ячеек, включая диагональные.

При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, от стартовой ячейки на первом шаге это будет 1. Каждая ячейка, меченая числом шагов от стартовой ячейки становится стартовой и из неё порождаются очередные шаги в соседние ячейки. Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки. Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек.

Вычислительная сложность волнового алгоритма близка к $O(N^2)$. В реальных задачах, например при трассировке печатных плат, приложение пути (трассы) выполняется многократно, что влечет существенные временные затраты. [13]

Пример работы волнового алгоритма

ДРП — это прямоугольник, разбитый на квадратные ячейки одинакового размера. Ячейки ДРП подразделяются на свободные, препятствия, источники и приемники. На рис. 2.10 свободные ячейки имеют светло-зеленый цвет, а препятствия — светло-коричневый. Источник залит синим цветом, а приемник — черным. Путь может быть проложен только по свободным ячейкам.

Рассматривается алгоритм построения ортогонального пути. Алгоритм состоит из двух частей. В первой от источника к приемнику распространяется волна. Во второй выполняется обратный ход, в процессе которого из ячеек волны формируется путь. Волна, идущая от источника к приемнику, на каждом шаге первой части алгоритма пополняется свободными ячейками ДРП, которые, во-первых, еще не принадлежат волне, и, во-вторых, явля-

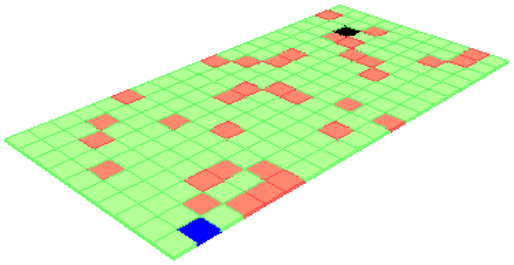

Рисунок 2.10: Пример волнового алгоритма. Исходное поле

ются 4-соседями ячеек, попавших в волну на предыдущем шаге. Процесс распространения волны иллюстрируют рис. 2.11 – 2.14.

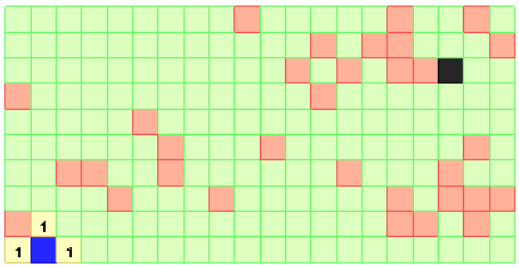

Рисунок 2.11: Пример волнового алгоритма. Первый шаг распространения волны

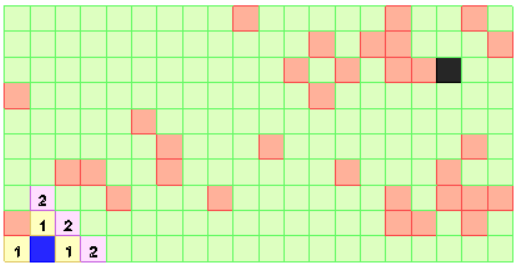

Рисунок 2.12: Пример волнового алгоритма. Второй шаг распространения волны

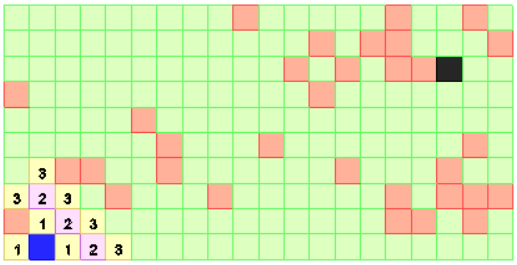

Рисунок 2.13: Пример волнового алгоритма. Третий шаг распространения волны

В примере волна достигла ячейку-приемник за 23 шага. При обратном ходе в путь включается по одной ячейке каждого шага распространения волны. При выборе из двух ячеек приоритет имеет ячейка, обеспечивающая горизонтальное продвижение, что приводит к пути, показанному на рис. 2.15.

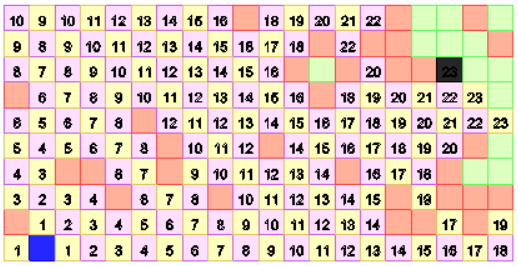

Рисунок 2.14: Пример волнового алгоритма. Последний шаг распространения волны

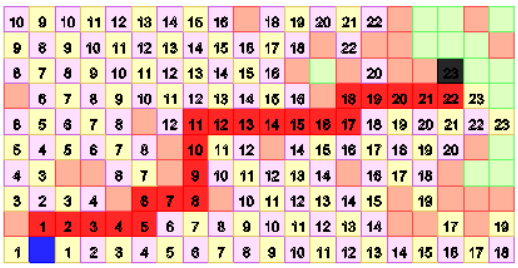

Рисунок 2.15: Пример волнового алгоритма. Обратный ход: формирование пути

2.3 Сравнительный анализ алгоритмов

Многообразие алгоритмов поиска пути обусловлено многообразием их применений, для каждого из которых эффективнее работает определенный алгоритм поиска пути.

- Алгоритм Дейкстры приоритетен в случаях поиска пути до всех точек области поиска, а также в случае отсутствия сколь либо эффективной эвристической функции оценки расстояния между элементами области поиска.

- Волновой алгоритм эффективен, если область поиска имеет неравномерную проходимость, что затрудняет эвристические вычисления для A*.

- Алгоритм A* эффективен при одиночном поиске пути между двумя точками, если возможно эффективно эвристически получать примерную дистанцию между элементами области поиска.

- Навигационная сетка с использованием алгоритма A* эффективна при создании AI, в чьи задачи входит не только поиск пути. Также этот метод эффективен при необходимости построить реалистичную сглаженную траекторию движения между двумя точками. Данный метод предполагает наличие детальной информации об области поиска или возможности получения такой информации.

- Эвристические алгоритмы поиска пути применимы и оптимальны, если необходим максимально простой алгоритм, при этом область поиска достаточно проста, а применения алгоритма допускают неточность полученного пути.

Таким образом, для построения оптимального порядка планово-профилактического ремонта реактора оптимальным будет использование алгоритма A* к графу ТА-модели реактора. Особым преимуществом алгоритма A* является наличие эвристической функции. Это позволяет проводить различные виды оптимизации ППР, изменяя только используемую эвристическую функцию.

Глава 3

Способ адаптивного моделирования параметров активной зоны

3.1 Описание способа адаптивного моделирования

Рассмотренный в главе 2 метод построения оптимального порядка имеет существенный недостаток: в базовой ТА-модели не представлены физические расчеты состояния реактора, подтверждающие безопасность каждого состояния. Согласно [14], расчеты состояния реактора являются довольно сложной вычислительной задачей. Для полного расчета всех состояний реактора потребуется проводить расчет для каждого состояния q из множества состояний автомата Q, что делает задачу вычислительно невыполнимой.

Однако, можно понизить вычислительную сложность метода, используя обобщенный аппрокисматор. Нейронные сети могут аппроксимировать непрерывные функции. Доказана обобщённая аппроксимационная теорема [15]: с помощью линейных операций и каскадного соединения можно из произвольного нелинейного элемента получить устройство, вычисляющее любую непрерывную функцию с некоторой наперёд заданной точностью. Это означает, что нелинейная характеристика нейрона может быть произвольной: от сигмоидальной до произвольного волнового пакета или вейвлета, синуса или многочлена. От выбора нелинейной функции может зависеть сложность конкретной сети, но с любой нелинейностью сеть остаётся универсальным аппроксиматором и при правильном выборе структуры может достаточно точно аппроксимировать функционирование любого непрерывного автомата.

В последние несколько лет наблюдается взрыв интереса к нейронным сетям, которые успешно применяются в самых различных областях — бизнесе, медицине, технике, геологии, физике. Нейронные сети вошли в практику везде, где нужно решать задачи прогнозирования, классификации или автоматизации. Такой успех определяется несколькими причинами.

1. Богатыми возможностями. Нейронные сети — мощный метод моделирования, позволяющий воспроизводить чрезвычайно сложные зависимости. Нейросети нелинейны по своей природе. Кроме того, нейронные сети справляются с «проклятием размерности», которое не позволяет моделировать линейные зависимости в случае большого числа переменных.

2. Простотой в использовании. Нейронные сети учатся на примерах. Пользователь нейронной сети подбирает представительные данные, а затем запускает алгоритм обучения, который автоматически воспринимает структуру данных. От пользователя, конечно, требуется какой-то набор эвристических знаний о том, как следует отбирать и подготавливать данные, выбирать нужную архитектуру сети и интерпретировать

результаты, однако уровень знаний, необходимый для успешного применения нейронных сетей, гораздо скромнее, чем, например, при использовании традиционных методов статистики. [16]

Таким образом, можно предложить следующий способ адаптивного моделирования:

1. Рассчитываются обучающее множество для нейронной сети состоящее из вариативных параметров активной зоны и соответствующих им характеристик активной зоны;
2. Производится обучение искусственной нейронной сети на обучающем множестве;
3. При поиске оптимального порядка обслуживания используются результаты работы искусственной нейронной сети в качестве предварительных оценок параметров активной зоны;
4. Предложенный порядок проверяется полным физическим расчетом.

3.2 Искусственные нейронные сети прямого распространения

3.2.1 Определение искусственной нейронной сети

Искусственными нейронными сетями (ИНС) называется совокупность моделей биологических нейронных сетей. ИНС представляет собой сеть искусственных нейронов, связанных между собой искусственными синаптическими соединениями. Сеть обрабатывает входную информацию и в процессе изменения своего состояния во времени формирует совокупность выходных сигналов. [17] Так как ИНС моделирует реальную нейронную сеть, рассмотрим принципы функционирования биологического образца. Структурной единицей биологической нейронной сети является нейрон (нервная клетка).

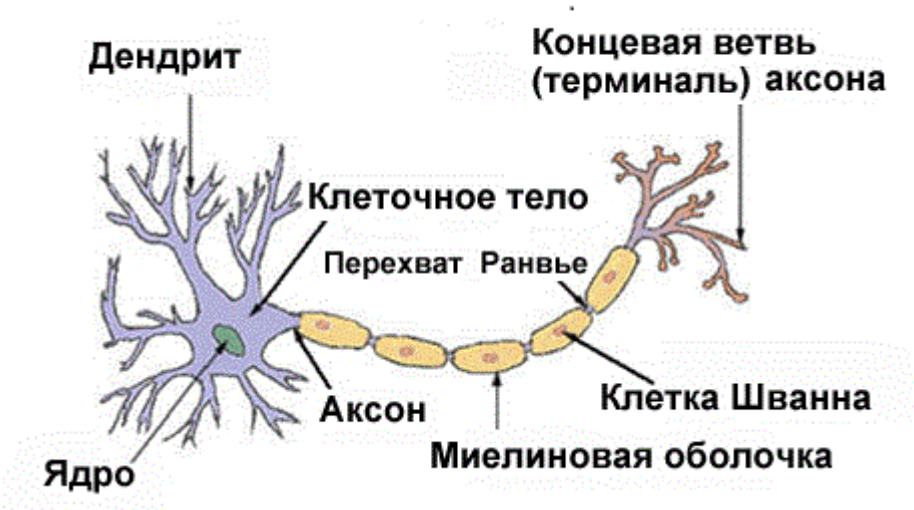

Рисунок 3.1: Структура нейрона

Нейрон (рис. 3.1) является особой биологической клеткой, которая обрабатывает информацию. Она состоит из тела клетки, или сомы, и двух внешних древоподобных ветвей: аксона и дендритов. Тело клетки включает ядро, которое содержит информацию о наследственных свойствах, и плазму, обладающую молекулярными средствами для производства необходимых нейрону материалов. Нейрон получает сигналы (импульсы) от других нейронов через дендриты и передают сигналы, сгенерированные телом клетки, вдоль

аксона, который в конце разветвляется на волокна. На окончаниях этих волокон находятся синапсы. Синапс является элементарной структурой и функциональным узлом между двумя нейронами (волокно аксона одного нейрона и дендрит другого). Когда импульс достигает синаптического окончания, высвобождаются определенные химические вещества, называемые нейротрансмиттерами. Нейротрансмиттеры диффундируют через синаптическую щель, возбуждая или затормаживая, в зависимости от типа синапса, способность нейрона-приемника генерировать электрические импульсы. Результативность синапса может настраиваться проходящими через него сигналами, так что синапсы могут обучаться в зависимости от активности процессов, в которых они участвуют. Эта зависимость от предыстории действует как память, которая, возможно, ответственна за память человека. [18]

Созданные из таких структурных элементов биологический нейронные сети обладают следующими свойствами:

- Параллельность обработки информации. Каждый нейрон формирует свой выход только на основе своих входов и собственного внутреннего состояния под воздействием общих механизмов регуляции нервной системы.

- Способность к полной обработке информации. Все известные человеку задачи решаются нейронными сетями. К этой группе свойств относятся ассоциативность (сеть может восстанавливать полный образ по его части), способность к классификации, обобщению, абстрагированию и множество других.

- Самоорганизация. В процессе работы БНС самостоятельно, под воздействием внешней среды, обучаются решению разнообразных задач. Неизвестно никаких принципиальных ограничений на сложность задач, решаемых БНС. Нервная система сама формирует алгоритмы своей деятельности, уточняя и усложняя их в течении жизни.

- Надежность. Биологические НС обладают фантастической надежностью: выход из строя даже 10% нейронов в нервной системе не прерывает ее работы. По сравнению с последовательными ЭВМ, основанными на принципах фон Неймана, где сбой одной ячейки памяти или одного узла в аппаратуре приводит к краху системы. [17]

3.2.2 Математические модели работы нейронов

Детерминистская модель

Как было выяснено ранее, нейрон представляет собой единицу обработки информации в нейронной сети. На блок-схеме рис. 3.2 показана модель нейрона, лежащая в основе искусственных нейронных сетей. В этой модели можно выделить три основных функциональных элемента.

1. Набор синапсов или связей, каждый из которых характеризуется своим весом или силой. В частности, сигнал x_j на входе синапса j, связанного с нейроном k, умножается на вес ω_{jk}. В отличии от синапсов мозга синаптический вес искусственного может принимать как положительные, так и отрицательные значения.

2. Сумматор складывает входные сигналы, взвешенные относительно соответствующих весов синапсов нейронов. Математически эту операцию можно описать как линейную комбинацию входных сигналов.

3. Функция активации ограничивает выходную амплитуду сигнала нейрона. Эта функция также называется функцией сжатия. Обычно выходной сигнал нормализуется в диапазоне $[0; 1]$ или $[-1; 1]$.

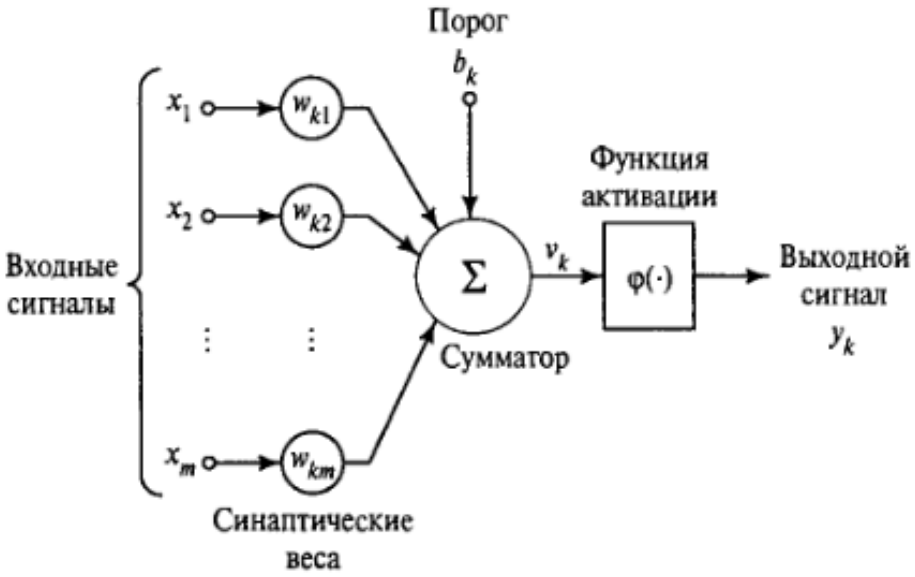

Рисунок 3.2: Явная нелинейная модель нейрона

В модель нейрона, показанную на рис. 3.2, включён пороговый элемент, который обозначается символом b_k. Эта величина отражает увеличение или уменьшение входного сигнала функции активации. В математической форме функционирование нейрона k можно описать следующей парой уравнений:

$$u_k = \sum_{j=1}^{m} \omega_{kj} x_j \tag{3.1}$$

$$y_k = \varphi(u_k + b_k) \tag{3.2}$$

где $x_1, x_2, \ldots, x_m$ — входные сигналы; $\omega_{k1}, \omega_{k2}, \ldots, \omega_{km}$ — синаптические веса нейрона k; u_k — линейная комбинация входных воздействий; b_k — порог; $\varphi()$ — функция активации; y_k — выходной сигнал нейрона. [19]

Функции активации

Функция активации, представленные в формулах как $\varphi()$, определяет выходной сигнал нейрона. Можно выделить три основных типов функции активации.

1. Функция единичного скачка, или пороговая функция. Этот тип функции описывается следующим уравнением:

$$\varphi(v) = \begin{cases} 1, v \geq 0 \\ 0, v < 0 \end{cases} \tag{3.3}$$

В технической литературе эта форма функции единичного скачка называется функцией Хэвисайда. Соответственно выходной сигнал нейрона k можно представить как

$$y_k = \begin{cases} 1, v_k \geq 0 \\ 0, v_k < 0 \end{cases} \tag{3.4}$$

где v_k — индуцированное локальное поле нейрона k. Эту модель в литературе называют моделью Мак-Каллока-Питца. В этой модели выходной сигнал нейрона принимает значение 1 при неотрицательном индуцированном локальным полем, и 0 —

в противном случае. Это выражение описывает свойство «все или ничего» модели Мак-Каллока-Питца.

2. Кусочно-линейная функция. Кусочно-линейная функция описывается следующим выражений:

$$\varphi(v) = \begin{cases} 1, v \geq +\frac{1}{2} \\ |v|, -\frac{1}{2} < v < +\frac{1}{2} \\ 0, v \leq -\frac{1}{2} \end{cases} \tag{3.5}$$

где коэффициент усиления в линейной области оператора предполагается равным единице. Следующие два варианта можно считать особой формой кусочно-линейной функции.

   - Если линейная область не достигает порога насыщения, они превращается в линейный сумматор
   - Если коэффициент усиления линейной области стремиться к бесконечности, то кусочно-линейная функция выражается в пороговую.

3. Сигмоидальная функция. Сигмоидальная функция, график которой напоминает букву S, является наиболее распространённой функцией для построения искусственных нейронных сетей. Это быстрорастущая функция, поддерживающая баланс между линейным и нелинейным поведением. Примером сигмоидальной функции может служить логистическая функция, задаваемая выражением:

$$\varphi(v) = \frac{1}{1 + e^{-av}} \tag{3.6}$$

где a — параметр наклона сигмоидальной функции. Изменяя этот параметр можно построить функции с различной крутизной. При бесконечно большом параметре наклона функция вырождается в пороговую. Если пороговая функция принимает только два значения (0 и 1), то сигмоидальная функция принимает бесконечное количество значений в диапазоне $[0; 1]$. При этом сигмоидальная функция является в отличии от пороговой дифференцируемая. [19]

3.2.3 Топологии нейронных сетей

Структура нейронных сетей тесно связана с используемыми алгоритмами обучения. В общем случае можно выделить три фундаментальных архитектуры нейронных сетей.

Однослойные прямого распространения

В многослойных нейронной сети нейроны располагаются по слоям. В простейшем случае у такой сети существует входной слой узлов источника, информация от которого передается на выходной слой нейронов (вычислительные узлы), но не наоборот. Такая сеть называется сетью прямого распространения или ациклической сетью.

На рис. 3.3 показана структура такой сети для случая четырёх узлов в каждом из слоев (входном и выходном). Такая нейронная сеть называется однослойной, при этом под единственным слоем подразумевается слой вычислительных элементов (нейронов). При подсчёте числа слоев мы не принимаем во внимание узлы источника, так как они не выполняют никаких вычислений. [19]

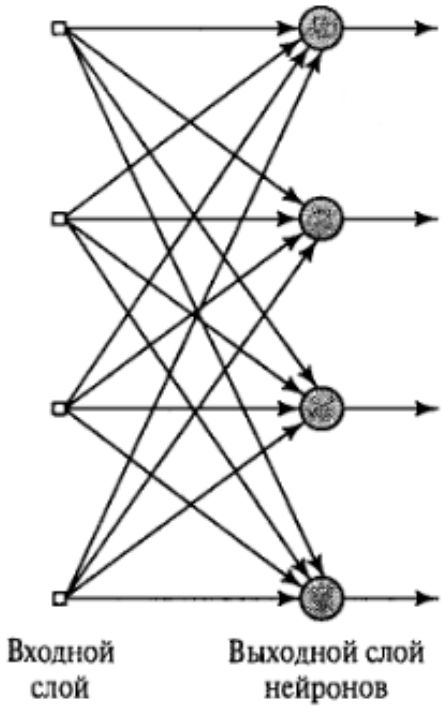

Рисунок 3.3: Однослойная сеть прямого распространения

Многослойные прямого распространения

Другой класс нейронных сетей прямого распространения характеризуется наличием одного или нескольких скрытых слоев, узлы которых называют скрытыми нейронами или скрытыми элементами. Функция последних заключается в посредничестве между внешним входным сигналом и выходом нейронной сети. Добавляя один или несколько нейронных слоев мы можем выделить статистики более высокого порядка. Такая сеть позволяет выделять глобальные свойства данных с помощью локальных соединений за счёт наличия дополнительных синаптических связей и повышения уровня взаимодействий нейронов. Способность скрытых нейронов выделять статистические зависимости высокого порядка особенно существенна, когда размер входного слоя достаточно велик.

Узлы источника входного слоя сети формируют соответствующие элементы шаблона активации (входной вектор), которые составляют входной сигнал, поступающий на нейроны (вычислительные элементы) второго слоя (т.е. первого скрытого слоя). Выходные сигналы второго слоя используются в качестве входных для третьего слоя и т.д. Обычно нейроны каждого из слоев сети используют в качестве входных сигналов только выходные сигналы нейронов предыдущего слоя. Набор выходных сигналов нейронов последнего слоя сети определяет общий отклик сети на данный входной образ, сформированный узлами источника входного слоя.

Сеть, показанная на рис. 3.4, называется сетью 10-4-2, так как имеет 10 входных, 4 скрытых и 2 выходных нейрона. Нейронная сеть, показанная на рис. 3.4, считается полносвязной в том смысле, что все узлы каждого конкретного слоя соединены со всеми узлами смежных слоев. Если некоторые из синаптических связей отсутствуют, то такая сеть называется неполносвязной. [19]

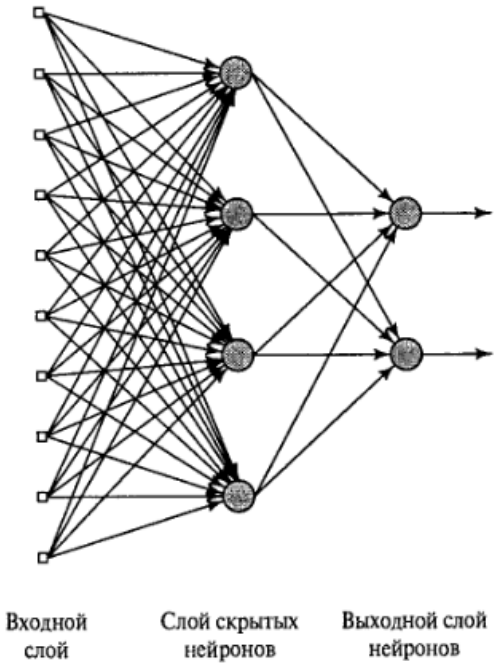

Рисунок 3.4: Многослойная сеть прямого распространения

3.2.4 Алгоритмы обучения нейронных сетей

Самым важным свойством нейронных сетей является их способность обучаться на основе данных окружающей среды и в результате обучения повышать свою производительность. Повышение производительности происходит со временем в соответствии с определёнными правилами. Обучение нейронной сети происходит посредством интерактивного процесса корректировки синаптических весов и порогов. В идеальном случае нейронная сеть получает знания об окружающей среде на каждой итерации процесса обучения.

С понятием обучения ассоциируется довольно много видов деятельности, поэтому сложно дать этому процессу однозначное определение. С позиций нейронных сетей мы можем использовать следующее определение обучения Обучение — это процесс, в котором свободные параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена. Тип обучения определяется способом подстройки этих параметров.

Это определение предполагает следующую последовательность событий при обучении нейронной сети:

1. В нейронную сеть поступают стимулы из внешней среды.
2. В результате этого изменяются свободные параметры нейронной сети.
3. После изменения внутренней структуры нейронная сеть отвечает на возбуждения уже иным образом.

Вышеуказанный список четких правил решения проблемы обучения называется алгоритмом обучения. Несложно догадаться, что не существует универсального алгоритма обучения, подходящего для всех архитектур нейронных сетей. Существует лишь набор средств, предоставленный множеством алгоритмов обучения, каждый из которых имеет свои достоинства. [19]

Обучение на основе коррекции ошибок

Для того, чтобы проиллюстрировать первое правило обучения, рассмотрим простейший случай нейрона k — единственного вычислительного узла выходного слоя нейронной сети прямого распространения. Нейрон k работает под управлением вектора сигнала $\vec{x}(n)$, производимого одним или несколькими скрытыми слоями нейронов, которые в свою очередь получают информацию из входного вектора, передаваемого начальным узлам нейронной сети. Под n подразумевается дискретное время или, более конкретно, — номер шага итеративного процесса настройки синаптических весов нейрона k. Выходной сигнал нейрона k обозначается $y_k(n)$. Этот сигнал будет сравниваться с желаемым выходом, обозначенным $d_k(n)$. В результате получим сигнал ошибки $e_k(n)$. По определению

$$e_k(n) = y_k(n) - d_k(n) \tag{3.7}$$

Сигнал ошибки инициализирует механизм управления, цель которого заключается применении последовательности корректировок к синаптическим весам нейрона k. Эти изменения нацелены на пошаговое приближение выходного сигнала $y_k(n)$ к желаемому $d_k(n)$. Эта цель достигается за счет минимизации функции стоимости или индекса производительности $E(n)$, определяемой в терминах сигнала ошибки следующим образом:

$$E(n) = \frac{1}{2}e_k^2(n) \tag{3.8}$$

где $E(n)$ — текущее значение энергии ошибки. Пошаговая корректировка синаптических весов нейрона k продолжается пока до тех пор, пока система не достигнет устойчивого состояния. В этой точке процесс обучения останавливается.

Процесс, описанный выше, называется обучением на основе коррекции ошибок. Минимизация функции стоимости $E(n)$ выполняется по так называемому дельта-правилу, или правилу Видроу-Хоффа, названному в честь его создателей. Обозначим $\omega_{kj}(n)$ текущее значение синаптического веса ω_{kj} нейрона k, соответствующему элементу $x_j(n)$ вектора $\vec{x}(n)$, на шаге дискретизации n. В соответствии с дельта-правилом изменение $\Delta\omega_{kj}(n)$, применяемое к синаптическому весу ω_{kj} на этом шаге дискретизации, задается выражением

$$\Delta\omega_{kj}(n) = \eta e_k(n)x_j(n) \tag{3.9}$$

где η — некоторая положительная константа, определяющая скорость обучения и используемая при переходе от одного шага процесса к другому. Эту константу естественно именовать параметром скорости обучения. Вербально дельта-правило можно определить следующим образом:

Корректировка, применяемая к синаптическому весу нейрона, пропорциональна произведению сигнала ошибки на входной сигнал, его вызвавший.

Необходимо помнить, что определенное таким образом дельта-правило предполагает возможность прямого измерения сигнала ошибки. Для обеспечения такого измерения требуется поступление желаемого отклика от некоторого внешнего источника, непосредственно доступного для нейрона k. Другими словами нейрон k должен быть видимым для внешнего мира. Вычислив величину изменения синаптического веса $\Delta\omega_{kj}(n)$, можно определить его новое значение для следующего шага дискретизации:

$$\Delta\omega_{kj}(n+1) = \omega_{kj}(n) + \Delta\omega_{kj}(n) \tag{3.10}$$

3.2.5 Парадигмы обучения нейронных сетей

Обучение с учителем

Рассмотрим парадигмы обучения нейронных сетей. Начнем с парадигмы обучения с учителем.

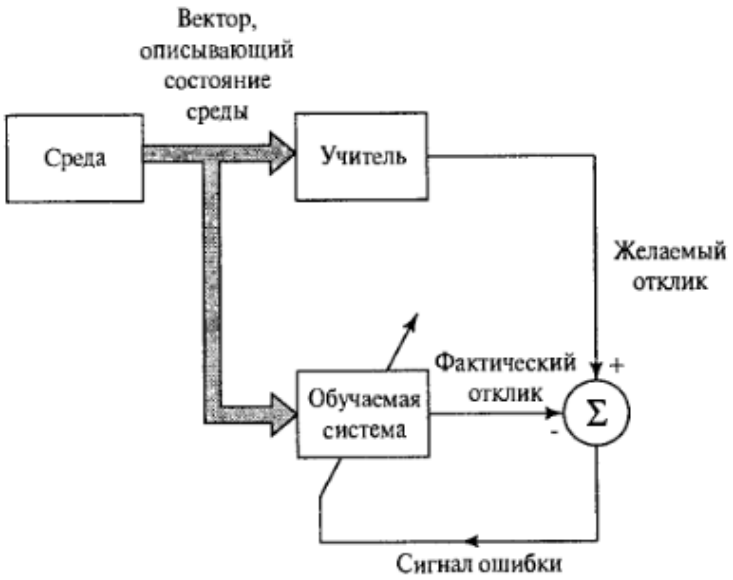

Рисунок 3.5: Блочная диаграмма обучения с учителем

На рис. 3.5 приведена блочная диаграмма, иллюстрирующая эту форму обучения. Концептуально участие учителя можно рассматривать как наличие знаний об окружающей среде, представленных в виде пар вход-выход. При этом сама среда неизвестна обучаемой нейронной сети. Теперь предположим, что учителю и обучаемой сети подается из окружающей среды вектор. На основе встроенных знаний учитель может сформировать и передать обучаемой нейронной сети желаемый отклик, соответствующий входному вектору. Этот желаемый результат представляет оптимальные действия, которые должна выполнить нейронная сеть. Параметры сети корректируются с учетом обучающего вектора и сигнала ошибки. Корректировка параметров должна выполняться пошагово с целью имитации нейронной сетью поведения учителя. Эта эмуляция в некотором статистическом смысле должна быть оптимальной. Таким образом, в процессе обучения знания учителя передаются в сеть в максимально полном объеме.

Глава 4

Метод моделирования и оптимизации процессов перезагрузки реакторов

В данной главе будет рассмотрен метод моделирования и оптимизации процессов перезагрузки реактора с использованием методов искусственного интеллекта. Будут рассмотрены входные и выходные данные для моделирования, а также алгоритмическое исполнение метода. В конце главы будут представлены преимущества и недостатки данного метода.

4.1 Исходные данные для моделирования

Пусть у нас имеется следующая информация о реакторе и его процессе перезагрузки:

1. ТА-модель. ТА-модель представляет собой описание реактора и его процесса ППР в виде автомата или композиции автоматов. То есть задано множество состояний реактора, множество операций над реактором, функция перехода, указывающая возможные переходы при из одного конечного состояния в другое при выполнении операции, а также начальное и множество конечных состояний. При этом для каждого состояния реактора и каждой операции заданы известные физические характеристики.

2. Данные физического моделирования. Данные физического моделирования представляют собой множество расчетных (или известных) характеристик реактора, соответствующих определенному физическому состоянию реактору. Примером таких данных может служить зависимость подкритичности от физической конфигурации активной зоны. Данное множество необязательно должно содержать все значения из множества состояния реактора. При этом могут использоваться данные прошлых кампаний. Также для этого каждого значения множества должна быть указана допустимость таких расчетных характеристик.

3. Функция оптимизации. Функция оптимизации — весовая функция для характеристик операций, проводимых над реактором. Она требуется для изменения значимости различных факторов при проведении оптимизации. Поиск оптимального порядка будет производится по минимальным значениям данной функции.

Обработав исходные данные, метод должен представить на выходе оптимальный с точки зрения функции оптимизации порядок проведения перезагрузки реактора. При этом искомый порядок не должен содержать физически недопустимых состояний (см. рис. 4.1).

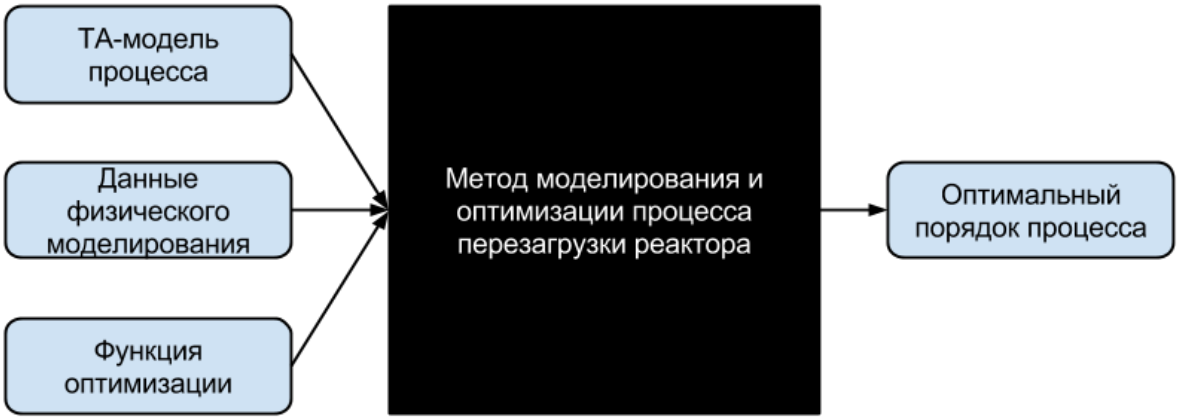

Рисунок 4.1: Входные и выходные данные метода моделирования и оптимизации

4.2 Описание метода моделирования и оптимизации

Рассмотрим работу метода моделирования и оптимизации перегрузки реактора (см. рис. 4.2).

1. Искусственная нейронная сеть обучается на данных физического моделирования. После успешного обучения нейронная сеть становится способна аппроксимировать расчетные характеристики реактора на основе физического состояния реактора, а также выносить решение о допустимости состояния.

2. Множество состояний ТА-модели реактора поэлементно подается в качестве входного вектора искусственной нейронной сети. ИНС делит множество на допустимые и недопустимые состояния. Допустимые состояния становятся узлами графа, а недопустимые — отбрасываются.

3. Между узлами графа проводятся направленные ребра на основе множества операции и функции перехода. Таким образом после завершения операции имеется граф переходов между допустимыми состояниями.

4. В полученном графе запускается алгоритм поиска A*. Алгоритм производит поиск кратчайшего пути между начальным состоянием и каждым состоянием из множества конечных. При этом в качестве эвристической функции используется функция оптимизации.

5. По завершению работы алгоритма поиска имеется множество путей, ведущих кратчайшим путем к каждому из конечных состояний. Путь с минимальным значением функции оптимизации является искомым.

Полученные в результате работы метода порядок перезагрузки необходимо проверить сертифицированными средствами физического моделирования реактора.

4.3 Преимущества и недостатки метода

К преимуществам данного метода относятся:

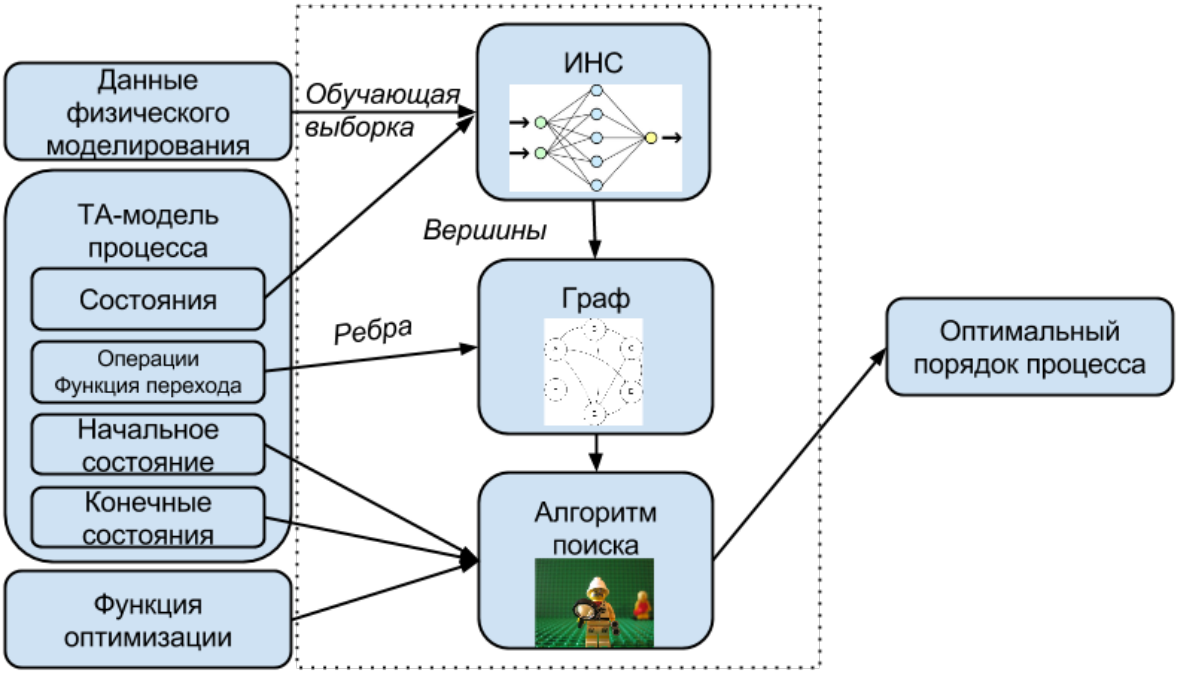

Рисунок 4.2: Метод моделирования и оптимизации

- Универсальность. Данный метод не ограничивается одним типом реакторов. При наличии исходных данных метод можно применить к любому типу реактора. В перспективе метод можно будет применить к любым системам, описываемым формализмом ТА-моделей

- Гибкость. Задаваемая функция оптимизации позволяет оптимизировать одну и ту же ТА-модель по различным параметрам. Для этого не требуется изменение метода и/или модели.

- Высокая скорость работы. За счет применения информированного алгоритма поиска и искусственных нейронных сетей для аппроксимации физических характеристик метод имеет приемлемую вычислительную сложность.

К недостаткам данного метода относятся:

- Сложность описания ТА-модели. Для построения ТА-модели реактора в нужной детализации требуется достаточно много усилий. Это увеличивает вероятность ошибок в задании модели реактора, а, следовательно, в расчетах.

- Возможность ошибки аппроксимации в ИНС. ИНС может иметь ошибки аппроксимации, связанные с переобучением или отсутствием информации об особенностях данной области аппроксимации.

Глава 5

Разработка программного комплекса

В данной главе будет описан процесс разработки программного комплекса, который позволяется смоделировать и построить оптимальный порядок перегрузки реактора.

5.1 Описание архитектуры программного комплекса

Программный комплекс для моделирования и построения оптимального порядка перезагрузки реактора представляет собой модульное веб-приложение, состоящие из веб-интерфейса управления и 3 модулей: сервис хранения состояний, сервис расчета порядка и сервис моделирования. Каждый модуль предоставляет интерфейс прикладного программирования (API), который работает по схеме передачи репрезентативного состояния. То есть каждый модуль является отдельным REST-сервисом. Данные между сервисами передаются в формате JSON.

Такая архитектура позволяет развернуть программный комплекс в облачных сервисах PaaS и IaaS. При этом будет достигнута возможность горизонтального масштабирования программного комплекса при необходимости.

5.1.1 Сервис хранения состояний

Сервис хранения состояний представляет собой хранилище TA-моделей. Пользователь имеет возможность создания, чтения, редактирования и удаления TA-моделей. Таким образом сервис хранения состояния является персистентным хранилищем моделей, предоставляемых пользователем.

TA-модель в сервисе задается как кортеж из пяти элементов: множество состояний, множество операций, множество допустимых переходов, начальное состояние и множество конечных состояний. Каждое состояние имеет свое название и словарь дополнительных данных. Каждая операция также имеет свое название и словарь дополнительных данных.

Сервис хранения состояний имеет следующие методы REST API (см. таблицу 5.1). Пример работы API приведен в приложении A.1.

Архитектура сервиса представлена на рис. 5.1. Поступающие HTTP-запросы обрабатываются HTTP-сервером фреймворка Flask. В зависимости от типа запроса расширение Flask-RESTful API маршрутизирует запросы на соответствующий парсер запроса. Каждый парсер изменяет модель данных в соответствии с запросом. При изменении модели данных ORM фреймворк SQLAlchemy производит синхронизацию данных модели и данных в базе данных. При этом SQLAlchemy может работать с большинством современных СУБД без изменения программного кода.

Таблица 5.1: Методы API сервиса хранения состояния

Адрес	HTTP-метод	Выполняемое действие
api/automata	GET	Получить список всех автоматов
api/automata	POST	Добавить новое описание автомата (из JSON-данных запроса)
api/automata/<id>	GET	Получить описание автомата номер <id>
api/automata/<id>	PUT	Изменить описание автомата номер <id> (из JSON-данных запроса)
api/automata/<id>	DELETE	Получить описание автомата номер <id>

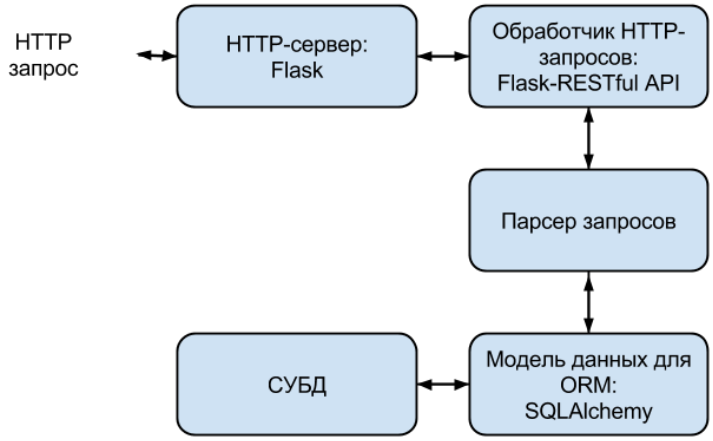

Рисунок 5.1: Архитектура сервиса хранения состояний

5.1.2 Сервис расчета порядка

Сервис расчета порядка представляет собой расчетный сервис для поиска оптимального порядка действий для достижения одного из конечных состояний ТА-модели. В качестве входных данных используется объект ТА-модели, переданный сервисом хранения состояний. Выходными данными является список действий для перехода из начального состояния автомата в одно из конечных состояний.

В качестве расчетной основы используется библиотека NetworkX, реализующая высокоэффективную модель графа и алгоритм поиска A*.

Сервис расчета порядка имеет следующие методы REST API (см. таблицу 5.2).

Таблица 5.2: Методы API сервиса расчета порядка

Адрес	HTTP-метод	Выполняемое действие
api/pathfinder	POST	Найти оптимальные пути в автомате (описание автомата в JSON-данных запроса)

Пример работы API приведен в приложении А.2.

Архитектура сервиса представлена на рис. 5.2. Поступающие HTTP-запросы обрабатываются HTTP-сервером фреймворка Flask. Расширение Flask-RESTful API маршрутизирует запросы на парсер автомата. Парсер автомата преобразует JSON-представление

автомата в объект графа библиотеки NetworkX. Затем вызывается алгоритм поиска A* для объекта графа. Полученный результат возвращается в HTTP-ответе.

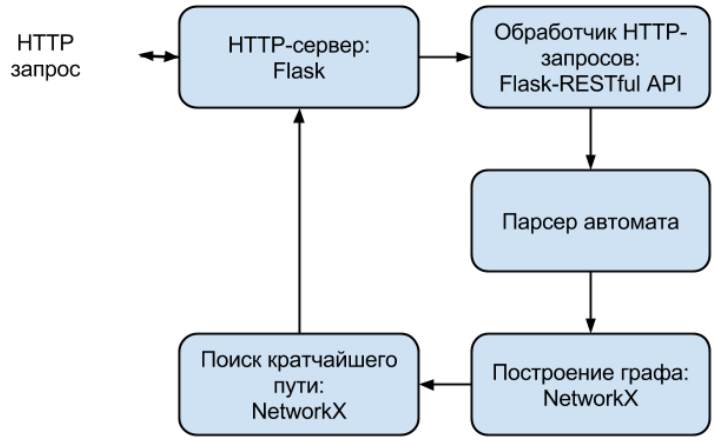

Рисунок 5.2: Архитектура сервиса расчета порядка

5.1.3 Сервис моделирования состояний

Сервис моделирования состояний представляет собой набор искусственных нейронных сетей и их обучающие выборки. Пользователь может создать нейронную сеть, указав параметры (количество слоев, количество нейронов в каждом слое и функции активации нейронов) и задав обучающую выборку. После обучения на вход ИНС можно подать входной вектор (в том числе ТА-модель) и получить отклик, который соответствует аппроксимированному значению обучающей выборки.

В качестве расчетной основы используется библиотека PyBrain, реализующая различные модели искусственных нейронных сетей, а также хранилище обучающих выборок и алгоритмы обучения.

Сервис моделирования состояний имеет следующие методы REST API (см. таблицу 5.3).

Таблица 5.3: Методы API моделирования состояний

Адрес	HTTP-метод	Выполняемое действие
api/neuronnetwork	POST	Задать нейронную сеть и обучающую выборку (описание в JSON-данных запроса)
api/neuronnetwork	GET	Получить отклик нейронной сети (входной вектор в данных запроса)

Архитектура сервиса представлена на рис. 5.3. Поступающие HTTP-запросы обрабатываются HTTP-сервером фреймворка Flask. Расширение Flask-RESTful API маршрутизирует запросы на парсер нейронной сети или на уже обученную нейронную сеть. Парсер нейронной преобразует JSON-представление в объект нейронной сети библиотеки PyBrain и его обучающую выборку. При этом производится обучение нейронной сети до конвергенции. Для обученной сети может быть получен отклик на входной вектор. Полученный результат возвращается в HTTP-ответе.

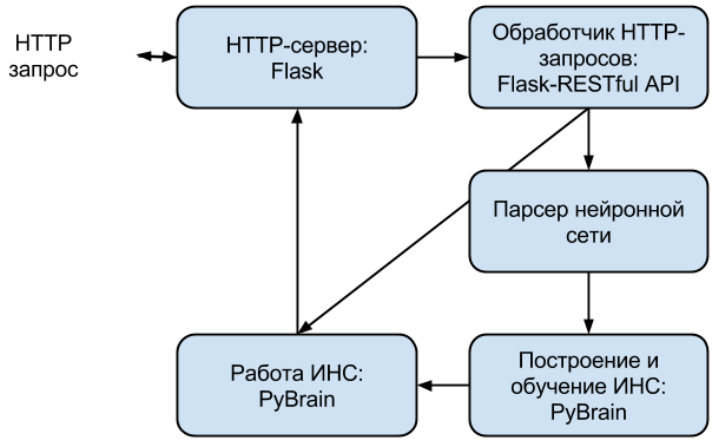

Рисунок 5.3: Архитектура сервиса моделирования состояния

5.2 Технологии, используемые при разработке программного комплекса

5.2.1 Python

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Питоне организовывается в функции и классы, которые могут объединяться в модули (они в свою очередь могут быть объединены в пакеты).

Python портирован и работает почти на всех известных платформах — от КПК до мейнфреймов. Существуют порты под Microsoft Windows, практически все варианты UNIX (включая FreeBSD и Linux), Plan 9, Mac OS и Mac OS X, iPhone OS 2.0 и выше, Palm OS, OS/2, Amiga, HaikuOS, AS/400 и даже OS/390, Windows Mobile, Symbian и Android.

Python — стабильный и распространённый язык. Он используется во многих проектах и в различных качествах: как основной язык программирования или для создания расширений и интеграции приложений. На Python реализовано большое количество проектов, также он активно используется для создания прототипов будущих программ. Python используется во многих крупных компаниях. Python с пакетами NumPy, SciPy и MatPlotLib активно используется как универсальная среда для научных расчётов в качестве замены распространенным специализированным коммерческим пакетам Matlab, IDL и др. [20]

5.2.2 Flask

Flask является микрофреймворком для создания веб-приложений на языке Python «Микро» не означает, что ваше веб-приложение целиком помещается в один файл с кодом на Python, хотя конечно же это может быть и так. Также, это не означает, что Flask ис-

пытывает недостаток функциональности. «Микро» в слове «микрофреймворк» означает, что Flask стремится придерживаться простого, но расширяемого ядра. Flask не может решить многие вещи, например, какую базу данных использовать. А те решения, которые он может принять, например, который из движков для работы с шаблонами использовать, легко изменить.

По умолчанию, Flask не включает уровень абстракции баз данных, валидации форм или каких-то иных, для чего уже существуют различные занимающиеся этим библиотеки. Вместо этого, Flask поддерживает расширения для добавления подобной функциональности в ваше приложение, таким образом, как если бы это было реализовано в самом Flask. Многочисленные расширения обеспечивают интеграцию с базами данных, валидацию форм, обработку загрузок на сервер, различные открытые технологии аутентификации и так далее. Flask может быть «микро», но при этом он готов для использования в реальных задачах для самых разнообразных нужд.

5.2.3 REST

REST (сокр. англ. Representational State Transfer, «передача состояния представления» или «передача репрезентативного состояния») — стиль построения архитектуры распределенного приложения. Был описан и популяризован в 2000 году Роем Филдингом (Roy Fielding), одним из создателей протокола HTTP. Самой известной системой, построенной в значительной степени по архитектуре REST, является современная Всемирная паутина.

Данные в REST должны передаваться в виде небольшого количества стандартных форматов (например HTML, XML, JSON). Сетевой протокол (как и HTTP) должен поддерживать кэширование, не должен зависеть от сетевого слоя, не должен сохранять информацию о состоянии между парами «запрос-ответ». Утверждается, что такой подход обеспечивает масштабируемость системы и позволяет ей эволюционировать с новыми требованиями.

5.2.4 SQLAlchemy

SQLAlchemy — это программная библиотека на языке Python для работы с реляционными СУБД с применением технологии ORM. Служит для синхронизации объектов Python и записей реляционной базы данных. SQLAlchemy позволяет описывать структуры баз данных и способы взаимодействия с ними на языке Python без использования SQL. Библиотека была выпущена в феврале 2006 под лицензией открытого ПО MIT.

Использование SQLAlchemy для автоматической генерации SQL-кода имеет несколько преимуществ по сравнению с ручным написанием SQL:

- Безопасность. Параметры запросов экранируются, что делает атаки типа внедрение SQL-кода маловероятными.

- Производительность. Повышается вероятность повторного использования запроса к серверу базы данных, что может позволить ему в некоторых случаях применить повторно план выполнения запроса.

- Переносимость. SQLAlchemy, при должном подходе, позволяет писать код на Python, совместимый с несколькими back-end СУБД. Несмотря на стандартизацию языка SQL, между базами данных имеются различия в его реализации, абстрагироваться от которых и помогает SQLAlchemy. [21]

5.2.5 NetworkX

Библиотека networkX создана на языке Python и предназначена для работы с графами и другими сетевыми структурами. Это свободное ПО распространяемое под новой BSD лицензией.

Основные возможности библиотеки:

- Классы для работы с простыми, ориентированными и взвешенными графами;
- Узлом может быть практически что угодно: time-series, текст, изображение, XML;
- Сохранение / загрузка графов в/из наиболее распространённых форматов файлов хранения графов;
- Встроенные процедуры для создания графов базовых типов;
- Методы для обнаружения подграфов, клик и K-дольных графов (K-core) (максимальный подграф в котором каждая вершина имеет по крайней мере уровень K).
- Получение таких характеристик графа как степени вершин, высота графа, диаметр, радиус, длинны путей, центр, промежуточности, и т. д.;
- Визуализировать сети в виде 2D и 3D графиков.

Заявляется, что библиотека свободно может оперировать весьма большими сетевыми структурами, уровня графа с 10 миллионами узлов и 100 миллионами дуг между ними. В виду того, что он базируется на низкоуровневой структуре данных языка Python под названием «словарь-словарей», память расходуется эффективно, графы хорошо масштабируются, мало зависят от особенностей операционной системы в которой выполняется скрипт и отлично подходят для популярного на данный момент направления по анализу данных из социальных сетей и графов. [22]

5.2.6 PyBrain

PyBrian представляет собой модульную библиотеку предназначенную для реализации различных алгоритмов машинного обучения на языке Python. Основной его целью является предоставление исследователю гибких, простых в использовании, но в то же время мощных инструментов для реализации задач из области машинного обучения, тестирования и сравнения эффективности различных алгоритмов. Название PyBrain является аббревиатурой от английского: Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library. Как сказано на одном сайте: PyBrain — это швейцарский армейский нож в области нейро-сетевых вычислений.

Библиотека построена по модульному принципу, что позволяет использовать её как студентам для обучения основам, так и исследователям, нуждающимся в реализации более сложных алгоритмов. Общая структура процедуры её использования приведена на следующей схеме (см. рис. 5.4). [23]

5.2.7 Облачные вычисления

Облачные вычисления (англ. cloud computing)— это модель обеспечения повсеместного и удобного сетевого доступа по требованию к общему пулу (англ. pool) конфигурируемых

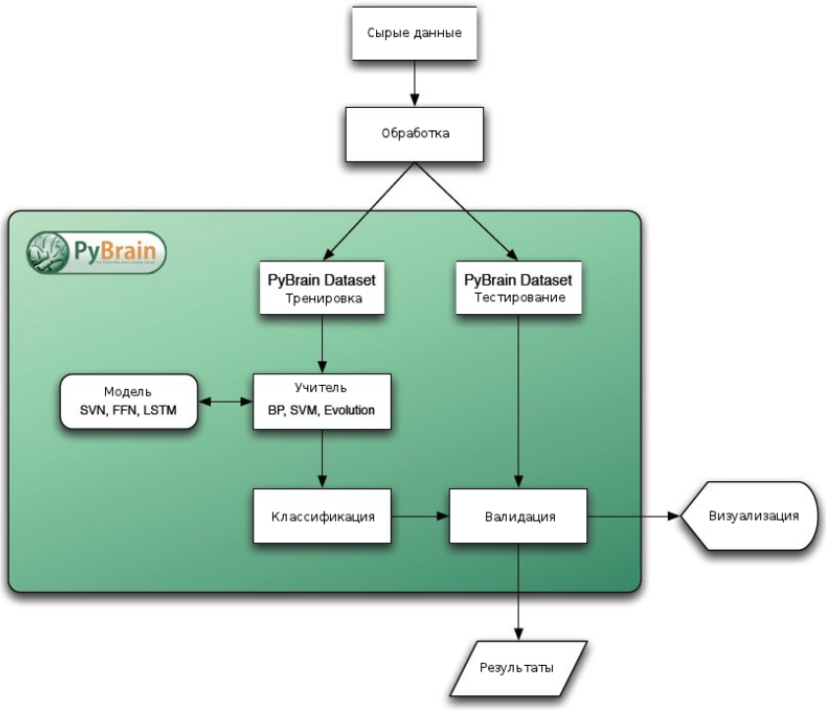

Рисунок 5.4: Процедура использования библиотеки PyBrain

вычислительных ресурсов (например, сетям передачи данных, серверам, устройствам хранения данных, приложениям и сервисам — как вместе, так и по отдельности), которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами и/или обращениями к провайдеру.

Потребители облачных вычислений могут значительно уменьшить расходы на инфраструктуру информационных технологий (в краткосрочном и среднесрочном планах) и гибко реагировать на изменения вычислительных потребностей, используя свойства вычислительной эластичности (англ. elastic computing) облачных услуг.

Платформа как услуга (PaaS, англ. PaaS or Platform as a Service) — модель предоставления облачных вычислений, при которой потребитель получает доступ к использованию информационно-технологических платформ: операционных систем, систем управления базами данных, связующему программному обеспечению, средствам разработки и тестирования, размещённым у облачного провайдера. В этой модели вся информационно-технологическая инфраструктура, включая вычислительные сети, серверы, системы хранения, целиком управляется провайдером, провайдером же определяется набор доступных для потребителей видов платформ и набор управляемых параметров платформ, а потребителю предоставляется возможность использовать платформы, создавать их виртуальные экземпляры, устанавливать, разрабатывать, тестировать, эксплуатировать на них прикладное программное обеспечение, при этом динамически изменяя количество потребляемых вычислительных ресурсов.

Провайдер облачной платформы может взимать плату с потребителей в зависимости от уровня потребления, тарификация возможна по времени работы приложений потребителя, по объёму обрабатываемых данных и количеству транзакций над ними, по сетевому трафику. Провайдеры облачных платформ достигают экономического эффекта за счёт использования виртуализации и экономии на масштабах, когда из множества потребителей в одно и то же время лишь часть из них активно использует вычислительные ресурсы, потребители — за счёт отказа от капитальных вложений в инфраструктуру и платфор-

мы, рассчитанных под пиковую мощность и непрофильных затрат на непосредственное обслуживание всего комплекса.

Инфраструктура как услуга (IaaS, англ. IaaS or Infrastructure-as-a-Service) предоставляется как возможность использования облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетями и другими фундаментальными вычислительными ресурсами, например, потребитель может устанавливать и запускать произвольное программное обеспечение, которое может включать в себя операционные системы, платформенное и прикладное программное обеспечение. Потребитель может контролировать операционные системы, виртуальные системы хранения данных и установленные приложения, а также ограниченный контроль набора доступных сервисов (например, межсетевой экран, DNS). Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, типов используемых операционных систем, систем хранения осуществляется облачным провайдером. [24]

Заключение

Основные результаты работы заключаются в следующем.

1. На основе исследований конструкции и порядка планово-профилактического ремонта реактора была предложена формальная информационная модель реактора. Отличительной особенностью предложенной модели является представление узлов и агрегатов реактора в виде автоматов. Это позволяет описывать процесс перезагрузки с любой детализацией, используя идентичные информационные структуры. Таким образом данная методология может быть применена к любому типу реакторов.

2. Исследование методов поиска кратчайшего пути в графах позволили предложить метод построения оптимального порядка перезагрузки. Модель реактора на основе теории автоматов (ТА-модель) допускает представление в виде графа, что позволяет найти путь от начального состояния автомата до конечного методами теории графов. Было предложено использовать информированный поиск на основе алгоритма A*. Задавая соответствующую эвристическую функцию поиска можно производить оптимизацию по различным критериям, не изменяя модели и методологии поиска.

3. Для предварительного моделирования физических параметров активной зоны было предложено использовать искусственную нейронную сеть. Данный подход требует предварительного обучения на множестве значений, однако позволяет избежать длительных расчетов на этапе поиска пути. Обучающая выборка может состоять не только из данных, рассчитанных по математическим моделям реактора, но и реальные данные предыдущих кампаний. Что в свою очередь позволяет учесть погрешности математических моделей.

4. Для выполнения поставленных задач был разработан программный комплекс. Разработанный комплекс позволяет создавать и хранить различные ТА-модели, находить оптимальный путь с заданной эвристикой, а также моделировать некоторые параметры состояний ТА-модели при наличии обучающей выборки. Данный программный комплекс в полной мере иллюстрирует действие положений настоящей диссертации.

Таким образом, предложенная методология расчета порядка перезагрузки реактора доказала свою перспективность. Однако, для использования в атомной промышленности потребуется ряд серьезных испытаний и доработок, которые позволят повысить точность и производительность методологии.

Список рисунков

Список таблиц

Литература

1. АЭС с реактором типа ВВЭР-1000. От физических основ эксплуатации до эволюции проекта. / С.А. Андрушечко, А.М. Арфов, Б.Ю. Васильев [и др.]. М.: Логос, 2010. С. 604. ISBN 978-5-98704-496-4.

2. И.П. Белявцев. Метод расчета оптимального порядка замены тепловыделяющих сборок реакторов класса БН с использованием методов искусственного интеллекта // XII Межрегиональная научно-техническая конференция студентов и аспирантов «Применение кибернетических методов в решении проблем общества XXI века». 2014.

3. Г.Б. Усынин, Кусмарцев. Реакторы на быстрых нейтронах: Учеб. пособие для вузов. М.: Энергоатомиздат, 1985. С. 288.

4. И.Р. Акишев, М.Э. Дворкин. О построении минимальных детерминированных конечных автоматов, распознающих префиксный код заданной мощности // Прикладная дискретная математика. 2010.

5. Дж. Хопкрофт, Р. Мотвани, Дж. Ульман. Введение в теорию автоматов, языков и вычислений = Introduction to Automata Theory, Languages, and Computation. М.: Вильямс, 2002. С. 528. ISBN 0-201-44124-1.

6. Богаченко Н.Ф. Файзуллин Р.Т. Синтез дискретных автоматов: Учебное пособие. Омск: Издательство Наследие. Диалог-Сибирь, 2006. С. 150.

7. А.М. Лупал. Теория автоматов: Учебное пособие. СПб.: ГУАП, 2000. С. 119.

8. Ф. Харари. Теория графов / под ред. Г.П. Гаврилова. Издательство мир, 1973. С. 300 с. ISBN 5-354-00301-6.

9. В.А. Евстигнеев. Применение теории графов в программировании / под ред. А.П. Ершова. Москва: Наука. Главная редакция физико-математической литературы, 1985. С. 352 с.

10. А. Сплитфейс. Алгоритм Дейкстры. Поиск оптимальных маршрутов на графе. URL: http://habrahabr.ru/post/111361/.

11. С.Дж. Рассел, П. Норвиг. Искусственный интеллект: современный подход = Artificial Intelligence: A Modern Approach / Пер. с англ. и ред. К. А. Птицына. — 2-е изд. М.: Вильямс, 2006. С. 157—162. ISBN 5-8459-0887-6.

12. Д. Платонов. Поиск пути: алгоритм A* для новичков
. URL: http://savepearlharbor.com/?p=159417.

13. Л.Б. Абрайтис. Автоматизация проектирования топологии цифровых интегральных микросхем. М.:Радио и связь, 1985. С. 200.

14. Г.Я. Румянцев. Расчет ядерного реактора на тепловых нейтронах. М.: Атомиздат, 1967. С. 117.

15. А.Н. Горбань. Обобщенная аппроксимационная теорема и вычислительные возможности нейронных сетей // Сибирский журнал вычислительной математики. 1998. С. 12—24.

16. Т.В. Филатова. Применение нейронных сетей для аппроксимации данных // Вестник Томского государственного университета. 2004. Т. 284.

17. И.В. Заенцев. Нейронные сети: основные модели. Воронеж: «Воронеж», 1999. С. 76.

18. А. Джейн, Ж. Мао. Введение в искусственные нейронные сети // Открытые системы. 1996.

19. С. Хайкин. Нейронные сети: полный курс, 2-е издание : Пер. с англ. М.: Издательский дом «Вильямс», 2006. С. 1104.

20. М. Лутц. Программирование на Python / Пер. с англ. СПб.: Символ-Плюс, 2011.

21. Copeland R. Essential SQLAlchemy. O'Reilly Media, 2008. P. 215.

22. П. Осипов. NetworkX для удобной работы с сетевыми структурами . URL: http://habrahabr.ru/post/125898/.

23. PyBrain / Tom Schaul, Justin Bayer, Daan Wierstra [и др.] // Journal of Machine Learning Research. 2010.

24. Gillam Lee. Cloud Computing: Principles, Systems and Applications. L.: Springer, 2010. С. 379.

Приложение А

Примеры работы программного комплекса

А.1 Сервис хранения порядка

Пример ответа на запрос curl GET api/automata/1:


```
{
 "states": [
   {
     "data": "{\"size\": \"10\"}",
     "id": 1,
     "name": "q1"
   },
   {
     "data": "{\"size\": \"20\"}",
     "id": 2,
     "name": "q2"
   },
   {
     "data": "{\"size\": \"30\"}",
     "id": 3,
     "name": "q3"
   },
   {
     "data": "{\"size\": \"40\"}",
     "id": 4,
     "name": "q4"
   }
 ],
 "start_state": {
   "data": "{\"size\": \"10\"}",
   "id": 1,
   "name": "q1"
 },
 "end_states": [
   {
     "data": "{\"size\": \"20\"}",
     "id": 2,
     "name": "q2"
   },
   {
     "data": "{\"size\": \"30\"}",
     "id": 3,
     "name": "q3"
   },
   {
     "data": "{\"size\": \"40\"}",
```

```
      "id": 4,
      "name": "q4"
    }
  ],
  "transitions": [
    {
      "action": {
        "data": "{\"time\": \"10\"}",
        "id": 1,
        "name": "n1"
      },
      "start": {
        "data": "{\"size\": \"10\"}",
        "id": 1,
        "name": "q1"
      },
      "end": {
        "data": "{\"size\": \"20\"}",
        "id": 2,
        "name": "q2"
      },
      "id": 1
    },
    {
      "action": {
        "data": "{\"time\": \"20\"}",
        "id": 2,
        "name": "n2"
      },
      "start": {
        "data": "{\"size\": \"20\"}",
        "id": 2,
        "name": "q2"
      },
      "end": {
        "data": "{\"size\": \"30\"}",
        "id": 3,
        "name": "q3"
      },
      "id": 2
    },
    {
      "action": {
        "data": "{\"time\": \"10\"}",
        "id": 1,
        "name": "n1"
      },
      "start": {
        "data": "{\"size\": \"10\"}",
        "id": 1,
        "name": "q1"
      },
      "end": {
        "data": "{\"size\": \"20\"}",
        "id": 2,
        "name": "q2"
      },
      "id": 3
    },
    {
```

```
      "action": {
        "data": "{\"time\": \"20\"}",
        "id": 2,
        "name": "n2"
      },
      "start": {
        "data": "{\"size\": \"20\"}",
        "id": 2,
        "name": "q2"
      },
      "end": {
        "data": "{\"size\": \"30\"}",
        "id": 3,
        "name": "q3"
      },
      "id": 4
    },
    {
      "action": {
        "data": "{\"time\": \"10\"}",
        "id": 1,
        "name": "n1"
      },
      "start": {
        "data": "{\"size\": \"10\"}",
        "id": 1,
        "name": "q1"
      },
      "end": {
        "data": "{\"size\": \"20\"}",
        "id": 2,
        "name": "q2"
      },
      "id": 5
    },
    {
      "action": {
        "data": "{\"time\": \"20\"}",
        "id": 2,
        "name": "n2"
      },
      "start": {
        "data": "{\"size\": \"20\"}",
        "id": 2,
        "name": "q2"
      },
      "end": {
        "data": "{\"size\": \"30\"}",
        "id": 3,
        "name": "q3"
      },
      "id": 6
    },
    {
      "action": {
        "data": "{\"time\": \"10\"}",
        "id": 1,
        "name": "n1"
      },
      "start": {
```

```
        "data": "{\"size\": \"10\"}",
        "id": 1,
        "name": "q1"
      },
      "end": {
        "data": "{\"size\": \"20\"}",
        "id": 2,
        "name": "q2"
      },
      "id": 7
    },
    {
      "action": {
        "data": "{\"time\": \"20\"}",
        "id": 2,
        "name": "n2"
      },
      "start": {
        "data": "{\"size\": \"20\"}",
        "id": 2,
        "name": "q2"
      },
      "end": {
        "data": "{\"size\": \"30\"}",
        "id": 3,
        "name": "q3"
      },
      "id": 8
    },
    {
      "action": {
        "data": "{\time\": \"40\"}",
        "id": 3,
        "name": "n3"
      },
      "start": {
        "data": "{\"size\": \"10\"}",
        "id": 1,
        "name": "q1"
      },
      "end": {
        "data": "{\"size\": \"40\"}",
        "id": 4,
        "name": "q4"
      },
      "id": 9
    },
    {
      "action": {
        "data": "{\"time\": \"10\"}",
        "id": 1,
        "name": "n1"
      },
      "start": {
        "data": "{\"size\": \"10\"}",
        "id": 1,
        "name": "q1"
      },
      "end": {
        "data": "{\"size\": \"20\"}",
```

```
        "id": 2,
        "name": "q2"
      },
      "id": 10
    },
    {
      "action": {
        "data": "{\"time\": \"20\"}",
        "id": 2,
        "name": "n2"
      },
      "start": {
        "data": "{\"size\": \"20\"}",
        "id": 2,
        "name": "q2"
      },
      "end": {
        "data": "{\"size\": \"30\"}",
        "id": 3,
        "name": "q3"
      },
      "id": 11
    },
    {
      "action": {
        "data": "{\time\": \"40\"}",
        "id": 3,
        "name": "n3"
      },
      "start": {
        "data": "{\"size\": \"10\"}",
        "id": 1,
        "name": "q1"
      },
      "end": {
        "data": "{\"size\": \"40\"}",
        "id": 4,
        "name": "q4"
      },
      "id": 12
    }
  ],
  "id": 1,
  "actions": [
    {
      "data": "{\"time\": \"10\"}",
      "id": 1,
      "name": "n1"
    },
    {
      "data": "{\"time\": \"20\"}",
      "id": 2,
      "name": "n2"
    },
    {
      "data": "{\time\": \"40\"}",
      "id": 3,
      "name": "n3"
    }
  ],
```

```
  "name": "test1"
}
```

A.2 Сервис расчета порядка

Пример ответа на запрос curl POST api/pathfinder с запросом из A.1:

```
[
    {
        "length": 10,
        "path": [
            "q1",
            "q2"
        ]
    },
    {
        "length": 30,
        "path": [
            "q1",
            "q2",
            "q3"
        ]
    },
    {
        "length": 30,
        "path": [
            "q1",
            "q4"
        ]
    }
]
```

Приложение В

Исходные коды программного комплекса

В.1 Сервис хранения порядка

В.1.1 Дескрипторы моделей

В.1.2 REST-сервис

В.2 Сервис расчета порядка

В.2.1 Дескрипторы контроллера

В.2.2 REST-сервис

В.3 Сервис моделирования состояния

В.3.1 REST-сервис

В.4 Утилиты

В.4.1 Скрипт запуска приложения

В.4.2 Скрипты миграции базы данных